

On Development of Systems for Number
Theoretic Calculation and Related Problems

MATSUI Tetsushi

Contents

1	Introduction	5
1.1	Abstract	5
1.2	Outline	7
1.3	Acknowledgment	8
2	SIMATH	9
2.1	Introduction	9
2.2	Internal Structure	10
2.2.1	SIMATH	10
2.2.2	simcalc	10
2.3	New Features	13
2.3.1	Supporting 64-bit Systems	13
2.3.2	Cycho-Elliptic Method	14
2.4	For Future Development	15
3	NZMATH	17
3.1	Introduction	17
3.2	Rationale	18
3.2.1	Problems in SIMATH	18
3.2.2	Python	19
3.2.3	Resolutions of Problems	20
3.2.4	Other Merits and Demerits	21
3.3	History and Current Status	22
3.3.1	History	22
3.3.2	Current Status	22
3.3.3	Modules for Basic Data Types	22
3.3.4	Number Theoretic Modules	23
3.3.5	Miscellaneous Modules	24
3.3.6	Manual	25
3.3.7	Applications	26
3.4	Development	26
3.4.1	User / Developer Fusion	26
3.4.2	Speed of Development	26
3.4.3	Open Source	27

3.4.4	Agile Development	28
3.5	Future of NZMATH	29
3.5.1	Algebraic Number Field	29
3.5.2	Web UI	29
3.5.3	Speed of Execution	30
3.5.4	Long Term Plans	30
3.6	Conclusions	31
4	Related Problems	33
4.1	Introduction	33
4.2	Bulk Quantum Turing Machine	34
4.2.1	Introduction	34
4.2.2	Definitions	34
4.2.3	Complexity Classes	36
4.2.4	Conclusion	40
4.3	Polynomial Time Computable Numbers	40
4.3.1	Introduction	40
4.3.2	Definitions	41
4.3.3	The Field of Polynomial Time Computable Numbers	45
4.3.4	Main Theorem	48
4.3.5	Transcendental Numbers	55
4.3.6	Concluding Remarks	57

Chapter 1

Introduction

1.1 Abstract

With the evolution of computers in twentieth century, mathematics also has been influenced and it seems to regain its experimental nature with rich example calculations. Number theory is not an exception; there have been many algorithms and large tables of numbers. The systems for number theoretic calculation are softwares to carry out such computations. The most well-known systems include PARI or KANT. SIMATH is also one of the most well-known systems, whose development had been led by H. G. Zimmer of Universität des Saarlandes in Germany. The development and maintenance had been taken over by K. Nakamura of Tokyo Metropolitan University in 2002 on time of Zimmer's retirement. The author took a part of the development in Nakamura's group; he port the software to a 64 bit environment Tru64 Unix system.

On 2003, when the development of SIMATH unfortunately stopped, the author proposed a new system of number theoretic calculation based on the experience with SIMATH, and started its development; it is NZMATH. NZMATH is implemented in a scripting language Python. There is no similar project to implement a system of number theoretic calculation with a scripting language, and it is the only system developed in Japan. The scripting language Python provides arbitrary precision integer as a basic data type, automated garbage collection and object-oriented mechanisms. Comparing to implementing in languages like C, these features reduce total amount of developers' works to do, thus it is helpful for speeding up the development.

There are two major visions of NZMATH development. One is user / developer fusion, and speed of development is another. On the one hand, the developers of systems of number theoretic calculation are limited to mathematicians by nature. Therefore, the other things but mathematical issues, such as memory management or designing and implementing an interpreter, are the tasks that seem too much. It is better, if possible, for developers to concentrate on mathematical issues avoiding unwanted tasks. On the other hand, useful

programs written by users can make the system evolution faster by merging them into the system itself. To realize it, it is the shortest path that both users and developers program with the same language. The main purpose of systems for number theoretic calculation is to provide calculation functions useful for researches, and the shorter time of computation is the better. It should be, however, noted that the time of computation means the total time including installation, programming and execution. Therefore, it is not a good strategy to speed up the last execution step only. In our vision, NZMATH should speed up pre-execution steps to be more useful.

As the result of development for about three years, NZMATH system provides basic data types such as rational numbers, finite fields, polynomials, etc., functions of elementary number theory such as integer factorization, primality test, etc., and number theoretic features, especially elliptic curves. The preparation for algebraic number data types is in progress now, and calculation of class numbers of imaginary quadratic fields is available.

In the future, NZMATH will provide algebraic numbers and number fields with their invariants calculation, p -adic numbers, group rings, and so on. Beside to the development, it is thought to be important to spread NZMATH into mathematical community so that research results using NZMATH will be popular.

During the development, computational complexity is always one of the most important concerns. Relating to it, the author studies two subjects. One is an investigation of a model of quantum computer using nuclear magnetic resonance (NMR) device, by which the most progressive result toward realizing quantum computer is achieved recently. Another is a proof of the algebraically closedness of the set of polynomial time computable numbers, which is the set of complex numbers reasonable to be provided by a system for mathematical calculation.

The quantum computation with NMR is characterized by the mean to obtain a computing result; a bunch of molecules each considered quantum computers are measured as a whole to yield an average value. The feature corresponds to a limitation to the model of quantum computer, and it is a question whether it makes a difference of computational capability or not. Previously, Atsumi and Nishino proposed a model called bulk quantum Turing machine, but it has incompatibility with the principle of quantum mechanics. Thus, we define a modified, quantum mechanically correct model.

Definition 1 (Tape, Symbols and Unitarity). A quantum Turing machine (QTM) (or bulk quantum Turing machine (BQTM)) \mathcal{M} is defined by a triplet (Σ, Q, δ) where: Σ is a finite alphabet with an identified blank symbol $\#$, Q is a finite set of states with an identified initial state q_0 and final state q_f which is not equal to q_0 , and δ is a function called quantum transition function:

$$\delta : Q \times \Sigma \rightarrow \mathbb{C}_c^{\Sigma \times Q \times D} \quad (1.1)$$

where D denotes directions $\{L, R\}$ or $\{L, N, R\}$, and \mathbb{C}_c denotes the set of computable numbers. The QTM (or BQTM) has a two-way infinite tape of cells indexed by \mathbb{Z} and a single tape head that moves along the tape.

The quantum transition function δ induces a time evolution operator $U_{\mathcal{M}}$ of the inner-product space of finite complex linear combination of configurations of \mathcal{M} . The time evolution operator $U_{\mathcal{M}}$ must be unitary.

Definition 2 ((ϵ, θ) -measurement). An (ϵ, θ) -measurement is an observation such that $|\alpha|^2 - |\beta|^2$ is obtained for a qubit $\alpha|1\rangle + \beta|0\rangle$ with error range less than θ and error probability less than ϵ . If either $|\alpha|^2$ or $|\beta|^2$ is zero, the error probability is exceptionally zero. The measurement does disturb the superposition which \mathcal{M} is in, and can not be repeated multiple times. No partial observation is allowed.

We prove the modified bulk quantum Turing machine essentially defined by (ϵ, θ) -measurement shows the equivalent computational power with ordinary quantum Turing machines for decision problems.

On the subject of computable numbers, first of all, recall that there exist uncountably many complex numbers but there only exist countably many definable numbers. It is, therefore, impossible to handle all complex numbers on any computers. From Turing, the computable numbers each defined by a computable function which produces its approximation sequence are considered and it is known by Rice that the set of all computable numbers forms an algebraically closed field. The result above is excellent but not perfect, because only polynomial time computable functions are generally considered feasible. It is, thus, natural to investigate complex numbers given by approximation sequences computed by polynomial time computable functions as a feasible set of numbers. We define polynomial time computable numbers as follows.

Definition 3. A *polynomial time computable number* is a complex number z that there exist two polynomial time computable functions f and g from \mathbb{N} to \mathbb{Z} such that

$$\left| z - \frac{f(n) + g(n)i}{n} \right| \leq \frac{1}{n}$$

is satisfied for any natural number n greater than 1.

We let $\mathbb{C}_{\mathcal{P}}$ denote the set of all polynomial time computable numbers. We prove the following result.

Theorem 1. $\mathbb{C}_{\mathcal{P}}$ is an algebraically closed field.

There are transcendental numbers in the set of polynomial time computable numbers such as the circle ratio π , thus the field of polynomial time computable numbers is a proper subfield of the field of computable numbers $\mathbb{C}_{\mathcal{C}}$ and a proper extension of the field of algebraic numbers $\overline{\mathbb{Q}}$.

1.2 Outline

The organization of the thesis is the following. The first chapter is about SIMATH. The second chapter is the main part of the thesis about NZMATH.

The third chapter consists of two sections: the first section is about modified bulk quantum Turing machine and the second is about polynomial time computable numbers.

1.3 Acknowledgment

I express my sincere thanks to who gives us explicit or implicit helps: Kobayashi Daisuke and Abe Mizuho for discussions about the first chapter. Associate professor Tsumura Hirofumi, associate professor Uchiyama Shigenori, Hirano Takato and Nishimoto Keiichiro for helpful discussions and comments, the anonymous referee of ICMS for the kind suggestions for the original paper submitted to ICMS [23]. The members of the `NZMATH` development group and other contributors for their efforts of restless improvement of the software. And last but not least, we express sincere thanks to my supervisor Professor Nakamura Ken for encouraging me to work on `NZMATH` and for his kind comments and discussions; especially for letting me know the theorem of Ostrowski in section 4.3.

Chapter 2

SIMATH

2.1 Introduction

The first system development we had involved is of **SIMATH**. **SIMATH** is a computer algebra system for algebraic number theory. From the beginning of its history it has been developed at Universität des Saarlandes in Saarbrücken, Germany [11]. The first machines on which the development started were SINIX PCs and the name of the system is derived from them [39]. Unfortunately, the development had been almost stopping for a few years. Then, the development center moved to TMU (Tokyo Metropolitan University) in Tokyo, Japan in 2002. **SIMATH** is highly integrated and easy to extend with a developing environment **SM**. There is also an interpreter interface called `simcalc`. See Section 2.2 for information about this internal structure.

In this chapter, we summarize our works after we took over the system, and propose some problems for future development. The first outcome is support for 64-bit Unices like Tru64 Unix on alpha. **SIMATH** has been supporting many 32-bit Unices: HP-UX, IRIX, Sun OS, x86 Linux, etc. We have successfully ported it to 64-bit environment. Another new feature is CEM (Cyclo-Elliptic Method) [25]. **SIMATH** has many functions, especially for elliptic curves. On the other hand, we can hardly say that it has sufficiently many system functions for algebraic number fields. As the first step to enrich those functions, we add the CEM which compute the class number and fundamental units for a subfield of an abelian extension over an imaginary quadratic number field. See Section 2.3 for detailed description of these new features.

SIMATH has been mainly developed by a rather small group in one university. In order to keep continuous and stable development of the system, we believe it is better to make the group independent of one person or one small organization. We start to use CVS (Concurrent Versions System) [8] for open source development of the system. We are intending to open and extend the group to a wider range including the former development group. We shall discuss this topic in Section 2.4

2.2 Internal Structure

2.2.1 SIMATH

SIMATH is a highly integrated system with the programming language C. Users can extend it without learning new syntax but with only the knowledge of C and some easy rules of SIMATH itself.

The deepest construct of the system is a list system. It has own garbage collection functions. The list system is used to construct most of the SIMATH data types, and it hides memory management from users.

On this basis, basic types are defined: integers, rational numbers, polynomials, matrices, vectors, elliptic curves, etc.... Then packages for algebraic number theory are constructed. The figure on the next page is taken from the manual [39].

It is easily seen that SIMATH can compute over many kinds of field, and there are various usable functions.

The points one has to care when he/she writes programs in SIMATH are:

- Write “#include <_simath.h>” on the top of the source code.
- Use original function “init()” or “bind()” always with SIMATH variables. It enables SIMATH to keep them for automatic garbage collection.

One needs to take special care of the use *printf()*. The user should get information about it from the online help of SM, which is a development environment of SIMATH, with command “@”.

SM also has commands related to compilation, “a”, “o”, “+”, etc. Using these commands on SM prompt, users will be free from troubles of other than the mathematics or algorithm when compiling their programs.

2.2.2 simcalc

The SIMath CALCulator *simcalc* is an interactive system, which is an application software on SIMATH foundation. No knowledge of programming language C is required for using it. The user can use operators instead of SIMATH function calls for addition, subtraction, etc. Variables in which computational results can be stored are available. There are many useful programs prepared into *simcalc*. The user, thus, can easily compute various mathematical expressions.

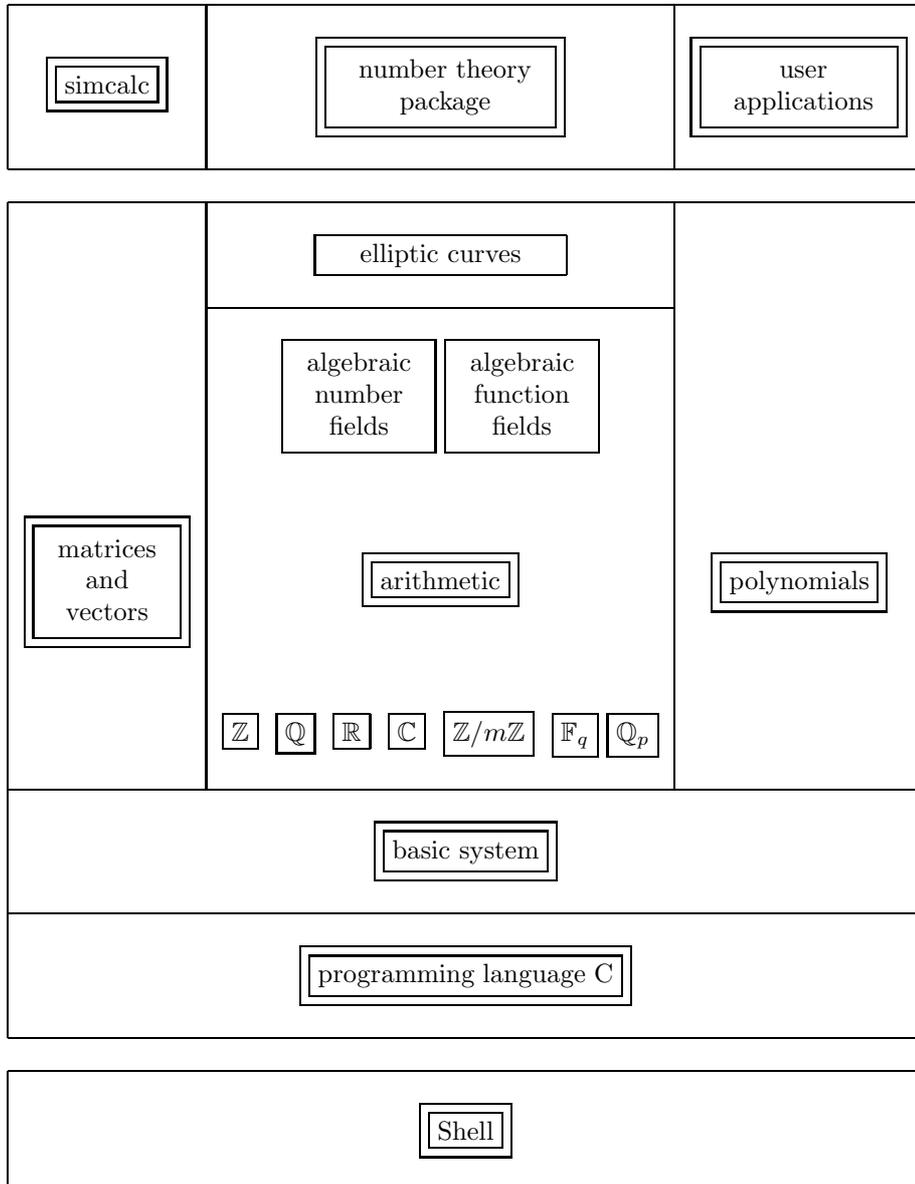
The following example shows how a session looks like, and gives a help to get detail¹.

To compute $(1 + 2) * (3 + 4) - 5$ on *simcalc*, simply input them.

```
> (1+2)*(3+4)-5
@ = 16
```

The user can use variables with any names except ones which are already used by the system. The length of variable name is at most 20.

¹> is the prompt symbol of *simcalc*.



```
> X1234567890123456789 = 1024
      X1234567890123456789 = 1024
> X1234567890123456789 + 2^10
      @ = 2048
```

One can use multi-variable functions.

```
> f=(x+3)^2 + (y+2)^2 - 63
      f = y^2 + 4*y + x^2 + 6*x - 50
> f(x=4,y=6)
      @ = 50
```

There are many kinds of functions. For example, the user can treat an elliptic curve as follows. $EC(1, 2, 3, 4, 6)$ defines an elliptic curve as $y^2 + 1 * xy + 3 * y = x^3 + 2 * x^2 + 4 * x + 6$.

The function $basismwg()$ returns the basis of the Mordell-Weil group of an elliptic curve f . The rank of the Mordell-Weil group of f is assigned to $@$.

The function $tors()$ returns the information of torsion group of elliptic curve f . The order of the torsion group of f is assigned to $@$.

```
> f=EC(1,2,3,4,6)
      f = EC(1, 2, 3, 4, 6)
> basismwg(f)
      basis : PT(-1, 1, 1)
      @ = 1
> tors(f,3)
      structure:      ( 0 )
      generator:      PT(0, 1, 0)
      torsion point:  PT(0, 1, 0)
      @ = 1
```

One can use vectors and matrices, too.

```
> A={{5,2,4},{4,36,7},{7,14,34}}
      A = { { 5  2  4 }
            { 4 36  7 }
            { 7 14 34 } }
> V={7,27,8}
      V = { 7  27  8 }
> A*V
      @ = { 121  1056  699 }
> det(A)
      @ = 4672
> A^(-1)
      @ = { { 563/2336  -3/1168  -65/2336 }
            { -87/4672  71/2336  -19/4672 }
            { -49/1168  -7/584   43/1168 } }
```

To get general information about `simcalc`, type in `?help`.

2.3 New Features

2.3.1 Supporting 64-bit Systems

We start supporting 64-bit Unix systems, for example Compaq Tru64 Unix on alpha. For a long time, 32-bit computer systems have been commonly used. Today there are some 64-bit computers and they are increasing their numbers. Especially, home PCs will be with 64-bit CPU of Intel or AMD only in a few years. Therefore it is very important to support 64-bit computers as soon as possible, and we start it.

To port a program from 32-bit system to 64-bit system is not an easy task. We are not the experts of Unix history, but as far as we know the industrial agreement about memory architecture of 64-bit computer systems called LP64 [30] has appeared at the first time in 1997. It was published as a part of the version 2 of the Single UNIX Specification [31] by the Open Group, a Unix standardization organization. The name LP64 means that the data type `long` (L) and pointers (P) of the programming language C have 64 bit width each. The usual 32-bit architecture called ILP32 has all of `int` (I), `long` (L) and pointer (P) as 32 bit wide data types. The biggest difference is that on LP64 architecture `int` is not compatible with other two data types. If a program was written on ILP32 architecture without expecting `long` and `int` are same length or a pointer can be substituted in `int`, it works on LP64 systems also. Otherwise, the program can not be compiled or on the luckiest case it outputs nonsense on LP64 systems. The assumption is, however, so unnatural for ILP32 systems. It means that to port a program written on 32-bit computer to 64-bit computer environment is a subtle and hard task.

Thus

we decided to use `int` as the basic data type.

The reason is that it is the simplest way to make `SIMATH` work on both 32-bit and 64-bit computers, i.e. we can assume `int` is always 32 bit wide. Using `long` as the basic type is more preferable from the point of view that it makes us to be able to use the full potential of 64-bit computer systems. It is, though, a hard way because the width differs between 32-bit and 64-bit systems.

In the future, we may face this kind of confusion again with the difference between 64-bit systems and 128-bit systems. We hope the users will be able to program without knowing such low level data types but an abstract integer type. It is obviously a desirable way

to separate the mathematical software into a lower level multiprecision library and an upper level abstract mathematical component.

There is another reason to do it, that is it makes us to be able to use third party's library, which may be faster than ours. It is true that the SIMATH uses third party's multiprecision arithmetic packages namely of Essen and Heidelberg, but it uses them as locally modified source code files. The use of the packages has not been fully documented. No one knows even whether they are still maintained or not.² This situation have made maintenance hard, and it must be fixed. We are planning that the separation will be accomplished in the next major release.

2.3.2 Cyclo-Elliptic Method

One of the features of SIMATH to be mentioned is that there are many functions for computing invariants of elliptic curves. We should, however, admit that there are not so many functions to compute important invariants of algebraic number fields such as ideal class groups, fundamental units and so on. At present, there are many known algorithms for computing those invariants as in [6], [7], [35], [34]. It will be our future important task to install them as system functions. In the first place, we decided to install system functions by the CEM (Cyclo-Elliptic Method) [25] developed by one of the authors.

Let K be a finite algebraic number field contained in an abelian extension of an imaginary quadratic field F . It includes the case where K is absolutely abelian. The CEM computes the class number h_K and the group E_K of units of K using cyclotomic/elliptic units. Let W_K be the torsion part of E_K . The CEM is based on an index class number formula of the form

$$c_K h_K = (E_K : E^K) \prod_{i \in I} (E_i : C_i), \quad E^K = W_K \prod_{i \in I} E_i \text{ (direct mod } W_K). \quad (2.1)$$

Here c_K is a simple natural number, I is a finite index set, E_i ($i \in I$) are subgroups of E_K , and C_i ($i \in I$) are groups generated by cyclotomic/elliptic units. Finding fundamental units of K from the generating cyclotomic/elliptic units, we determine these group indices. This provides us a way to compute h_K and E_K at a time. The CEM consists of the following steps:

1. We evaluate the generators of C_i ($i \in I$) in (2.1) given by means of special values of well-known functions.
2. We determine the minimal polynomials of those generators over \mathbb{Q} or F , and, on the other hand, we compute upper bounds of the indices in (2.1).
3. We determine the indices in (2.1), finding generators of E_K together in terms of their minimal polynomials over \mathbb{Q} or F .
4. As a consequence, we can compute h_K by (2.1) because c_K is always easily known.

²Heidelberg is an alias of LiDIA libI and it is alive. The status of Essen package is unknown.

The simplest K is a real quadratic field with discriminant d . Then the index class number formula is given by

$$h_K = (E_K : \langle -1, \eta \rangle), \quad \eta = \prod_{k=1}^{\lfloor d/2 \rfloor} \left(\sin \frac{\pi k}{d} \right)^{-\left(\frac{d}{k}\right)}. \quad (2.2)$$

Here $\left(\frac{d}{\cdot}\right)$ is the Kronecker symbol. In this case, we have already written a program by the CEM on `SIMATH`, which was a few days' task to translate the codes in [25].

The next targets are the cases where K is a cyclic cubic field and a cubic field with negative discriminant. There are, however, several problems to be solved in advance. For a real quadratic K as in (2.2), only the computation of the Kronecker symbol and the values of the sine function are necessary in the step 1 above. For a cyclic cubic or a more general absolutely abelian K , evaluation of the generating cyclotomic units requires some computation in the group of reduced residues modulo the conductor of K . For a cubic field with negative discriminant or a more general not absolutely abelian K , evaluation of the generating elliptic units requires much harder computation of the corresponding ideal class group of the imaginary quadratic F . Moreover, we must compute the values of special functions such as the Dedekind eta function or other elliptic modular functions. Including the case of a cyclic cubic K as in [9], we should perform some arithmetic of cyclotomic fields to install an efficient algorithm to determine the indices in the step 3. These programs are not served as standard system functions in the current version of `SIMATH`. Therefore, adding necessary programs as standard system functions, we install the CEM as a new feature. This will also help our future task of installing system functions to compute invariants of algebraic number fields described at the beginning of this subsection.

2.4 For Future Development

In this section, we point out several problems arose from the process of the development.

There is a bug, known for several years, concerning the calculation of the rank and a basis of an elliptic curve. We are trying to fix it, but we have not succeeded yet. One of the reasons is that we can get no information or document on the development of the package. It is not a good idea to say that there are open source codes. They will not help us to fix the bug. Therefore, we are now thinking

to rewrite the package by the method on the current system.

From this experience and from the process to add the CEM, we conclude to organize the `SIMATH`-group for open source development with CVS. Such a way of development requires to record the process and the document in public.

To make it clearer, we list the problems in order to add our programs in `SIMATH` or `simcalc` package. We have only written the CEM programs, and they are not incorporated to `SIMATH` or `simcalc` yet. For this purpose, we must do at least the following three works:

- Whenever users want to compute the class number and fundamental units by the CEM, they can call our functions and use it in `SM` or `simcalc`. Therefore we need to add our functions to the library.
- We need to add manuals of our functions to the online manual, the `SIMATH` Manual, and the web manual.
- We need to add there help so that users can read it on running `SM` or `simcalc`.

We get, however, useful information neither from the `SIMATH` manual nor the sources of other packages. There are no manual for developers. For open source development of the `SIMATH` system, it is strongly required

- to make a manual for developers.

These problems are common and instructive for the development of a stable system in general.

Chapter 3

NZMATH

3.1 Introduction

Since the last half of the 20th century with evolution of computers, mathematics has been influenced and remembered its experimental feature. Especially, number theory is influenced greatly since it is one of the most experimental area in mathematics, and a number of algorithms have been discovered and large tables have been constructed.

NZMATH is a newly developed system for number theoretic calculation. It is the first such system in the world written in a scripting language. Systems for number theoretic calculation are calculation systems mainly for computation related to number theory. There are several such systems; PARI formerly lead by H. Cohen and taken over by K. Belabas [33], KANT lead by M. Pohst [13] or SIMATH, which as we already described in the previous chapter, formerly lead by H. G. Zimmer and has been taken over by K. Nakamura et al. of Tokyo Metropolitan University since 2002. Some more general systems such as MAGMA [19] also provide number theoretic functionality. The author also has been participated to the SIMATH development. The present development plan of the NZMATH, a system for number theoretic calculation, is based on the experience of the SIMATH development.

The primary goal of the development is to implement various number theoretic algorithms. The latest release is available at:

<http://tnt.math.metro-u.ac.jp/nzmath/>.

The key visions of the NZMATH development are [26, 22]:

1. user / developer fusion,
2. speed of development.

The first vision means that ideally there should be no distinction between users and developers. In other systems, developers can, on the one hand, develop their software according to their interest but ought to learn programming

more deeply, collect informations about hardwares, design user interface and so on. On the other hand, users who may have frustration with provided features cannot resolve the situation by themselves. In the NZMATH vision, from the user's view point, users should be able to be developers easily. Their programs should be merged into the system without difficulties. From the developer's view point, developers should be able to concentrate on implementing mathematical concepts, especially mathematical objects and algorithms for number theory, on the system. To learn memory management or hardware variations, which is required to do if we choose C or C++, is not our desire. By using modern programming languages including Python offering automatic memory management and hiding the hardware layer from programmers, we can devote to implement the algorithms.

The second vision means that we put emphasis on the development speed of system rather than the execution speed of resulting programs. It is a paraphrasing of a commonly accepted principle "too early optimization should be avoided." Later in section 3.4, there will be more detailed discussion about the visions.

Outline of The Chapter

We shall, at first, describe rationale to develop a new system using a scripting language. It consists of a summary of the problems of SIMATH, introduction of the Python language and resolution of problems by employing a the language as the implementation language. Then, we shall summarize the vision of NZMATH in the next section. Then, we shall discuss in section 3.4 about our method of the development and how the visions are realized. Finally in section 3.5, we shall present the future works.

3.2 Rationale

3.2.1 Problems in SIMATH

We have noticed some problems in the development of SIMATH.

Since SIMATH itself is written in the C language, the memory management should be done by programmers. More precisely, each variable and argument in the definition of functions should be registered in order to protect from being garbage collected.¹ It is true that it is unnecessitated to use `malloc` and `free` directly, but such a hiding trick is also a part of SIMATH and thus in some cases it is required to maintain it.

In addition to it, the data structure managed by the mechanism is limited to lists of single precision integers. Even though the lists can represent almost every structure such as the multiprecision integer, elliptic curve, etc. and the limitation does not seem to be a true limitation, but from the point of view of a

¹In a source code, they are written as registrations of variables, then it is pre-processed to their pointers.

reader of the source code for maintenance or enhancement, the situation turns out to be an obstacle to understand it with lack of type informations.

Moreover, the reason we have enhanced **SIMATH** to run on 64 bit environment, as described in the previous chapter, is that the original **SIMATH** source code is written only for 32 bit CPUs. Though limiting executable environments eases programming, at the same time it leaves a future task for porting. Thus, it may result overload than benefit.

At last, users often use an interactive interpreter `simcalc` rather than the C language, we have seen in the previous paragraphs. In other words, the languages for system and for users are divided, and users' experience cannot easily be reflected to the system. Moreover, since implementing and maintaining such an original interpreter is a task seldomly related to mathematics, it is desirable if we can avoid to do it.

In order to resolve these problems, we shall think of utilizing a scripting language. In the next subsection, we shall overlook scripting languages, especially Python. Then, in the following subsection, we shall see how the problems are solved.

3.2.2 Python

Python is a scripting language. The word 'scripting language' has no clear definitions, but in general, it means languages implemented as interpreters and easy to use. The most popular ones are Perl a famous CGI language, Python having an origin of educational languages and Ruby founded by a Japanese and having domestic popularity in Japan.

Characteristics of Scripting Languages

A "scripting language" means a programming language usually implemented as an interpreter and developed for ease of use. Ease of use can be achieved through various strategies, thus each scripting language has its own characteristics. For example, Perl is good at filterings, which are common to the Unix tools such as `awk` or `sed`, and it provides many convenient ways to write such processes. Introductions of object oriented mechanisms are in recent trends; Ruby, for example is designed to be an object oriented scripting language from the beginning. Besides, in fact it is not limited to the scripting languages, many languages have used the implementation technic of byte-code interpreter since 1990's. The most remarkable example of the byte-code interpreter is Java, but it is not a scripting language; it has characteristics different to those of scripting languages such as declarations of variables or explicit compilation step.

We restrict ourselves to mean by the word "scripting language" the languages with the following characteristics:

- executed on interpreter via intermediate language,
- without variable declarations,

- dynamically typed, and
- with garbage collection.

Recently, there are a few scripting languages comparative to Python; Perl is the most popular one, and Ruby is becoming more popular.

Table 3.1: Comparison of scripting languages

	Perl	Python	Ruby
object orientedness	C	A	A
multiprecision integer	D	A	A
simple syntax	D	A	B
worldwide popularity	A	B	C

Python

Python is a scripting language which a Dutch programmer G. van Rossum has started developing from 1989 [37]. On its web page, the language is introduced as “an interpreted, interactive and object-oriented” language. The syntax is simple but powerful so that with its module mechanism one can write a rather large scaled program. There already exists the multiprecision integer data type, which is essential to any number theoretic calculation systems we are targeting in this paper. There also is an interactive interpreter.

The execution speed of Python is as fast as of other byte-code compiled scripting languages, i.e., it is slower than of C or Java. The slowness can be overcome with partially implementing in C language or introducing a just-in-time compiler.

Notably, there are a lot of third party libraries including mathematical libraries such as NumPy² among several numerical modules, SciPy³, etc. SAGE (Software for Algebra and Geometry Experimentation)⁴ by W. Stein targets similar areas with our project, but uses a different approach.

3.2.3 Resolutions of Problems

We shall discuss how the four problems in section 3.2.1 can be solved by adapting Python.

At first, about memory management, since garbage collection is provided, users do not have to think about memory management at all. Developers are in the same situation with users while they write code in Python. It means they are also free from memory management. There is no need to maintain code for memory management as in SIMATH.

²NumPy: <http://numpy.scipy.org/>

³SciPy: <http://www.scipy.org/>

⁴SAGE: <http://modular.fas.harvard.edu/SAGE/>

About the next problem of types, since Python has object-oriented mechanism, users can define arbitrary types.

Portability means the portability of Python, thus it is not our concern. The only question is whether Python will be ported to various environment, but at least the recent trend shows that support for major operating systems such as Windows, Macintosh and various Unices would continue.

An interactive interpreter is provided, and it is sufficient for daily use cases. That is, we have a tool to construct simcalc-like interactive environment. If we, however, hope Mathematica-like GUI environment, there are several concerns. For example, how to render mathematical equations, or which toolkit is better for supporting wider range of environments.

3.2.4 Other Merits and Demerits

As we have seen, by adapting Python to our project, the problems above will be solved. We shall discuss here for other merits and demerits.

At first, about the speed that a lot of people may be interested in, it is true that the execution of written program alone in Python is slower than that of in C or Java.

We would like to point two things. One is that the speed does not mean the execution speed alone. By using Python, both implementation speed of the system itself and programming speed of users own can be faster. In addition to it, by using the same language for implementing the system and users' programming, it gets easier to take in users' program to the system, and it is beneficial for implementation speed again. Therefore, little CPU time will be wasted as a total.

Another is that there are ways to increase execution speed, indeed. The first method is implementing in C. Since there is an API of Python by which a C library can be called, it is possible to call a C-implemented speed critical part from Python. Of course, the method requires more; detailed knowledges about the internal memory management of Python, programming skill of C, etc. The second method is using a just-in-time compiler. It is not a part of Python standard library, but there is a just-in-time compiler called Psyco⁵ by A. Rigo. At the time of writing, regretfully it is usable only on x86⁶, but porting project to other CPUs is progressing. Psyco utilizes the characteristics of the dynamically typed language that the type of a variable can only be known on time of execution, and it generates an execution machine code for the specialized type. It is said that its execution speed is two to one hundred times as fast as plain Python execution.⁷

⁵Psyco: <http://psyco.sourceforge.net/>

⁶It refers CPUs derived from Intel 8086, and there are Intel Pentium series, AMD Athlon series, etc.

⁷Four times faster is in average.

3.3 History and Current Status

3.3.1 History

NZMATH has been developed for about three years. We recall the history of the development of NZMATH, briefly.

We started the development of NZMATH in 2003, after the contract of maintenance of SIMATH between Simens and Nakamura's group had been finished. Before its first release we announced the plan in AC2003 conference [20]. The first version 0.0.0 was released on the November.

There are several releases in 2004: 0.1.0 on March, 0.1.1 on May and finally 0.3.1 on December⁸. The change of minor version number means the change in the module structure, while the micro version number means bug fixes and/or enhancement. The major version number has not been incremented yet, but it is reserved for incompatible change of based Python versions, and it will firstly be incremented when we will drop compatibility with 2.3 version. The modules of factorizations, elliptic curves were written in the period [17, 18].

KNOPPIX/Math 2005 became the first Linux distribution officially included NZMATH on February 2005⁹. There were two more releases 0.4.0 and 0.4.1 before the AC2005 conference. From the version 0.4.1, we have started to distribute the test code also.

In 2006, the versions 0.5.0 and 0.5.1 has been released. From 0.5.0, NZMATH adds modules for computing class numbers of imaginary quadratic fields, and some for group related modules.

3.3.2 Current Status

NZMATH is provided as a library package of Python. In the Python terminology, a file consists of classes or functions definitions is called "module", and a directory containing modules is called "package".

Currently, NZMATH has several basic data types and several modules for number theoretic computations. It runs with Python 2.3 or higher¹⁰. Here are the entire module descriptions of the latest release at the time of writing, version 0.5.1. Each heading of the paragraphs in the next few subsections are the names of the modules in NZMATH.

3.3.3 Modules for Basic Data Types

There is a multiprecision integer data type in Python by default, but there must be more data types for number theory.

rational provides rational numbers and rational integers. Although a multiprecision integer type is provided by Python, its result of division is either an

⁸The releases 0.2.0, 0.2.1 and 0.3.0 have bugs and are quickly replaced by 0.3.1.

⁹<http://www.knoppix-math.org/>

¹⁰The latest release at the time of writing is 2.5.

integer, i.e. Euclidean quotient, or a floating point number. Our rational integer returns a rational number as a result of division. The module also provides the field of rational numbers \mathbb{Q} and the ring of rational integers \mathbb{Z} .

integerResidueClass ¹¹ provides integer residue classes. The class for rings of integer residue classes or $\mathbb{Z}/n\mathbb{Z}$ and the class for their elements are defined.

finitefield provides finite fields and their elements. There are six classes, three for fields and other three for their elements. The correspondence reflects the class hierarchy explained with the `ring` module below. The finite fields with prime cardinality are provided as `FinitePrimeField` and their elements are `FinitePrimeFieldElement`. The extended fields and their elements are provided as `FiniteExtendedField` and `FiniteExtendedFieldElement`, respectively. The remaining two classes are the abstract base classes for the classes above.

polynomial provides univariate or multivariate polynomials. The basic polynomial class can be used with any commutative rings as the coefficient ring. There are specialized ones for special coefficient rings; such as integers or rational numbers. Polynomial rings are also provided.

matrix, vector provides matrices, vectors and related functions such as determinant, trace, LU decomposition, etc.

rationalFunction ¹¹ provides a class `RationalFunction` for rational functions. It is only for the result of a division of polynomials.

3.3.4 Number Theoretic Modules

The main part of the system for number theory. There are only a few modules now, but the development directed to enrich this category in the future.

multiplicative provides multiplicative number theoretic functions such as the Euler totient function φ , the Möbius function μ , and the sum of k^{th} powers of all divisors σ_k .

gcd provides some functions related to compute the greatest common divisors of integers.

prime provides primality testing functions including the trial division method, some pseudo prime tests, the Jacobi sum method [6, section 9.1], etc., and some functions related to prime generation.

¹¹ The module names `integerResidueClass` and `rationalFunction` are slightly violating the naming convention and can be renamed in the future.

factor provides integer factorization functions¹². There are methods from the trial division to the multiple polynomial quadratic sieve method [18].

elliptic provides elliptic curves and related functions. Elliptic curves are defined over a finite prime field with any characteristic but 2 or 3, and their order and structure of the Mordell-Weil groups can be computed [17, 1]. The structure computation is carried out via the Weil pairing.

quad provides class number computation of imaginary quadratic fields. The method used in the module is to count reduced quadratic forms [17, 1]. Other methods will be implemented.

3.3.5 Miscellaneous Modules

The rests are auxiliary modules.

arith1 provides variety of functions: **floorsqrt** to compute the integer part of the square root of an integer, **log** to compute the integer part of the logarithm of an integer with specified integer base, etc.

bigrandom provides random number generator for big numbers. It is based on the Python's default random number generator. Since the Python implementation has bug with integers of size more than 32 bit, the module should wrap it.

combinatorial provides combinatorial functions including factorial, binomial coefficients, the Bernoulli number and the Catalan number.

ring provides abstract declaration of rings. There are two hierarchical trees of classes: one for rings and the other for their elements.

For example, **Ring** is the root class of ring classes. It declares that all descendants must have attributes **zero** and **one**, in other words, we call with 'ring' a ring with unity. Then, **CommutativeRing** inherits **Ring**. and so on.

The counter part of the **Ring** class in the other tree is **RingElement**. The definition of an element of a ring is too abstract to write down concrete code in the class, there is almost nothing in the body of the class definition. However, there is a commonly used method name to know a ring to which the element belongs: the **getRing** method.

There are also classes for fields, since field is a kind of commutative rings. We do not much care about skew fields and do not provide them.

lattice provides the LLL algorithm of lattice [6, section 2.6].

¹²Actually, **factor** is not a plain module like others, but a sub-package consists of a few modules.

zassenhaus provides factorization of integer coefficient polynomials. The algorithms to do it are the Berlekamp-Zassenhaus method and van Hoeij's algorithm [10].

equation provides functions to solve algebraic equations $f(x) = 0$ in the complex field or finite fields for lower degree equations.

group provides finite abelian groups and methods to compute their group orders.

permute provides permutation groups.

real provides functions for real numbers. The module corresponds to the **math** standard module of Python. There are no definitions for real numbers or arbitrary precision floating point number. The results are in rational numbers and very slow.

imaginary provides functions for complex numbers, but since the name **complex** is already used by Python for the type name of complex numbers, it is named **imaginary**. Similar to the **real** module, the **imaginary** module corresponds to the **cmath** standard module of Python.

3.3.6 Manual

The manual is provided as HTML files included in the distribution. It is also available on the web¹³. With the Python interpreter, **help** function can extract documentation strings embedded in the source files. This functionality is a part of Python, and the extracted document is also viewable from shell command **pydoc**. However, documentation strings are not always written for all functions. Some programmers sometimes forget to write them, and some feel hard to write them in English. The **help** on the interpreter is, thus, not always helpful.

Instead of the documentation strings in the source code, HTML manual is maintained with a wiki¹⁴ system to keep up to date. Someone who feels less hard to write English can help to write the wiki manual pages, if the programmer feels it is hard. Then, on the time of releases, the pages are converted to plain HTML files.

In addition to the process, the wiki approach has an advantage that translations of the manual into other languages than English are possible. It is recommended by many people that documentation should be in the code, but it is reasonable only for a domestic project. Having manuals in more than one languages in the code seems ridiculous. NZMATH must be an international project, and there are needs for translated manuals. Though English is the lingua franca

¹³NZMATH manual: <http://tnt.math.metro-u.ac.jp/nzmth/manual/>

¹⁴NZMATH Wiki: <http://nzmth.sourceforge.net/wiki/>

in the academic world for the last half of a century, neglecting other languages is not justified.

3.3.7 Applications

There is a research using NZMATH; related to a quantum public key cryptosystem (QPKC), some computations have been carried out with NZMATH by Nishimoto and Nakamura [27]. The computations are about norms of quadratic or cubic field elements. Since there is no module for such elements, they write their own modules with the help of some NZMATH functions.

We are now implementing algebraic numbers on NZMATH, as discussion will be in section 3.5.1. Integration with their implementation is hoped.

3.4 Development

We have been developing NZMATH for over two years. The visions explained in section 3.1 remain unchanged through the development. We now have two more development concepts: “open source” and “agile development”.

3.4.1 User / Developer Fusion

One of the NZMATH’s visions is to get rid of the distinction between users and developers.

From the developer side, since we have chosen Python as the implementation language, developers do not have to think about memory management nor portability of code; such features are the subjects for the development of the language itself. In other words, developers can concentrate on mathematical problems. It is a direct benefit of choosing Python toward user / developer fusion.

From the user side, the fact that both users and developers use the common language makes contributions easy. The already existing systems have different languages for users: GP for PARI, KASH for KANT, simcalc for SIMATH. The language gap hardens for users to feed back their experience to the system development. In addition to it, since the Python language is widely used and general purpose, the knowledge is useful even apart from NZMATH.

It is true that user base is still too small to say anything. When the number of users increases, our claim shall be proven.

3.4.2 Speed of Development

Another one of the NZMATH’s visions is to put emphasis on speed of development. The choice of Python as the implementation language helps to achieve the vision. Since the NZMATH project have started later, catching up faster to already existing systems is necessary to be accepted by a broader audience.

Python has been chosen partly because of reducing educational cost of students. The way of developing the software is influenced by resources. Our dominant resource is a human resource. There is no full-time staff; main developers are master course or undergraduate students. They leave from the development in a year or two on time of their graduations. It is, thus, desirable to choose a language easy to learn.

Almost all other systems are implemented in C or C++, because the resulting programs run fast, but it is “too early optimization.” We would like to devote to implement algorithms for number theory. Writing in C like language, however, requires other knowledge such as memory management or hardware variations. Modern programming languages including Python offer automatic memory management and hide the hardware layer from programmers. Another feature of Python is that it is dynamically typed language. Statically typed languages can detect more inconsistency of types of objects at the compilation time. There are two strategies of static typing: variable declaration and type inference. C uses former strategy, in other words, programmers are forced to declare types for every variable. The latter strategy is mainly used for functional languages and we do not argue about it here. Spreading dynamically typed languages indicates that costs of such declarations are not worth to pay. Both features of Python enable the developers to concentrate on mathematics, as desired. If it is realized later that a part of the system needs more speed, then it is possible to reimplement the part by C with the experience of implementation in Python.

We achieved writing the modules explained in section 3.3 in less than 3 years. It shows that our choice of language was right.

3.4.3 Open Source

We distribute NZMATH under the BSD license. The license basically permits users to do anything. The program distributed under the license can be freely available, freely redistributed, and freely used. Since we have chosen a scripting language as the implementation language, users can always read the source code to be interpreted. It is, therefore, natural to make our source programs available freely. It means our system can be distributed as an open source software. Moreover, we believe that there is no need to restrict usages of the system; even including it to a closed source software. For such purpose, the GNU general public license, which is used by PARI for example, is too restrictive, because the license does not allow derived works without source code accessibility. Therefore, our choice of license is the BSD license.

The project uses a mailing list, a CVS repository, web pages and a wiki. The `nzmath-user` mailing list¹⁵ is for user to user communications. The CVS repository is for version control of the source code. The web pages are for releases of NZMATH to the public, and for the user manual. And the wiki¹⁴ is mainly for communications among developers, including maintenance of the user manual.

¹⁵`nzmath-user` mailing list: `nzmath-user@tnt.math.metro-u.ac.jp`

All services were served on the machines at TMU. They, however, need not be served by ourselves. The Internet has grown rapidly during the last decade. Universities are not the only places to host such servers. Nowadays, there are several sites hosting free / open source softwares with version control softwares like CVS, mailing lists and web pages. We have come to use SourceForge.net for serve them since August 2006¹⁶.

There are several merits and a few demerits. The first merit is that the project will be recognized widely as an open source project. Secondly, participation to the development will be easier. People may hesitate to edit wiki pages if the host is in a university domain. The last point has only a small importance, but it will reduce our daily routines as a side effect. One of the demerits is that we loose control of server softwares, but it is negligible compared to the merits.

3.4.4 Agile Development

Because the author is personally influenced by the ideas of agile paradigm [4] or, more precisely, extreme programming methodology [3], we shall discuss about the development here from such a point of view. It is clear that our way of development is not of “water fall” model, but rather incremental one. Therefore, there should be some hints for improvements from agile models.

The first point is about tests. In the context of a professional software development, the code coverage may have importance, but we do not require such a level of test for all developers. It is true that writing merely a test to see that the usual cases run correctly is effective to prevent most of ridiculous bugs. I do not know whether the other projects have explicit test codes or not, but it is worth writing.

The second point is about code reviews. By reviewing the code the other member have written, the members can learn about the module reviewed, acquire the sense of goodness of code, or detect minute bugs. There might be possibility to practice pair programming instead, since pair programming is a sort of continual reviewing. However, the development group has master course students as its main members, and they need their own result for their degrees, the activity that makes personal contributions ambiguous is not acceptable.

The third point is to release small. Our release schedules have not been so punctual, but the releases were, at least, the results of shortly scheduled plans. The plans include a module to be added or to be modified. If there is no such module because the development of it will take more than the usual release period, we plan a bug fix release. Such continual releases may stimulate the potential users, result some feed backs, and keep the motivations of the members.

The last point is about incremental design. Design should evolve day by day, not because demands for number theory system change, but because developers’ understandings of the area progress. Regretfully, on this point, the activities such as discussing the design or writing the design explicitly on a paper be-

¹⁶NZMATH Project page: <http://sourceforge.net/projects/nzmath/>

fore coding are rarely done. Moreover, refactorings after implementations are occasional. It is necessary to improve these practices.

3.5 Future of NZMATH

The development of NZMATH will continue further. We shall discuss about some short term perspectives in the next subsections, then about longer term perspectives.

3.5.1 Algebraic Number Field

The algebraic number field is the next big subject for NZMATH development. The only thing NZMATH has had toward the area is the `quad` module, which computes the class numbers of imaginary quadratic fields.

Since algebraic number fields offer a fundamental tool to study number theory, NZMATH should handle them to be more “number theory oriented” system.

We have started writing programs for it after the release of 0.5.0 in February. The very first step is to define classes to represent algebraic numbers with arithmetic operations. The methods to compute norm or trace are also provided. The next step will be to define algebraic number fields and their integer rings. Then, invariants of the fields, such as discriminants, class numbers, unit groups, etc. will follow. Handling ideals and computing prime decompositions, will also be provided.

There will be specialized classes or modules for small degree fields similar to the `quad` module: real quadratic and/or cyclic cubic fields. Other kinds of fields may also have specialized modules, depending on the easiness of implementation.

3.5.2 Web UI

There has been no special interface for NZMATH other than the default Python interpreter. Though we think the interpreter remains as the primary interface, a possibility of another interface has been sought occasionally.

We are planning to make a web user interface. By “web” we mean that users will use their web browsers to start calculations. The most simple implementation looks like the Online MATH Calculator by W. Stein¹⁷. There can be, on the other hand, several choices how to construct the server side: a good old CGI, a plug-in module for web server or a dedicated web application server. Fortunately, there are plenty of choices with Python web servers.

The heavy calculations are allowed to be carried out only on users’ desktop with or without the web interface. If users will be able to run the server on their own desktop easily, the web user interface will still be useful. Such installation of server / client pair on user’s desktop is called “desktop server”¹⁸.

¹⁷Online MATH Calculator: <http://modular.math.washington.edu/calc/>

¹⁸The author does not know who invented the word, but have learned the concept from PyDS blog system.

The merits of constructing a user interface with the web technology comparing to with an independent client are: (1) the required knowledge for developer is smaller and (2) it is platform independent. It can be the first step towards using NZMATH for potential users, if the pages contain more introductory materials or integrated with manual pages.

The weakness of the web interface is lack of programmability. The Python interpreter, therefore, remains as the interface for programming, and we expect that almost all users will program anyway.

We have started to write a prototype called “ikura” upon Django framework¹⁹.

3.5.3 Speed of Execution

It has been and will be always the problem how to respond to the request of speed. We have chosen to put higher priority on easiness of programming and speed of development rather than on speed of execution. The formerly planned schedule about speed of execution is that at some point when the pace of additions of new modules will slow down, running time improvement will be a main topic of the development.

The strategy is endangered if execution is too slow to make the total time of programming and execution is slower with NZMATH than with other systems. We should, then, make some optimizations for speed of execution. There are requests of speeding up polynomial computation, for example. It is in fact the bottleneck of a certain kind of computations. Our polynomial implementation is designed for general purpose, i.e. choosing the variable and specifying coefficient ring are always required even if one only deals with polynomials in a specific ring; $\mathbb{Z}[X]$ for example. They are obviously eliminatable overheads if there is a specialized polynomial type. Use cases show that there are a few very commonly used types of polynomials. It is, thus, possible to make such special types to resolve the bottleneck of polynomial computations.

3.5.4 Long Term Plans

Finally, we shall discuss about long term future plans.

Of course, the implementation and improvement of wider range of algorithms will continue. Especially, construction of modules for invariant computations of algebraic number fields and elliptic curves continues to be in the main stream of the development. Elliptic curves over the rational field and hopefully algebraic number fields should be a target. p -adic numbers or group rings are considered useful. We hope to provide also some tools for analytic number theory.

Speeding up execution may be one of the main topics, when the development will be saturated with variety of modules or when we find a bottleneck of the speed of computations. The former seems very far from now, but the latter may start earlier.

¹⁹Django: <http://www.djangoproject.com/>

Connecting to other systems is another future issue. Implementing some protocol stack will be in consideration. OpenXM [32] used in Risa/Asir is one of the candidates.

3.6 Conclusions

We have been developing the calculation system for number theory called NZMATH for about three years. The system is implemented in Python, a scripting language, in order to achieve the visions 1) user / developer fusion and 2) speed of development. The development has successfully been providing the wide variety of modules and the manuals of them, in the short period. It shows the correctness of the concepts.

There will be more topics to be covered by the system, including algebraic number fields. Web user interface is considered as a probable option to be implemented. Speed of execution is not ignorable factor of the development. Such problems arose in the development are discussed.

We hope the system be accepted by wider range of people, from number theorists to students of other areas.

Chapter 4

Related Problems

4.1 Introduction

In this chapter, two related problems shall be discussed. The first one is about quantum computation, and the second one is about computable numbers.

Theory of quantum computation is one of the most important break-through of complexity theory. Since Shor have shown discrete logarithm problem and factorization are polynomial time on quantum computer [38], current cryptography systems relying upon difficulty of factorization or discrete logarithm problem are insecure when quantum computer will be realized. However, realization of quantum computer is not easy. There are a number of approaches but most of them can achieve to construct only a few quantum bits (qubits). Amongst of them, Nuclear Magnetic Resonance (NMR) quantum computer is the most successful approach. For example, Vandersypen and his colleagues of IBM and Stanford university compose organic molecules realizing 7 quantum bits, and factor 15 with them using Shor's algorithm [42]. A Subtlety with The NMR quantum computer is that it is not a single quantum computer but a large number of quantum computers computing in parallel and a result obtained is actually an average of results of them. Therefore, it should be considered carefully that whether an NMR quantum computer is truly a quantum computer or not. The first section of this chapter shall discuss models of NMR quantum computer: one proposed by Atumi and Nishino [2] and ours. It is based on [21].

The second problem is from developing NZMATH: what kind of real or complex numbers can be provided? The set of real numbers has uncountable cardinality, and of course the set of complex numbers does. It is well known, though, that even the whole set of definable numbers only has countable cardinality and the whole set of complex numbers can never be subjects of computation. From the beginning of the theory of computation, Turing defined computable numbers [41]. Rice showed that the set of the whole complex computable numbers forms an algebraically closed field [36]. Computable numbers, though, contain many numbers hard to compute. Polynomial time computability is the com-

monly accepted characteristics of feasibly computable functions. Therefore, it is natural to think of polynomial time computable numbers. The second section of this chapter shall discuss about them. It is based on [24].

4.2 Bulk Quantum Turing Machine

4.2.1 Introduction

Recently, among experiments for realization of quantum computers, Nuclear Magnetic Resonance (NMR) quantum computers have achieved the most impressive succession. For example, Vandersypen and his colleagues of IBM and Stanford university compose organic molecules realizing 7 quantum bits, and factor 15 with them using Shor's algorithm [42].

The NMR quantum computation differs from the ordinary quantum computation; since it manipulates many molecules simultaneously, one cannot utilize the projection property of measurement but can only obtain the ensemble average. Note that, however, it is merely unusable the projection in order to select a specific state, but in fact each molecules is projected to the state when it is measured so that the whole ensemble is the ensemble which gives the measurement value of the ensemble average. It is a basic assumption of the quantum mechanics.

Today, there is a model of the NMR quantum computation by Atsumi and Nishino [2]: bulk quantum Turing machine (BQTM). It has, however, two assumptions contradicting with quantum mechanics: (1) a measurement does not cause projection on each molecules, and (2) the probability to obtain the ensemble average value in a certain interval is 1. We already mentioned about the first point above, and the second point will be shown false later.

We shall define a new model of the NMR quantum computer removing the difficulties of Atsumi and Nishino's model. Then we shall show the computational power is the same with both Atsumi and Nishino's BQTM and the original quantum Turing machine (QTM) for decision problems.

4.2.2 Definitions

At first, we recall the definitions of the QTM and BQTM. The first part of the definition is common to both.

Definition 4 (Tape, Symbols and Unitarity). A quantum Turing machine (QTM) (or bulk quantum Turing machine (BQTM)) \mathcal{M} is defined by a triplet (Σ, Q, δ) where: Σ is a finite alphabet with an identified blank symbol $\#$, Q is a finite set of states with an identified initial state q_0 and final state q_f which is not equal to q_0 , and δ is a function called quantum transition function:

$$\delta : Q \times \Sigma \rightarrow \mathbb{C}_C^{\Sigma \times Q \times D} \quad (4.1)$$

where D denotes directions $\{L, R\}$ or $\{L, N, R\}$, and \mathbb{C}_C denotes the set of

computable numbers. The QTM (or BQTM) has a two-way infinite tape of cells indexed by \mathbb{Z} and a single tape head that moves along the tape.

The quantum transition function δ induces a time evolution operator $U_{\mathcal{M}}$ of the inner-product space of finite complex linear combination of configurations of \mathcal{M} . The time evolution operator $U_{\mathcal{M}}$ must be unitary.

The observation of QTM is carried out as usual physics.

Definition 5 (Observation of QTM). When a QTM \mathcal{M} in a superposition $\Psi = \sum_i \alpha_i c_i$ is observed, each configuration c_i is obtained with probability $|\alpha_i|^2$ and the superposition of \mathcal{M} is updated to $\Psi' = c_i$. A partial observation is also allowed.

Atsumi and Nishino replaced the observation of QTM above with the measurement of BQTM below in [2].

Definition 6 (Measurement of BQTM). A measurement is an observation such that $|\alpha|^2 - |\beta|^2$ is obtained for a qubit $\alpha|1\rangle + \beta|0\rangle$ with error range less than θ and with probability 1. The measurement does not disturb the superposition which \mathcal{M} is in, and can be repeated several times. No partial observation is allowed.

We make a modification to the measurement.

Definition 7 ((ϵ, θ) -measurement). An (ϵ, θ) -measurement is an observation such that $|\alpha|^2 - |\beta|^2$ is obtained for a qubit $\alpha|1\rangle + \beta|0\rangle$ with error range less than θ and error probability less than ϵ . If either $|\alpha|^2$ or $|\beta|^2$ is zero, the error probability is exceptionally zero. The measurement does disturb the superposition which \mathcal{M} is in, and can not be repeated multiple times. No partial observation is allowed.

We call a quantum computer defined by definition 4 and 7 a modified bulk quantum Turing machine (BQTM*).

Since a set of parallel statistically independent QTMs gives an model of BQTM*; the bigger the number of QTMs gets, the smaller the parameters θ and ϵ of BQTM* get. A quantitative relationship among parameters is given in the following subsection, but beforehand we give a qualitative statement.

Lemma 1. *If a QTM without partial measurements and resulting 0 or 1 exists, corresponding BQTM* exists for $\frac{1}{2} > \forall \theta, \epsilon > 0$.*

Proof. Consider n parallel independent QTMs without partial measurements and resulting 0 or 1. Since partial measurements are not used, the computation is also carried out by BQTM* having the same quantum transition function. Therefore, the only difference is the final observation or measurement. If the final superposition of the cell is $\alpha|1\rangle + \beta|0\rangle$ and one assigns -1 to $|0\rangle$, the ensemble average is $|\alpha|^2 - |\beta|^2$. The law of large numbers assures, the bigger n gets, the smaller the error probability ϵ becomes with given value range of θ . \square

Table 4.1: value of n for θ, ϵ

value of n		ϵ		
		0.04550	0.02000	0.01000
θ	2^{-5}	1024	1699	2018
	2^{-6}	4096	6795	8069
	2^{-7}	16384	27177	32275

Note that the measurement of BQTM is understood as the $(0, \theta)$ -measurement, if we forget about the disturbance on the superposition. However, the parameters of positive θ with $\epsilon = 0$ is not realized by the parallel independent QTMs until the superposition is in one of the eigenstates ($|0\rangle$ or $|1\rangle$).

Relationship among ϵ, θ and the Number of QTMs

As stated above, BQTM* is realizable by parallel independent QTMs and the parameters θ and ϵ of BQTM* and the number n of QTMs are dependent. The relationship among them are known by de Moivre-Laplace's theorem. It holds asymptotically:

$$\frac{1}{\sqrt{2\pi}} \int_{-t}^t e^{-\frac{x^2}{2}} dx \sim 1 - \epsilon \quad (4.2)$$

where $t = 2\theta\sqrt{n}$. See, for example, [16].

By this formula, the table 4.1 is obtained. The number n is considered as the number of molecules in NMR to be observed.

4.2.3 Complexity Classes

There are some complexity classes for QTM and corresponding classes for BQTM. We shall define the corresponding classes for BQTM*, then to show their equivalences. Since we concern classes of decision problems, we assume that the alphabet Σ includes $\{0, 1\}$. Moreover, a tape cell called acceptance cell be in the superposition $\alpha|1\rangle + \beta|0\rangle$ when it is observed, i.e. there is no possibility to have blank or any other symbols.

We shall discuss about three kinds of classes. The classes for QTM was defined by Bernstein and Vazirani [5], and for BQTM by Nishino et al. [28][29].

Exact Quantum Polynomial Time

First of all, we see the classes of exact quantum polynomial time languages.

Definition 8 (EQP, EBQP, EBQP*). The quantum complexity classes of exact quantum polynomial time languages are defined according to the models of the quantum computers:

1. A language L is in the class **EQP** if and only if there exists a QTM and polynomial p such that for any input x an observation of a certain tape cell after calculation of $p(|x|)$ steps gives 1 with probability 1 if x belongs to L , 0 with probability 1 otherwise.
2. A language L is in the class **EBQP** if and only if there exists a BQTM and polynomial p such that for any input x a measurement of a certain tape cell after calculation of $p(|x|)$ steps gives more than $1 - \theta$ with probability 1 if x belongs to L , less than $-1 + \theta$ with probability 1 otherwise.
3. A language L is in the class **EBQP*** if and only if there exists a BQTM* and polynomial p such that for any input x an (ϵ, θ) -measurement of a certain tape cell after calculation of $p(|x|)$ steps gives more than $1 - \theta$ with probability 1 if x belongs to L , less than $-1 + \theta$ with probability 1 otherwise.

Theorem 2. **EQP** and **EBQP*** are equivalent.

Proof. It is possible to observe a value from a tape cell of QTM with probability 1 only when the cell is equal to one of the eigenstates; $|0\rangle$ or $|1\rangle$. Thus, if a language L is in the class **EQP**, there exists a QTM \mathcal{M} with a certain cell in the eigenstate to be observed at the last step. Suppose BQTM* \mathcal{M}^* which has the same quantum transition function δ with \mathcal{M} . The calculation steps are identical and the tape cell to be read is in the eigenstate. Then, by the definition 7, it is possible to obtain the value with probability 1. Thus, the language L is in the class **EBQP***.

Conversely, if a language L is in the class **EBQP***, there exists a BQTM* \mathcal{M}^* . Since the probability to observe either one of the values 1 or -1 is 1, the tape cell is in either one of the eigenstate when it is observed. Suppose QTM \mathcal{M} which has the same quantum transition function δ^* with \mathcal{M}^* . Then, the calculation steps are identical and the final result is obtained with probability 1, i.e. the language L is in the class **EQP**. \square

Corollary 1. **EBQP** and **EBQP*** are equivalent.

Proof. It is a direct consequence of **EQP** = **EBQP** [28]. \square

Bounded Error Quantum Polynomial Time

Next, we see the classes of bounded error quantum polynomial time languages. These are the most important classes.

Definition 9 (BQP, BBQP, BBQP*). The quantum complexity classes of bounded error quantum polynomial time languages are defined according to the models of the quantum computers:

1. A language L is in the class **BQP** if and only if there exists a QTM such that for any input x an observation of a certain tape cell after calculation of polynomial time of its size gives 1 with probability more than $2/3$ if x belongs to L , 0 with probability more than $2/3$ otherwise.

2. A language L is in the class **BBQP** if and only if there exists a BQTM such that for any input x a measurement of a certain tape cell after calculation of polynomial time of its size gives more than $1/3 - \theta$ if x belongs to L , less than $-1/3 + \theta$ otherwise.
3. A language L is in the class **BBQP*** if and only if there exists a BQTM* such that for any input x an (ϵ, θ) -measurement of a certain tape cell after calculation of polynomial time of its size gives more than $1/3$ if x belongs to L , less than $-1/3$ otherwise.

Theorem 3. **BQP** and **BBQP*** are equivalent.

Proof. Assume that L is in the class **BQP**. By the definition there is a QTM \mathcal{M} which accepts L . We can assume that the tape cell to be observed is in superposition $\alpha|1\rangle + \beta|0\rangle$ and $p = |\alpha|^2 > 2/3$ if an input belongs to L . By the lemma of section 4.2.2, there is a BQTM* \mathcal{M}^* corresponding to \mathcal{M} with $\theta < p - 2/3$. Then, the (ϵ, θ) -measurement of \mathcal{M}^* gives a value in range $(|\alpha|^2 - |\beta|^2 - \theta, |\alpha|^2 - |\beta|^2 + \theta)$.

$$|\alpha|^2 - |\beta|^2 - \theta = p - (1 - p) - \theta \quad (4.3)$$

$$> 2p - 1 - p + 2/3 \quad (4.4)$$

$$> 1/3. \quad (4.5)$$

The other case is shown similarly. Thus, we can conclude L is in the class **BBQP***.

Conversely, Assume that L belongs to **BBQP***. By the definition there is a BQTM* \mathcal{M}^* which accepts L . If an input belongs to L , the (ϵ, θ) -measurement of the acceptance cell gives a value more than $1/3$. With an identity $|\alpha|^2 + |\beta|^2 = 1$, it gives $|\alpha|^2 > 2/3$. In the case an input does not belongs to L , it is shown in the same way that $|\beta|^2 > 2/3$. Thus, considering a QTM \mathcal{M} which has the same quantum transition function with \mathcal{M}^* leads to the conclusion that L is in the class **BQP** \square

Corollary 2. **BBQP** and **BBQP*** are equivalent.

Proof. It is a direct consequence of **BQP = BBQP** [28]. \square

Zero Error Quantum Polynomial Time

Finally, we see the classes of zero error quantum polynomial time languages.

Definition 10 (ZQP, ZBQP and ZBQP*). The quantum complexity classes of zero error quantum polynomial time languages are defined according to the models of the quantum computers:

1. A language L is in the class **ZQP** if and only if there exists a QTM such that for any input x an observation of a certain tape cell (halt cell) after calculation of polynomial time of its size gives 1 with probability more than $1/2$ and then an observation of another cell (decision cell) gives 1 with probability 1 if x belongs to L , 0 with probability 1 otherwise.

2. A language L is in the class **ZBQP** if and only if there exists a BQTM such that for any input x an observation of a certain tape cell (halt cell) after calculation of polynomial time of its size gives more than 0 with probability 1 and then either one of the following cases holds:
 - measurement of another cell (accept cell) gives 1 with probability 1 if x belongs to L ,
 - measurement of one another cell (reject cell) gives -1 with probability 1 if x does not belong to L .
3. A language L is in the class **ZBQP*** if and only if there exists a BQTM* such that for any input x an observation of a certain tape cell (halt cell) after calculation of polynomial time of its size gives more than 0 with probability more than $1 - \epsilon$ and then either one of the following cases holds:
 - (ϵ, θ) -measurement of another cell (accept cell) gives 1 with probability 1 if x belongs to L ,
 - (ϵ, θ) -measurement of one another cell (reject cell) gives -1 with probability 1 if x does not belong to L .

Theorem 4. **ZQP** and **ZBQP*** are equivalent.

Proof. If a language L is in the class **ZQP**, there exists a QTM \mathcal{M} and a polynomial p , which is a time estimation polynomial. We construct a BQTM* \mathcal{M}^* from \mathcal{M} in the following way.

At first, we replace the initialization steps of the decision cell of \mathcal{M} with the steps to initialize the accept cell to $|1\rangle$ and the reject cell to $|0\rangle$. If there is no initialization steps in \mathcal{M} , we insert the steps above.

The steps of \mathcal{M}^* after initialization are identical to \mathcal{M} until it reaches to the step to write the result in decision cell.

Finally, the writing step is replaced with those which write the same result in both the accept cell and the reject cell: if x belongs to L then the result is $|1\rangle$, otherwise $|0\rangle$.

The changes increase at most a constant k steps. Thus, after $p(|x|) + k$ steps (ϵ, θ) -measurement of halt cell can give, if one choose an appropriate θ , more than 0 with probability more than $1 - \epsilon$, since the corresponding observation of \mathcal{M} gives 1 with probability more than $1/2$. At the moment, if x belongs to L , the accept cell has not been changed since the initialization and reading $|1\rangle$ gives 1 with probability 1. On the other hand, reject cell is not in the eigenstate and it is impossible to obtain -1 with probability 1 by (ϵ, θ) -measurement. In the case when x does not belong to L , the behaviors of the accept and reject cell are switched and an (ϵ, θ) -measurement of reject cell gives -1 with probability 1. Thus, L is in **ZBQP***.

Conversely, we assume L is in the class **ZBQP***. Then, there exists a BQTM* \mathcal{M}^* to accept L . Consider a QTM \mathcal{M} which has the same quantum transition function with \mathcal{M}^* , and identify the accept cell or the reject cell

as the decision cell. Obviously, if an observation of the halt cell gives 1, the result is obtained correctly with probability 1. Moreover, the probability that the observation of halt cell gives 1 is more than $1/2$, because \mathcal{M}^* gives more than 0 with probability more than $1 - \epsilon$. Therefore, L is in the class **ZQP**. \square

Corollary 3. **ZBQP** and **ZBQP*** are equivalent.

Proof. It is a direct consequence of **ZQP** = **ZBQP** [29]. \square

4.2.4 Conclusion

We construct a model of NMR quantum computation named modified bulk quantum Turing machine **BQTM***. It can be realized as a set of statistically independent QTMs and more consistent with quantum physics than **BQTM**, but still the computational power is equivalent to that of QTM and **BQTM**. Since, the main difference between **BQTM** and **BQTM*** is the consistency with quantum physics, it is better to replace **BQTM** with **BQTM***.

4.3 Polynomial Time Computable Numbers

4.3.1 Introduction

The computable numbers have already appeared in Turing's paper [41] in which he defined the Turing machine. Turing defined them in real numbers, the definition is naturally extended to complex numbers. Rice showed that the whole complex computable numbers forms an algebraically closed field [36]. The field is strictly larger than the algebraic number fields $\overline{\mathbb{Q}}$ since it contains transcendental numbers such as π , but is strictly smaller than the complex field \mathbb{C} since it has only countable cardinality.

On the other hand, the polynomial time computability was getting more importance as a characteristic property of feasibly computable functions in the theory of computation. Our main subject, the polynomial time computable numbers, can be, thus, considered as a definition of the feasibly computable numbers. However, the interests it attracts are few; there is only a paper of Ko [15], in which he investigates several ways of defining the polynomial time computable numbers.

In the paper, we shall follow the direction of Rice; we shall investigate algebraic properties of the set of whole polynomial time computable numbers, and in fact it forms an algebraically closed field. The main theorem we shall show is the following:

Theorem 5 (main theorem). *The set of whole polynomial time computable numbers forms an algebraically closed field.*

In the next section, we shall clarify the definition of computable numbers and discuss about the relation with Ko's definition. The fact that the whole polynomial time computable numbers forms a field shall be shown in section 4.3.3.

Then, section 4.3.4 shall be the proof of the main theorem. Finally in section 4.3.5, we shall show the fact that the circle ratio π is a polynomial time computable numbers to prove that the set of polynomial time computable numbers is a proper superset of the field of algebraic numbers.

4.3.2 Definitions

At first, we introduce some notations and recall well-known results.

\mathbb{N} , \mathbb{Z} , \mathbb{Q} , \mathbb{R} and \mathbb{C} denote the set of natural numbers, the set of integers, the rational field, the real field and the complex field, respectively.

Theory of Computation

The first things we recall are terminologies in theory of computation.

Definition 11. We call a function a *computable* function if there exists a deterministic Turing machine which terminates with the result of the function for each input.

Definition 12. We call a function a *polynomial time computable* function if there exists a deterministic Turing machine which terminates with the result of the function in time of a polynomial of the input size for each input.

From now for simplicity, \mathcal{C} denotes the class of computable functions, and \mathcal{P} the class of polynomial time computable functions.

Computable number means that it can be approximated by a computable function, and polynomial time computable number by a polynomial time computable function. There are several ways to define them as definitions of real number: the Cauchy sequence, the Dedekind cut, etc. We choose a way to define them through giving approximation fractions. Accurately, the following is our definition.

Definition 13. A *computable number* is a complex number z that there exist two computable functions f and g from \mathbb{N} to \mathbb{Z} such that

$$\left| z - \frac{f(n) + g(n)i}{n} \right| \leq \frac{1}{n}$$

is satisfied for any natural number n greater than 1. Especially, if z is a real number, we call it a *computable real number*.

Definition 14. A *polynomial time computable number* is a complex number z that there exist two polynomial time computable functions f and g from \mathbb{N} to \mathbb{Z} such that

$$\left| z - \frac{f(n) + g(n)i}{n} \right| \leq \frac{1}{n}$$

is satisfied for any natural number n greater than 1. Especially, if z is a real number, we call it a *polynomial time computable real number*.

Let $\mathbb{C}_{\mathcal{C}}$ denote the set of whole computable numbers, and $\mathbb{C}_{\mathcal{P}}$ the set of whole polynomial time computable numbers. In case we restrict to real numbers $\mathbb{R}_{\mathcal{C}}$ and $\mathbb{R}_{\mathcal{P}}$ denote the set of computable real numbers and polynomial time computable real numbers respectively.

Comparison with Ko's Definition

We compare our definition of polynomial time computable real number and Ko's definition in [15].

Definition 15 (Ko's polynomial time computable real number). $\phi : \mathbb{N} \rightarrow \{0, 1\}^*$ *binary converge* to $\alpha \in [0, 1]$ means that for any $n \in \mathbb{N}$

$$\log_2(\phi(n)) = n \wedge |\phi(n)2^{-n} - \alpha| \leq 2^{-n}$$

is satisfied. α is *polynomial time computable* if there is a computable function ϕ binary converge to α in time of a polynomial of n .

The most significant difference is that Ko defines it only in the range $[0, 1]$. Another point is that because Ko utilizes only 2^n for denominators, the input n have to be interpreted unnaturally as if it is given by unary expansion. Our definition uses functions which give numerators of approximation for any denominators. We can show that the definitions are equivalent in the range $[0, 1]$. For the proof, we define the following sub-sequence of the natural numbers.

Definition 16. A sub-sequence of the natural numbers S is a *polynomially increasing sequence* if and only if there exist a polynomial time computable function ϕ to enumerate each element $s_i = \phi(i) \in S$ monotonically increasingly and a polynomial p satisfying $s_i < s_{i+1} \leq p(s_i)$ for any i .

For example, an arithmetic sequence with a positive integer difference or a geometric sequence with a positive integer ratio are polynomially increasing sequences. Especially, the sequence $\{2^i\}$ appearing in Ko's definition is a polynomially increasing sequence.

Lemma 2. *A complex number z is a polynomial time computable number if there exist polynomial time computable functions \hat{f}, \hat{g} from a polynomially increasing sequence S to \mathbb{Z} satisfying*

$$\left| z - \frac{\hat{f}(n) + \hat{g}(n)i}{n} \right| \leq \frac{1}{n}$$

for any $n \in S$.

Proof. We prove the lemma by constructing polynomial time computable functions f, g to define z .

At first, for an element s of S we let $f(s) = \hat{f}(s)$ and $g(s) = \hat{g}(s)$. For any $m \in \mathbb{N} \setminus S$, we choose $n \in S$ satisfying $(2 + \sqrt{2})m \leq n$. The size of n can be estimated as $n < p((2 + \sqrt{2})m)$ by the definition of S . We let f, g be

$f(m) = \lfloor \frac{\hat{f}(n)m}{n} \rfloor$, $g(m) = \lfloor \frac{\hat{g}(n)m}{n} \rfloor$, then f and g are polynomial time computable functions defining z . Actually,

$$f(m) = \left\lfloor \frac{\hat{f}(n)m}{n} \right\rfloor = \frac{\hat{f}(n)m + \delta_1}{n}$$

$$g(m) = \left\lfloor \frac{\hat{g}(n)m}{n} \right\rfloor = \frac{\hat{g}(n)m + \delta_2}{n}$$

by letting δ_1, δ_2 denote adjustment terms for the roundings. Then,

$$|\delta_i| \leq \frac{n}{2} \quad (i = 1, 2)$$

is satisfied and the total error is estimated as the following.

$$\begin{aligned} & \left| z - \frac{\left\lfloor \frac{\hat{f}(n)m}{n} \right\rfloor + \left\lfloor \frac{\hat{g}(n)m}{n} \right\rfloor i}{m} \right| \\ &= \left| z - \frac{\frac{\hat{f}(n)m}{n} + \frac{\hat{g}(n)m}{n}i}{m} + \frac{\delta_1 + \delta_2 i}{nm} \right| \\ &\leq \frac{1}{n} + \left| \frac{1+i}{2m} \right| = \frac{1}{n} + \frac{1}{\sqrt{2}m} \\ &\leq \frac{1}{(2 + \sqrt{2})m} + \frac{1}{\sqrt{2}m} \\ &= \frac{\sqrt{2}-1}{\sqrt{2}m} + \frac{1}{\sqrt{2}m} = \frac{1}{m} \end{aligned}$$

The functions f and g are polynomial time computable, because of the estimation $n < p((2 + \sqrt{2})m)$ already mentioned. \square

The following proposition is a direct consequence of the lemma.

Proposition 1. *Ko's definition of polynomial time computable real numbers is equivalent to the restriction of our definition of polynomial time computable numbers in the range $[0, 1]$.*

Another Equivalent Definition

To use later in proofs, we prepare another form of definition of polynomial time computable numbers as a lemma.

Lemma 3. *A complex number z is a polynomial time computable number if there exist two polynomial time computable functions F and G from \mathbb{N} to \mathbb{Q} satisfying*

$$|z - (F(n) + G(n)i)| \leq \frac{1}{n}$$

for any natural number n .

Proof. We assume the existence of F and G satisfying the condition. It is sufficient to show that the denominators for n , which are not necessarily n for $F(n)$ or $G(n)$, can be n .

We let $f(n) = \lfloor nF(4n) \rfloor$, $g(n) = \lfloor nG(4n) \rfloor$, and assume $F(4n) = \frac{p_1}{q_1}$, $G(4n) = \frac{p_2}{q_2}$. Then, there exist d_i such that

$$\begin{aligned} f(n) &= \frac{p_1 n + d_1}{q_1} \\ g(n) &= \frac{p_2 n + d_2}{q_2} \end{aligned}$$

and d_i satisfy $|d_i| \leq \frac{q_i}{2}$.

$$\begin{aligned} \left| z - \frac{f(n) + g(n)i}{n} \right| &= \left| z - \frac{p_1 n + d_1}{q_1 n} + \frac{p_2 n + d_2}{q_2 n} i \right| \\ &= \left| z - (F(4n) + G(4n)i) - \frac{d_1}{q_1 n} - \frac{d_2}{q_2 n} i \right| \\ &\leq |z - (F(4n) + G(4n)i)| + \left| \frac{d_1}{q_1 n} + \frac{d_2}{q_2 n} i \right| \\ &\leq \frac{1}{4n} + \left| \frac{1}{2n} + \frac{1}{2n} i \right| \\ &= \frac{1}{4n} + \frac{1}{\sqrt{2}n} \\ &< \frac{1}{n} \end{aligned}$$

Since it is obvious that f and g are polynomial time computable functions, f and g satisfy the conditions of the definition 14. \square

Algebra

Finally, let's recall the definition of the algebraically closed fields.

Definition 17. An *algebraically closed field* is a field K satisfying one of the following equivalent conditions.

1. Each element of $K[X]$ has at least one root in K .
2. Each element of $K[X]$ can be factored into linear factors in $K[X]$.

Popular examples of algebraically closed fields are the field of algebraic numbers $\overline{\mathbb{Q}}$ and the complex number field \mathbb{C} . As already mentioned, it is shown by Rice that the whole computable numbers form an algebraically closed field.

4.3.3 The Field of Polynomial Time Computable Numbers

In this section, we show that the whole polynomial time computable numbers $\mathbb{C}_{\mathcal{P}}$ is a field. For ease at the beginning, we start from proving that the $\mathbb{R}_{\mathcal{P}}$ is a field.

Lemma 4. *The $\mathbb{R}_{\mathcal{P}}$ is a subfield of the $\mathbb{R}_{\mathcal{C}}$.*

Proof. Since it is clear that the $\mathbb{R}_{\mathcal{P}}$ is a subset of the $\mathbb{R}_{\mathcal{C}}$, it only needs to show the lemma that the $\mathbb{R}_{\mathcal{P}}$ is closed under the four arithmetic operations: additions, multiplications, negations and inversions.

Let λ and μ be elements of $\mathbb{R}_{\mathcal{P}}$. By definition, there exist polynomial time computable functions f and g from \mathbb{N} to \mathbb{Z} , and they satisfy:

$$\left| \lambda - \frac{f(n)}{n} \right| \leq \frac{1}{n}$$

and

$$\left| \mu - \frac{g(n)}{n} \right| \leq \frac{1}{n}$$

each for any $n \geq 1$.

The first thing we show is the closedness under additions, i.e. $\lambda + \mu$ is an element of the $\mathbb{R}_{\mathcal{P}}$. By defining a function S from the natural numbers to the rational numbers by

$$S(n) = \frac{f(2n) + g(2n)}{2n},$$

it is a direct consequence of the fact that f and g are polynomial time computable functions that S is a function of polynomial time of input size $\log n$ for the input n .

$$\begin{aligned} & |\lambda + \mu - S(n)| \\ = & \left| \lambda + \mu - \frac{f(2n) + g(2n)}{2n} \right| \\ = & \left| \lambda - \frac{f(2n)}{2n} + \mu - \frac{g(2n)}{2n} \right| \\ \leq & \left| \lambda - \frac{f(2n)}{2n} \right| + \left| \mu - \frac{g(2n)}{2n} \right| \\ \leq & 2 \cdot \frac{1}{2n} = \frac{1}{n} \end{aligned}$$

Therefore, $\lambda + \mu$ has a defining polynomial time computable function from \mathbb{N} to \mathbb{Q} , and it is a polynomial time computable real number from the lemma 3. Thus, $\mathbb{R}_{\mathcal{P}}$ is closed under additions.

It is trivial to show the closedness under negation, or flipping the sign.

The next thing to show is the closedness under multiplications. Let P be a function from \mathbb{N} to \mathbb{Q} s.t.

$$P(n) = \frac{f(cn)g(cn)}{c^2n^2},$$

where c is a constant $|f(1)| + |g(1)| + 4$. It is clear that P is polynomial time computable.

$$\begin{aligned} & |\lambda\mu - P(n)| \\ = & \left| \lambda\mu - \frac{f(cn)g(cn)}{c^2n^2} \right| \\ = & \left| \left(\lambda - \frac{f(cn)}{cn} \right) \left(\mu - \frac{g(cn)}{cn} \right) + \frac{f(cn)}{cn} \left(\mu - \frac{g(cn)}{cn} \right) + \frac{g(cn)}{cn} \left(\lambda - \frac{f(cn)}{cn} \right) \right| \\ \leq & \left| \lambda - \frac{f(cn)}{cn} \right| \left| \mu - \frac{g(cn)}{cn} \right| + \left| \frac{f(cn)}{cn} \right| \left| \mu - \frac{g(cn)}{cn} \right| + \left| \frac{g(cn)}{cn} \right| \left| \lambda - \frac{f(cn)}{cn} \right| \\ \leq & \frac{1}{c^2n^2} + \left| \frac{f(cn)}{cn} \right| \frac{1}{cn} + \left| \frac{g(cn)}{cn} \right| \frac{1}{cn} \end{aligned}$$

Here, f satisfies $\left| \frac{f(cn)}{cn} \right| \leq |f(1)| + \frac{3}{2}$ and so g does.

$$\begin{aligned} & \leq \frac{1}{c^2n^2} \left(1 + \left(cn|f(1)| + \frac{3cn}{2} \right) + \left(cn|g(1)| + \frac{3cn}{2} \right) \right) \\ & = \frac{cn}{c^2n^2} \left(|f(1)| + |g(1)| + 3 + \frac{1}{cn} \right) \\ & \leq \frac{c^2n}{c^2n^2} = \frac{1}{n} \end{aligned}$$

Therefore, P is a polynomial time computable function from \mathbb{N} to \mathbb{Q} defining $\lambda\mu$. Again by lemma 3, $\lambda\mu$ is a polynomial time computable real number, and $\mathbb{R}_{\mathcal{P}}$ is closed under multiplications.

At last, we show the closedness under inversions. Let λ be a polynomial time computable real number, which is not 0. There exists a natural number k such that $|f(k)| > 1$, since λ is not 0. With defining a polynomial p with the k as

$$p(X) = 2k^2X + k,$$

we define a polynomial time computable function I from \mathbb{N} to \mathbb{Q} by

$$I(n) = \frac{p(n)}{f(p(n))}.$$

Then,

$$\begin{aligned}
& |\lambda^{-1} - I(n)| \\
&= \left| \lambda^{-1} - \frac{p(n)}{f(p(n))} \right| \\
&= |\lambda^{-1}| \left| \frac{p(n)}{f(p(n))} \right| \left| \frac{f(p(n))}{p(n)} - \lambda \right| \\
&\leq |\lambda^{-1}| \left| \frac{p(n)}{f(p(n))} \right| \frac{1}{p(n)} \\
&\leq \left(\frac{|f(k)|}{k} - \frac{1}{k} - \frac{1}{p(n)} \right)^{-2} \frac{1}{p(n)} \\
&= \frac{k^2 p(n)}{(p(n) (|f(k)| - 1) - k)^2}
\end{aligned}$$

Here, $|f(k)| - 1 \geq 1$ holds.

$$\begin{aligned}
&\leq \frac{k^2 p(n)}{(p(n) - k)^2} \\
&= \frac{2k^4 n + k^3}{(2k^2 n)^2} \\
&\leq \frac{3k^4 n}{4k^4 n^2} \\
&< \frac{1}{n}
\end{aligned}$$

Therefore, I is a polynomial time computable function from \mathbb{N} to \mathbb{Q} defining λ^{-1} . Once again by lemma 3, λ^{-1} is a polynomial time computable real number, and $\mathbb{R}_{\mathcal{P}}$ is closed under inversion. \square

By the lemma above, $\mathbb{R}_{\mathcal{P}}$ is a field. The next lemma clarifies the relationship between $\mathbb{R}_{\mathcal{P}}$ and $\mathbb{C}_{\mathcal{P}}$.

Lemma 5. *A complex number $z = x + yi$ is an element of $\mathbb{C}_{\mathcal{P}}$ if and only if both its real part x and its imaginary part y are elements of $\mathbb{R}_{\mathcal{P}}$.*

Proof. $z = x + yi \in \mathbb{C}_{\mathcal{P}}$ clearly implies to $x \in \mathbb{R}_{\mathcal{P}} \wedge y \in \mathbb{R}_{\mathcal{P}}$ by the definition.

Conversely, we assume $x \in \mathbb{R}_{\mathcal{P}} \wedge y \in \mathbb{R}_{\mathcal{P}}$. Let f (g) be a defining function of x (y resp.). Moreover, we define two functions ξ and η :

$$\xi(n) = \frac{f(3n) + \kappa_1}{3}$$

$$\eta(n) = \frac{g(3n) + \kappa_2}{3},$$

where κ_1 and κ_2 are adjustment terms to keep the values of ξ and η respectively in integers, and thus their absolute values are at most 1. Then,

$$\begin{aligned}
& \left| x + yi - \frac{\xi(n) + \eta(n)i}{n} \right| \\
= & \left| x + yi - \frac{f(3n) + g(3n)i + \kappa_1 + \kappa_2 i}{3n} \right| \\
= & \sqrt{\left(x - \frac{f(3n) + \kappa_1}{3n} \right)^2 + \left(y - \frac{g(3n) + \kappa_2}{3n} \right)^2} \\
\leq & \frac{1}{3n} \sqrt{1 - 2\kappa_1(3nx - f(3n)) + \kappa_1^2 + 1 - 2\kappa_2(3ny - g(3n)) + \kappa_2^2} \\
\leq & \frac{1}{3n} \sqrt{(\kappa_1 + 1)^2 + (\kappa_2 + 1)^2} \\
\leq & \frac{\sqrt{8}}{3n} \\
< & \frac{1}{n}
\end{aligned}$$

Therefore, $z = x + yi$ belongs to $\mathbb{C}_{\mathcal{P}}$. \square

The two lemmas above imply the following theorem.

Theorem 6. *The whole set of polynomial time computable numbers $\mathbb{C}_{\mathcal{P}}$ forms a field.*

Proof. From the lemma 5, both real and imaginary parts of an element of $\mathbb{C}_{\mathcal{P}}$ are elements of $\mathbb{R}_{\mathcal{P}}$. All of four arithmetic operations of $\mathbb{C}_{\mathcal{P}}$ are defined only from the four arithmetic operations of real and imaginary parts. By the lemma 4, $\mathbb{R}_{\mathcal{P}}$ is closed under the four arithmetic operations. Thus, both real and imaginary parts of the result of operations in $\mathbb{C}_{\mathcal{P}}$ are in $\mathbb{R}_{\mathcal{P}}$. From the lemma 5 again, the result is in $\mathbb{C}_{\mathcal{P}}$. It means that $\mathbb{C}_{\mathcal{P}}$ is closed under all of four operations, and $\mathbb{C}_{\mathcal{P}}$ forms a field. \square

4.3.4 Main Theorem

We shall, in this section, prove the main theorem already stated.

Theorem 1. *The set of whole polynomial time computable numbers $\mathbb{C}_{\mathcal{P}}$ forms an algebraically closed field.*

In the previous section, we have already shown that $\mathbb{C}_{\mathcal{P}}$ is a field, and it is sufficient to show the algebraically closedness of the field. By the definition 17, it is the subject to prove that any $\mathbb{C}_{\mathcal{P}}$ coefficient polynomials have a root in $\mathbb{C}_{\mathcal{P}}$ or they are factored into linear $\mathbb{C}_{\mathcal{P}}$ coefficient factors. Note that existence of a root or factorization into linear factors are sufficient to prove, and it is not necessary to show that any given polynomial will be factored into linear factors *in polynomial time*.

Moreover, we can exclude polynomial with double roots transcendently. When one thinks about algebraic extensions, all roots including double roots can be constructed from single roots.

An outline of the proof will be as follows. Think about all roots of given $\mathbb{C}_{\mathcal{P}}$ coefficient polynomials f . There are two factors of the errors in computation of the roots of f . The first factor is from expressing the coefficients in finite precisions. The second factor is from terminating an algorithm of root finding at some precision. It is, therefore, sufficient to show the theorem that making the errors from both factors in a desired precision takes at most a polynomial time of the precision. We shall show the following lemma for the approximations of coefficients.

Lemma 6. *Let f be a given $\mathbb{C}_{\mathcal{P}}$ coefficient monic polynomial. For any natural number m , it takes at most polynomial time of $\log m$ to compute all coefficients of an approximation polynomial¹ \tilde{f} of f so that roots $\{\rho_i\}$ of f and $\{\tilde{\rho}_i\}$ of \tilde{f} satisfy*

$$|\rho_i - \tilde{\rho}_i| \leq \frac{1}{m}$$

for any i if appropriately arranged.

For the root approximations, we need two more lemmas. Before stating the lemmas, let us name the condition that appears in the lemmas.

Definition 18. We call the following condition of a complex number ζ for a polynomial f *converging initial condition*.

- There is an open convex set D to which ζ belongs, and there exists a real number $L > 0$ such that for any $z_1, z_2 \in D$ it satisfies

$$|f'(z_1) - f'(z_2)| \leq L|z_1 - z_2|.$$

- $f'(\zeta) \neq 0$ holds, and there are real numbers a, b such that $|f'(\zeta)^{-1}| \leq a$, $|f'(\zeta)^{-1}f(\zeta)| \leq b$ and $h = abL \leq \frac{1}{2}$.
- A closed disc \overline{U} with radius $t^* = (1 - \sqrt{1 - 2h})/(aL)$ centered at ζ or:

$$\overline{U} = \{z \in \mathbb{C}; |z - \zeta| \leq t^*\}$$

is in D .

The two lemmas follow.

Lemma 7. *Let f be a $\mathbb{Q}[i]$ coefficient monic polynomial, which has no multiple roots. If $\rho^{(0)}$ satisfying converging initial condition for f is given, then for any natural number m , it takes at most polynomial time of $\log m$ to compute an approximation $\tilde{\rho}$ of the root ρ of f to satisfy*

$$|\rho - \tilde{\rho}| \leq \frac{1}{m}.$$

¹We call a polynomial \tilde{f} an approximation polynomial of f if all coefficients of \tilde{f} are obtained from computing the defining function of the coefficients of f .

Lemma 8. *Let f_1, f_2, \dots be a series of polynomials uniformly converging to a polynomial $f \in \mathbb{C}[X]$ with no double roots in any compact region. Then, there exists m_0 such that there exists $\rho^{(0)}$ which satisfies converging initial condition for any f_m ($m > m_0$).*

Proof of the Main Theorem

Firstly, we prove the main theorem assuming the validity of the lemmas.

proof of the theorem 5. We prove the theorem by showing that each $\mathbb{C}_{\mathcal{P}}$ coefficient polynomial has at least one root in $\mathbb{C}_{\mathcal{P}}$.

The case of degree 1 is trivial, thus we assume that f is a polynomial of degree $d > 1$. Without loss of generality, we can assume that f is monic. Moreover, we can assume that f is irreducible over a field over \mathbb{Q} generated by all coefficients of f . Otherwise, f can be factored into lower degree irreducible polynomials.

Then, by the lemma 6, for any m , an approximation polynomial \tilde{f}_m , whose roots are distant at most $\frac{1}{2m}$ from each of the roots of f , can be computed in polynomial time of $\log 2m$, and thus of $\log m$.

Since the approximation polynomials $\{\tilde{f}_m\}_{m=1}^{\infty}$ of f uniformly converge to f in any compact regions of \mathbb{C} , there exists a number m_0 such that there exists $\rho^{(0)}$ which satisfies converging initial condition for any \tilde{f}_m ($m > m_0$) by the lemma 8. By the lemma 7, then, we can compute an approximation of a root of \tilde{f}_m in precision of $\frac{1}{2m}$ from $\rho^{(0)}$ in polynomial time of $\log 2m$ and thus of $\log m$ again. The cases of $m \leq m_0$ are ignorable since they are only finite numbers.

As a consequence, a root of f can be computed for any natural number m in polynomial time of $\log m$ with an error at most $\frac{1}{m}$. Therefore, it is an element of $\mathbb{C}_{\mathcal{P}}$. \square

The rest of the section consists of proofs of the lemmas.

Proof of the Lemma 6

In this section, we prove the lemma 6. In the proof, the following theorem plays the central role.

Theorem 3 (Ostrowski). *[12, section 2.3] Let f and g be two different complex coefficient polynomials.*

$$\begin{aligned} f(X) &= X^n + a_{n-1}X^{n-1} + \dots + a_1X + a_0 \\ g(X) &= X^n + b_{n-1}X^{n-1} + \dots + b_1X + b_0 \end{aligned}$$

Moreover, γ denotes a real constant:

$$\gamma = 2 \max(\{|a_{n-j}|^{1/j}\} \cup \{|b_{n-j}|^{1/j}\})$$

and ϵ denotes a positive real number satisfies:

$$\epsilon^n = \sum_{j=0}^{n-1} |b_j - a_j| \gamma^j.$$

Then, zeros z_k (w_k) of f (g resp.) can be arranged to satisfy

$$|z_j - w_j| < 2n\epsilon.$$

proof of the lemma 6. We write the given polynomial f explicitly:

$$f(X) = X^n + a_{n-1}X^{n-1} + \cdots + a_1X + a_0$$

with a_i in $\mathbb{C}_{\mathcal{P}}$. Let \tilde{f}

$$\tilde{f}(X) = X^n + \tilde{a}_{n-1}X^{n-1} + \cdots + \tilde{a}_1X + \tilde{a}_0$$

be an approximation polynomial of f .

Let $\gamma_{f,\tilde{f}}$ and $\epsilon_{f,\tilde{f}}$ be:

$$\begin{aligned} \gamma_{f,\tilde{f}} &= 2 \max(\{|a_{n-j}|^{1/j}\} \cup \{|\tilde{a}_{n-j}|^{1/j}\}) \\ \epsilon_{f,\tilde{f}}^n &= \sum_{j=0}^{n-1} |\tilde{a}_j - a_j| \gamma_{f,\tilde{f}}^j. \end{aligned}$$

Then, by the theorem 3, the roots $\{\rho_i\}$ of f and $\{\tilde{\rho}_i\}$ of \tilde{f} satisfy the following inequality:

$$|\rho_i - \tilde{\rho}_i| < 2n\epsilon_{f,\tilde{f}}$$

with an appropriate arrangement.

At first, we estimate $\gamma_{f,\tilde{f}}$. Since \tilde{a}_{n-j} is calculated from the defining function of a_{n-j} , $|\tilde{a}_{n-j}| < |a_{n-j}| + 1$ holds. Then,

$$\gamma_{f,\tilde{f}} < 2 \max\{(|a_{n-j}| + 1)^{1/j}\}$$

and by letting γ denote the right hand side, $\gamma \geq 1$ holds.

Secondly, we estimate $\epsilon_{f,\tilde{f}}$. By replacing $\gamma_{f,\tilde{f}}$ in the estimation of $\epsilon_{f,\tilde{f}}$ by γ , it holds that:

$$\epsilon_{f,\tilde{f}}^n = \sum_{j=0}^{n-1} |\tilde{a}_j - a_j| \gamma_{f,\tilde{f}}^j < \sum_{j=0}^{n-1} |\tilde{a}_j - a_j| \gamma^j.$$

If for any j

$$|\tilde{a}_j - a_j| \gamma^j \leq \frac{1}{k}$$

hold, then

$$\epsilon_{f,\tilde{f}} < \sqrt[n]{\frac{n}{k}}$$

is implied.

To make the all differences of the roots be smaller than $\frac{1}{m}$, using the theorem

$$|\rho_i - \tilde{\rho}_i| < 2n\epsilon_{f,\bar{f}}$$

and the assumption above, we can obtain:

$$\begin{aligned} 2n\epsilon_{f,\bar{f}} < 2n\sqrt[n]{\frac{n}{k}} &\leq \frac{1}{m} \\ k^{\frac{1}{n}} &\geq 2n^{1+\frac{1}{n}}m \\ k &\geq 2^n n^{n+1} m^n. \end{aligned}$$

Therefore, it is sufficient if computation is carried out to the place $|\tilde{a}_j - a_j|\gamma^j$ is smaller than $(2^n n^{n+1} m^n)^{-1}$. The errors in coefficients themselves are estimated as

$$|\tilde{a}_j - a_j| \leq \frac{1}{k\gamma^j} \leq \frac{1}{2^n n^{n+1} m^n \gamma^j}.$$

Since a_j of a coefficient of f is a polynomial time computable number, it takes at most polynomial steps of $\log(2^n n^{n+1} \gamma^j m^n)$ to calculate \tilde{a}_j with error at most $(2^n n^{n+1} \gamma^j m^n)^{-1}$. Because n and γ are constant depending only on f , the total time complexity is a polynomial of $\log m$. For all other coefficients, the estimations are similar, thus the total complexity is a sum of n polynomials, i.e. the total complexity is polynomial time. \square

Proof of the Lemma 7

In this section, we prove the lemma 7. In the proof, the following theorem is used.

Theorem 4 (Kantorovich). [40, section 4.3] Let $f(\mathbf{x})$ be a differentiable function defined in a open convex set D of \mathbb{R}^n , $\mathbf{x}^{(0)}$ be in D . Assume the following conditions are satisfied.

- Jacobian $J(\mathbf{x})$ is Lipschitz continuous on D , i.e. it satisfies:

$$\|J(\mathbf{x}) - J(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$$

where $\mathbf{x}, \mathbf{y} \in D$ and $L > 0$.

- $J(\mathbf{x}^{(0)})$ is regular and it satisfies $\|J(\mathbf{x}^{(0)})^{-1}\| \leq a$, $\|J(\mathbf{x}^{(0)})^{-1}f(\mathbf{x}^{(0)})\| \leq b$ and $h = abL \leq 1/2$.
- An open ball \bar{U} centered at $\mathbf{x}^{(0)}$ and with the diameter $t^* = (a - \sqrt{1 - 2h})/(aL)$

$$\bar{U} = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{x}^{(0)}\| \leq t^*\}$$

is in D .

Then,

- There exists in \bar{U} only a solution \mathbf{x}^* of $f(\mathbf{x}) = \mathbf{0}$.
- An approximation sequence of the Newton method $\{\mathbf{x}^{(\nu)}\}$ with the initial value $\mathbf{x}^{(0)}$ is defined, and then $\mathbf{x}^{(\nu)} \in D$ and

$$\|\mathbf{x}^{(\nu)} - \mathbf{x}^*\| \leq \frac{(1 - \sqrt{1 - 2h})^{2^\nu}}{2^\nu aL}$$

hold for any ν .

proof of the lemma 7. From the formula of the Newton method giving an approximation sequence $\{\rho^{(\nu)}\}$ of a root starting from $\rho^{(0)}$ is

$$\rho^{(\nu+1)} = \rho^{(\nu)} - \frac{f(\rho^{(\nu)})}{f'(\rho^{(\nu)})}$$

and it takes at most $O(\deg f)$ rational operations per step. Moreover, by the inequality of the theorem, we can estimate the number of iterations ν such that:

$$\frac{(1 - \sqrt{1 - 2h})^{2^\nu}}{2^\nu aL} \leq \frac{1}{m}$$

in terms of m . The fact that $h = \frac{1}{2}$ holds only if the approximating root is a multiple root implies $h < \frac{1}{2}$ since f is assumed not to have a multiple root, thus the constant $1 - \sqrt{1 - 2h}$ of the numerator is less than 1. The power of the numerator, therefore, dominates over the factor 2^ν of the denominator. As a consequence, we can use the following inequality for large ν :

$$\frac{(1 - \sqrt{1 - 2h})^{2^\nu}}{2^\nu aL} \leq \frac{(1 - \sqrt{1 - 2h})^{2^\nu}}{2aL}.$$

Then,

$$\begin{aligned} \frac{(1 - \sqrt{1 - 2h})^{2^\nu}}{2aL} &\leq \frac{1}{m} \\ (1 - \sqrt{1 - 2h})^{2^\nu} &\leq \frac{2aL}{m} \\ 2^\nu \log(1 - \sqrt{1 - 2h}) &\leq \log(2aL) - \log m \end{aligned}$$

Let a constant c_0 be

$$c_0 = \frac{1}{-\log(1 - \sqrt{1 - 2h})},$$

then,

$$\begin{aligned} 2^\nu &\geq c_0 \log m - c_0 \log(2aL) \\ \nu \log 2 &\geq \log(c_0 \log m - c_0 \log(2aL)) \\ &= \log \log m + \log c_0 + \log \left(1 - \frac{\log(2aL)}{\log m}\right) \\ \nu &\geq \frac{\log \log m}{\log 2} + \frac{\log c_0 + \log \left(1 - \frac{\log(2aL)}{\log m}\right)}{\log 2}. \end{aligned}$$

We can conclude that the time complexity to obtain an approximation $\rho^{(\nu)}$ distant from a root ρ of f at most $\frac{1}{m}$ starting from $\rho^{(0)}$ is $O(\log \log m)$ rational operations. Thus, it is polynomial time. \square

Proof of Lemma 8

In this section, we prove the last lemma.

Proof. Since the given f has no double roots and $\{f_i\}$ converges to f , a definition

$$\mu_0 = \max(\{0\} \cup \{m \in \mathbb{N}; f_m \text{ has double roots}\})$$

makes sense.

A root ρ of f is not a root of f' , i.e. $f'(\rho) \neq 0$, since again f has no double roots. In the following proof, we shall fix a root ρ of f and let w be $|f'(\rho)|$. Choose and fix an arbitrary real number α in a range $0 < \alpha < 1$. Then, there are at most finitely many m satisfying $|f'_m(\rho)| < \alpha w$, because $\{f'_i(\rho)\}$ converge to $f'(\rho)$. Let

$$\mu_1 = \max(\{\mu_0\} \cup \{m; |f'_m(\rho)| < \alpha w\})$$

and

$$A(\mu_1, \alpha) = \bigcup_{m > \mu_1} \{z \in \mathbb{C}; |f'_m(z)| < \alpha w\}.$$

Then, ρ does not belong to $A(\mu_1, \alpha)$ by the definition of μ_1 .

The next step is to determine an open convex set D to which ρ belongs. Let δ_A be the distance between ρ and $A(\mu_1, \alpha)$:

$$\delta_A = \inf\{|\rho - z|; z \in A(\mu_1, \alpha)\}$$

and

$$D = D(\delta_A) = \{z \in \mathbb{C}; |\rho - z| < \delta_A\}.$$

It is clear that f and all of $\{f_i\}$ are Lipschitz continuous in D , since they are polynomials. Actually, L can be:

$$L = \max\left(\frac{1}{2}, \sup\{|f''_m(z)|; z \in D \wedge m > \mu_1\}\right).$$

To satisfy the rest of the conditions, let $\delta_0 = \min(\delta_A, \frac{2\alpha w}{L})$ and let γ be 1 if $\delta_0 \neq \frac{3\alpha w}{2L}$ or an arbitrarily chosen real number satisfying $\frac{1}{2} \leq \gamma < 1$ otherwise. Besides, we define β as:

$$\beta = \gamma \left(\frac{2\delta_0}{3w} - \frac{2L\delta_0^2}{9\alpha w^2} \right).$$

Then $h = \alpha^{-1}\beta L < \frac{1}{2}$. Actually,

$$\begin{aligned} h &= \alpha^{-1}\beta L \\ &= \alpha^{-1}L\gamma \left(\frac{2\delta_0}{3w} - \frac{2L\delta_0^2}{9\alpha w^2} \right) \\ &= \gamma \frac{2L\delta_0}{3\alpha w} \left(1 - \frac{L\delta_0}{3\alpha w} \right) \end{aligned}$$

By writing $\xi = \frac{L\delta_0}{3\alpha w}$, we have:

$$h = 2\gamma\xi(1 - \xi)$$

and by solving $h < \frac{1}{2}$ we obtain $\xi \neq \frac{1}{2}$. Therefore, if $\delta_0 \neq \frac{3\alpha w}{2L}$ then $h < \frac{1}{2}$ holds. On the other hand, if $\delta_0 = \frac{3\alpha w}{2L}$ then $h = \frac{\gamma}{2}$ and by the definition of γ we have $h < \frac{1}{2}$.

We think a set determined with the β and a parameter μ :

$$B(\mu, \beta) = \bigcup_{m > \mu} \{z \in \mathbb{C}; |f'_m(z)^{-1} f_m(z)| > \beta w\}$$

and we would like to settle μ so that ρ does not belong to $B(\mu, \beta)$. Transformation of the condition of each set consisting $B(\mu, \beta)$ gives $|f_m(z)| > \beta w |f'_m(z)|$. Since we know that $\rho \notin A(\mu_1, \alpha)$, by letting $\mu \geq \mu_1$ it is sufficient for the condition to be satisfied outside $A(\mu_1, \alpha)$. A condition $|f'_m(z)| \geq \alpha w$ is satisfied outside $A(\mu_1, \alpha)$, thus

$$|f_m(z)| > \beta w |f'_m(z)| > \alpha \beta w^2.$$

Especially, at ρ , there are at most finitely many m to satisfy $|f_m(\rho)| > \alpha \beta w^2$ since $\{f_i\}$ converge to f . Then, with

$$\mu_2 = \mu_2(\beta) = \max(\{\mu_1\} \cup \{m; |f_m(\rho)| > \alpha \beta w^2\}),$$

ρ does not belong to $B(\mu_2, \beta)$. We let δ_B denote the distance between ρ and $B(\mu_2, \beta)$:

$$\delta_B = \delta_B(\beta) = \inf\{|\rho - z|; z \in B(\mu_2, \beta)\}.$$

Finally, let $\hat{\gamma}$ be $1 - \sqrt{1 - \gamma}$ and

$$\delta = \min(\delta_B, \frac{\hat{\gamma}\delta_0}{3}).$$

Then, with a point $\rho^{(0)}$ inside the disc centered at ρ with the radius δ , a closed disc \bar{U} centered at $\rho^{(0)}$ with the radius $\frac{2\hat{\gamma}\delta_0}{3}$ is contained in D and ρ belongs to \bar{U} .

Consequently, by choosing m_0 as μ_2 , $\rho^{(0)}$ satisfies the converging initial condition for f_m for any $m > m_0$. \square

4.3.5 Transcendental Numbers

To show that the field of polynomial time computable numbers $\mathbb{C}_{\mathcal{P}}$ contains a part of transcendental numbers, we demonstrate that the circle ratio π is a polynomial time computable number.

Proposition 2. *The circle ratio π is a polynomial time computable number.*

Proof. There are numbers of methods to compute the circle ratio. We choose Machin's formula:

$$\frac{\pi}{4} = 4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right).$$

Since a sum of polynomial time computable numbers is also a polynomial time computable number as shown in the lemma 4, it is sufficient to show that for any $k > 1$, $\arctan\left(\frac{1}{k}\right)$ is a polynomial time computable number. The Taylor expansion of \arctan for $|x| < 1$ is:

$$\arctan(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^{2i+1}}{2i+1}$$

and, by truncating to m terms, the error is at most $\frac{x^{2m+1}}{(2m+1)}$. Ignoring the numerator for simplicity, m for at most $\frac{1}{n}$ error can be estimated as the following.

$$\begin{aligned} \left(\frac{1}{k}\right)^{2m+1} &< n^{-1} \\ (2m+1) \log\left(\frac{1}{k}\right) &< -\log n \\ 2m+1 &> \frac{\log n}{\log k} \\ m &> \frac{\log n - 1}{2 \log k} \end{aligned}$$

Thus the number of terms can be at most $\log n$.

The computations of evaluating the m term expansion of $\arctan\left(\frac{1}{k}\right)$ is estimated as follows. We compute

$$\left| \arctan\left(\frac{1}{k}\right) - \sum_{i=0}^{m-1} \frac{(-1)^i}{(2i+1)k^{2i+1}} \right| < \frac{1}{n}$$

by using a common denominator:

$$\frac{1}{(2m-1)!!k^{2m-1}} \sum_{i=0}^{m-1} (-1)^i k^{2m-2-2i} \prod_{j=0}^{m-1-i} (2j+1).$$

Then the size of the denominator is

$$\begin{aligned} \log((2m-1)!!k^{2m-1}) &< \log([(2m-1)k]^{2m-1}) \\ &= (2m-1)(\log(2m-1) + \log k) \\ &= O(\log n)O(\log \log n) = O(\log n \log \log n) \end{aligned}$$

and it requires about $2m$ multiplications. Thus, the number of steps is estimated as:

$$O(\log n)O((\log n \log \log n)^2) = O(\log^4 n).$$

Similarly, the number of steps to compute the numerator is $O(\log^4 n)$. Thus, the total number of steps is also $O(\log^4 n)$.

Therefore, $\arctan(\frac{1}{k})$ are polynomial time computable and the circle ratio π as a sum of them is a polynomial time computable number. \square

4.3.6 Concluding Remarks

We showed that the field of polynomial time computable numbers $\mathbb{C}_{\mathcal{P}}$ is an algebraically closed field. Because there are transcendental numbers including π in $\mathbb{C}_{\mathcal{P}}$, the field is a proper extension of $\overline{\mathbb{Q}}$ the algebraic closure of \mathbb{Q} . On the other hand, because the class of polynomial time computable functions does not contain any EXPTIME-complete functions by the hierarchy theorem [14, section 7], the field is a proper subfield of the field of whole computable numbers $\mathbb{C}_{\mathcal{C}}$. One may be interested in whether the subsets of $\mathbb{C}_{\mathcal{C}}$ corresponding to other complexity classes are also algebraically closed fields or not. By following the argument of this paper, it is easy to conclude that such sets corresponding to any classes containing \mathcal{P} are algebraically closed fields. We do not know about the proper subclasses of \mathcal{P} , even whether the corresponding sets are fields or not.

Bibliography

- [1] C. A. M. Antonio, K. Saito, S. Tanaka, J. S. Asuncion, and K. Nakamura. Implementation of imaginary quadratic fields and elliptic or hyperelliptic curves over finite prime fields on the system NZMATH for number theory. In *electronical proceedings of 6th Symposium on Algebra and Computation AC2005*, November 2005. <ftp://tnt.math.metro-u.ac.jp/pub/ac05/Antonio/proceedings.pdf>.
- [2] K. Atsumi and T. Nishino. Solving np-complete problems and factoring problems by using nmr quantum computation. *Transaction of Information Processing Society Japan*, 43(SIG 7 (TOM 6)):10–18, September 2002.
- [3] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Pearson Education Inc., 2nd edition, 2005.
- [4] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. <http://www.agilemanifesto.org/>.
- [5] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proceedings of 25th ACM symposium on Theory of computing*, pages 11–20, 1993.
- [6] H. Cohen. *A Course in Computational Algebraic Number Theory*. GTM 138. Springer-Verlag, 1993.
- [7] H. Cohen. *Advanced Topics in Computational Number Theory*. GTM 193. Springer-Verlag, 2000.
- [8] K. Fogel. *Open Source Development with CVS*. The Coriolis Group, 2000.
- [9] M.-N. Gras. Méthodes et algorithmes pour le calcul numérique du nombre de classes et des unités des extensions cubiques cycliques de \mathbf{Q} . *J. reine angew. Math.*, 277:89–116, 1975.
- [10] M. van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number Theory*, 95(2):167–189, 2002.

- [11] C. Hollinger and P. Serf. SIMATH — a computer algebra system. In *Computational Number Theory*, pages 331–342. Walter de Gruyter, Berlin, New York, 1991.
- [12] A. S. Householder. *The Numerical Treatment of a Single Nonlinear Equation*. McGraw-Hill, 1970.
- [13] The KANT Project. KANT / KASH. <http://www.math.tu-berlin.de/~kant/kash.html>.
- [14] T. Kasai. *Theory of Computational Complexity*. Kindaikagakusha, April 1987. (Japanese).
- [15] K.-I Ko. On the definitions of some complexity classes of real numbers. *Mathematical System Theory*, 16:95–109, 1983.
- [16] A. N. Kolmogorov, I. G. Zhurbenko, and A. V. Prokhorov. *Vvedenie v Teoriyu Veroiatnostei*. Nauka, Moscow, 2nd edition, 1995. T. Maruyama, Y. Baba (Japanese translation), Korumogorohu no kakuritsuronnyuumon, Morikita Shuppan, Tokyo, 2003.
- [17] H. Komai. Implementation of arithmetic of elliptic curves over finite prime fields on NZMATH. Master’s thesis, Tokyo Metropolitan University, 2005.
- [18] K. Kumaki. Implementation of multiple polynomial quadratic sieve on the number theoretic system NZMATH and its analysis. Master’s thesis, Tokyo Metropolitan University, 2005. (Japanese).
- [19] The Magma Computational Algebra System. <http://magma.maths.usyd.edu.au/magma/>.
- [20] T. Matsui. Development of computational number theory system by a scripting language. In M. Noro, editor, *Computer Algebra — Design of Algorithms, Implementations and Applications*, number 1395 in Suriken Kokyuroku, pages 144–149. Kyoto University Research Institute for Mathematical Sciences, October 2004.
- [21] T. Matsui. A note on bulk quantum turing machine. preprint, November 2004. arXiv:cs.CC/0411037.
- [22] T. Matsui. NZMATH: past and future of the development. In *electronical proceedings of 6th Symposium on Algebra and Computation AC2005*, 2005. <ftp://tnt.math.metro-u.ac.jp/pub/ac05/Matsui/nzmath.pdf>.
- [23] T. Matsui. Development of NZMATH. In A. Iglesias and N. Takayama, editors, *Mathematical Software — ICMS 2006*, volume 4151 of *Lecture Notes in Computer Science*, pages 158–169. Springer-Verlag, September 2006.
- [24] T. Matsui. On polynomial time computable numbers. preprint, August 2006. arXiv:cs.CC/0608067.

- [25] K. Nakamura. Class number computation by cyclotomic or elliptic units. In *Computational Number Theory*, pages 139–162. Walter de Gruyter, Berlin, New York, 1991.
- [26] K. Nakamura and T. Matsui. Developing a system for number theory by script language — announcement of the release of NZMATH 0.1.1. presentation on Algorithms and Number Theory, Dagstuhl, May 2004. available at <http://tnt.math.metro-u.ac.jp/~nakamura/talk/dag2004-a.pdf>.
- [27] K. Nishimoto and K. Nakamura. Computer experiment on key generation for the quantum public key cryptosystem over quadratic fields. In *electronic proceedings of 6th Symposium on Algebra and Computation AC2005*, November 2005. <ftp://tnt.math.metro-u.ac.jp/pub/ac05/Nishimoto/nishimoto.pdf>.
- [28] T. Nishino. How to design efficient quantum algorithms. *Transaction of Information Processing Society Japan*, 43(SIG 7 (TOM 6)):1–9, September 2002.
- [29] T. Nishino, H. Shibata, K. Atsumi, and T. Shima. Solving function problems and np-complete problems by nmr quantum computation. In *Technical Report of IEICE COMP*, volume 98-71, pages 65–72. IEICE, December 1998.
- [30] The Open Group. 64-bit programming models. http://www.opengroup.org/public/tech/aspen/lp64_wp.htm.
- [31] The Open Group. *Go Solo 2 — The Authorized Guide to Version 2 of the Single UNIX Specification*. Prentice Hall, Englewood Cliffs, N.J., 1997.
- [32] OpenXM, a project to integrate mathematical software systems. <http://www.openxm.org/>, 1998–2005.
- [33] The PARI group. PARI/GP Development Headquarter. <http://pari.math.u-bordeaux.fr/>.
- [34] M. Pohst. *Computational Algebraic Number Theory*. DMV Seminar 21. Birkhäuser Verlag, Berlin, 1993.
- [35] M. Pohst and H. Zassenhaus. *Algorithmic Algebraic Number Theory*. Cambridge University Press, revised edition, 1997.
- [36] H. G. Rice. Recursive real numbers. *Proc. American Math. Soc.*, 5:784–791, October 1954.
- [37] G. van Rossum. Foreword. In *Programming Python*. O’Reilly, 1st edition, May 1996.
- [38] P. W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of 35th Annual IEEE Symposium on Foundation of Computer Science*, pages 116–123, 1994.

- [39] The SIMATH Group. SIMATH-manual. http://simath.info/html_manual/index.html.
- [40] M. Sugihara and K. Murota. *Theory of Numerical Computation Methods*. Iwanamishoten, November 1994. (Japanese).
- [41] A. C. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 42:230–265, November 1936.
- [42] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang. Experimental realization of shor’s quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(20/27 December):883–887, December 2001.