석 사 학 위 논 문
Master's Thesis

피리어드의 계산

Computing Periods

2018

조 준 희 (趙駿熙 Cho, Junhee)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

피리어드의 계산

2018

조 준 희

한 국 과 학 기 술 원

전산학부

피리어드의 계산

조 준 희

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2018년 6월 12일

심사위원장	Martin Ziegler	(인)
심 사 위 원	Otfried Cheong	(인)
심 사 위 원	Paul Jung	(인)

Computing Periods

Junhee Cho

Advisor: Martin Ziegler

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
June 12, 2018

Approved by

Martin Ziegler
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS 20164450

조준희. 피리어드의 계산. 전산학부 . 2018년. 21+iii 쪽. 지도교수: 마틴 지글러. (영문 논문)
Junhee Cho. Computing Periods. School of Computing . 2018. 21+iii pages. Advisor: Martin Ziegler. (Text in English)

초 록

컴퓨터 대수 시스템(CAS)은 대수적인 수($\overline{\mathbb{Q}}$)를 일반 대중들이 쉽게 사용할 수 있게 만들었다. 하지만 아직 셀 수 없을 만큼의 초월수($\mathbb{C} \setminus \overline{\mathbb{Q}}$)가 존재한다. 콘체비치(Kontsevich)와 자기어(Zagier)는 이 틈을 메꾸기 위해 피리어드를 제안했다. 피리어드는 두 준대수적 집합의 부피의 차이다. 피리어드는 유한한 표현 즉 다항식의 계수들을 가지고 있고, 모든 대수적인 수와 일부 초월수를 포함하고 있기 때문에 대수적 모델 이론에서 많은 관심을 받고 있다. 최근 연구에서 피리어드의 최악의 경우에 대한 복잡도가 제고직 계층(Grzegorczyk Hierarchy)에서 낮은 단계에 있다는 것이 밝혀졌다. 한편 흔히 쓰이는 부동소수점 연산은 반올림 오차를 야기하고, 시간이 지남에 따라 그 오차가 누적되어 신뢰할 수 있는 연산을 제공하지 못 한다. 구간 연산은 하나의 부동소수가 아닌, 실제값이 포함된 구간($[a;b]$)을 다루고 모든 연산의 오차를 추적한다. 하지만 그 오차 범위는 점점 커져서 쓸모가 없어지기도 한다. 반대로 정확한 실수 계산은 타입-2 기계의 무한한 연산을 임의의 오차 이내의 유한한 연산으로 근사한다. 다시 말해서 출력 정확도(n)가 주어졌을 때, 2^{-n} 오차 범위 안의 다이애딕 근사값($a/2^n$)을 구하는 것이다. 이러한 배경에서 이 논문은 피리어드를 계산하는 일반적인 문제를 정확한 실수 계산의 관점에서 명세한다. 그리고 이 논문은 피리어드를 2^{-n} 오차 범위 안으로 계산하는 알고리즘을 제시하고, 분석하고, 평가한다. 또한 우리는 이 알고리즘을 우리의 선행 연구에서 제시한 알고리즘들과 비교하여 전통적인 이산 계산과 정확한 실수 계산의 장단점을 비교한다. 더 나아가 우리는 피리어드를 계산하는 문제가 NP-hard에 속함을 보인다.

핵 심 낱 말 정확한 실수 계산, 신뢰 가능한 수치해석, 계산 대수 기하학, 랜덤 알고리즘, 계산 복잡도 이론

Abstract

Computer algebra systems (CAS) brought algebraic numbers $\overline{\mathbb{Q}}$ accessible to end-users. Still, there are uncountable *transcendentals* $\mathbb{C} \setminus \mathbb{Q}$. Kontsevich and Zagier introduced *periods* to fill in this gap: a period is the difference between the volumes of two algebraic sets. Periods are receiving increasing interest in algebraic model theory as they have finite descriptions, i.e., the polynomials' coefficients, and include all algebraic reals as well as some transcendentals. Recent research has located their worst-case complexity in low levels of the Grzegorczyk Hierarchy. Meanwhile, regular floating-point arithmetic incurs rounding errors that accumulate over time and hamper reliable computations. Interval arithmetic deals with an interval $[a; b]$ where the actual value reside instead of a single floating-point approximation to keep track of the error bounds. The error bound, however, may blow up beyond use. On the other hand, *Exact Real Computation* is an approximation of infinite computations of Type-2 machines by finite computations with arbitrary precision, i.e., given an output precision n, to produce a dyadic approximation $a/2^n$. In this background, this thesis specify a general problem of computing periods in sense of Exact Real Computation; then, introduces, analyzes, and evaluates a rigorous algorithms for rigorously computing periods up to error 2^{-n}. We discuss pros and cons of traditional discrete computation and Exact Real Computation with our previous algorithms. Furthermore, we show the general problem of computing periods up to error 2^{-n} is NP-hard.

Keywords Exact Real Computation, Reliable Numerics, Computational Algebraic Geometry, Randomized Algorithms, Computational Complexity Theory

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Humans have been using numbers since the prehistory. Perhaps, they may have started from counting with fingers or objects: one, two, three, four, five, ... When they ran out of fingers or objects, they must have looked for an advanced method. One of the advanced methods is a notched bone, which dates back to Aurignacian era (35,000 to 20,000 BCE) when Cro-Magnons lived [Ifr00, p. 62]. In modern mathematics, these numbers are called *natural numbers* $\mathbb{N}$. The concept of zero has been introduced in the fifth century CE [Ifr00, p. 416]. Many mathematicians prefer including 0 in natural numbers.

$$\mathbb{N} = \{0, 1, 2, 3, \cdots\} \tag{1.1}$$

The Neolithic Revolution, the first agriculture revolution, made food surpluses resulting the advent of commerce. While adding and subtracting numbers in commerce, people must have needed negative numbers, filling *integers* $\mathbb{Z}$. When they distribute resources uniformly, they must have needed fractions, which Pythagoras referred *rational* $\mathbb{Q}$. Legend says Pythagoras murdered Hippasus for discovering $\sqrt{2}$, a *irrational number.*

$$\mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\} \tag{1.2}$$

$$\mathbb{Q} = \mathbb{Z} \times (\mathbb{Z} \setminus \{0\}) \tag{1.3}$$

Filling in the gap between rational numbers with the limits of Cauchy sequences in $\mathbb{Q}$ is *real numbers* $\mathbb{R}$. Cantor proved that $\mathbb{R}$ is uncountable [Can74]. One of the greatest achievements in modern algebra is *algebraic numbers* $\overline{\mathbb{Q}}$, roots of a non-zero polynomial in one variable with rational coefficients in $\mathbb{Q}[x]$, and *complex numbers* $\mathbb{C} = \mathbb{R}(i)$, the field generated by $i = \sqrt{-1}$ over $\mathbb{R}$.

Thanks to the solid foundation in modern algebra, computer science, and computer engineering, *computer algebra systems (CAS)* brought algebraic numbers accessible to end-users. Still, there are uncountable *transcendentals* $\mathbb{C} \setminus \overline{\mathbb{Q}}$. Kontsevich and Zagier [KZ01] introduced *periods* $\mathbb{P}$, the differences between the volumes of two semi-algebraic sets, to fill in this gap; see Definition 1. In summary, the relation of the set $\mathbb{N}$, the group $\mathbb{Z}$, the rings $\mathbb{R} \cap \mathbb{P}$ and $\mathbb{P}$, and the fields $\mathbb{Q}, \mathbb{R} \cap \overline{\mathbb{Q}}, \overline{\mathbb{Q}}, \mathbb{R}$ and $\mathbb{C}$ is in Figure 1.1:

$$\begin{array}{ccccccccccc}
\mathbb{N} & \subset & \mathbb{Z} & \subset & \mathbb{Q} & \subset & \mathbb{R}\cap\overline{\mathbb{Q}} & \subset & \mathbb{R}\cap\mathbb{P} & \subset & \mathbb{R} \\
 & & & & & & \cap & & \cap & & \cap \\
 & & & & & & \overline{\mathbb{Q}} & \subset & \mathbb{P} & \subset & \mathbb{C}
\end{array}$$

Figure 1.1: Relation of basic sets of numbers: $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R} \cap \overline{\mathbb{Q}}, \mathbb{R}, \overline{\mathbb{Q}}$, and $\mathbb{C}$

This thesis extends our previous publication "Computing Periods. . . " in WALCOM 2018, based on ideas in the same title presented at CCA 2017, which is a joint work with Sewon Park and Martin Ziegler [CPZ18]. This work was supported by the National Research Foundation of Korea (grant NRF-2017R1E1A1A03071032) and the International Research & Development Program of the Korean Ministry of Science and ICT (grant NRF-2016K1A3A7A03950702). We thank the thesis committee members for feedback!

Definition 1 (periods)**.** *A* period *is a complex number whose real and imaginary parts are values of absolutely convergent integrals of rational functions with rational coefficients, over Euclidean domains in* $\mathbb{R}^d$ *given by polynomial inequalities with rational coefficients [KZ01]:*

$$\int_{\Delta} \frac{p(x_1,\ldots,x_d)}{q(x_1,\ldots,x_d)}\, dx_1 \cdots dx_d \tag{1.4}$$

where

$$\Delta = \bigcup_{s=1}^{S} \bigcap_{t=1}^{T_s} \{\vec{x} \in \mathbb{R}^d \mid p_{s,t}(\vec{x}) > 0\} \quad \text{and} \quad p, q, p_{s,t} \in \mathbb{Q}[x_1, \cdots, x_d] \tag{1.5}$$

In Equation (1.5), it is sufficient to restrict the polynomial inequalities to strict inequalities because of Fact 2. Fact 3 strengthens Fact 2 and proceeds by induction on d: for any fixed $(x_1, \ldots x_{d-1}) \in \prod_{i=1}^{d-1} S_i$, $p(x_1, \ldots x_{d-1}, X_d)$ is a univariate polynomial of degree $\le k$.

Fact 2. *The set* $\{\vec{x} \mid p(\vec{x}) = 0\}$ *of roots of a non-zero polynomial* p *has measure zero.*

Fact 3. *Let* $S_1, \ldots, S_d \subseteq \mathbb{R}$ *be arbitrary subsets of cardinality* $|S_i| > k$ *and* $p \in \mathbb{R}[X_1, \ldots X_d]$ *non-zero of maximal degree* $\le k$*. Then there exists some* $\vec{x} \in \prod_{i=1}^{d} S_i$ *with* $p(\vec{x}) \ne 0$*.*

In fact, every period can be expressed as difference of two semi-algebraic volumes: for co-prime $p, q \in \mathbb{Q}[x_1, \cdots, x_d]$, Equation (1.4) translates to

$$\int_{\Delta_{p,q,+}} 1\, d\vec{x}\, dy \;-\; \int_{\Delta_{p,q,-}} 1\, d\vec{x}\, dy \;=\; \operatorname{vol}(\Delta_{p,q,+}) \;-\; \operatorname{vol}(\Delta_{p,q,-}) \;, \tag{1.6}$$

where

$$\Delta_{p,q,+} := \{(\vec{x}, y) \in \Delta \times \mathbb{R} \mid 0 \le y \wedge y \cdot q(\vec{x}) \le p(\vec{x})\} \tag{1.7}$$

and

$$\Delta_{p,q,-} := \{(\vec{x}, y) \in \Delta \times \mathbb{R} \mid 0 \ge y \wedge y \cdot q(\vec{x}) \ge p(\vec{x})\} \tag{1.8}$$

Note that $\Delta_{p,q,+}, \Delta_{p,q,-} \subseteq \mathbb{R}^{d+1}$ may be unbounded even when Δ was bounded. However by means of Singularity Resolution one can, without loss of generality, restrict to compact domains [VS17, Theorem 1.1]. This shows that sum and (Cartesian) product of periods are again periods.

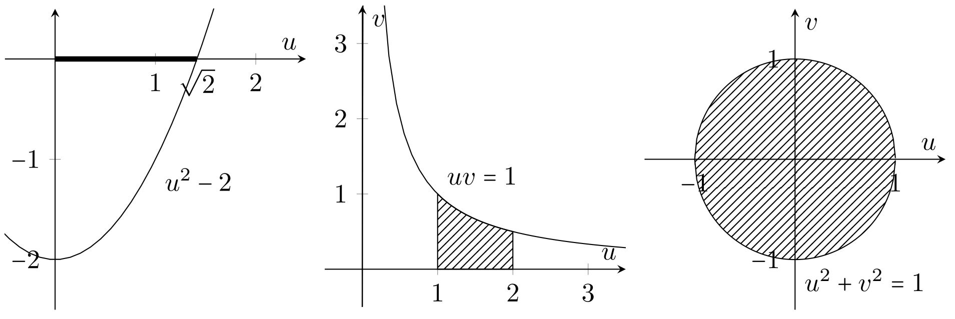
Furthermore, by appropriate rational scaling and shifting, it suffices to restrict the domain bounded by the unit cube $[0;1)^d$. Given a domain, we can fit the domain inside the unit cube by scaling the domain by a_j along each x_j axis; then, multiplying $a_1^{-1} \cdots a_d^{-1}$ to the volume of the restricted domain.

Periods are receiving increasing interest in algebraic model theory as they have finite descriptions, i.e., the polynomials' coefficients, and include all algebraic reals as well as some transcendentals. An algebraic real α has a minimal polynomial m_α, a unique monic polynomial of least degree in $\mathbb{Q}[x]$ of which α is a root. With appropriate rationals $a, b, a', b' \in \mathbb{Q}$, the following combination of polynomial inequality gives a line segment of length $|\alpha|$:

$$(m_\alpha(x) > 0 \wedge a \le x \le b) \vee (m_\alpha(x) < 0 \wedge a' \le x \le b') \tag{1.9}$$

Thus, all algebraic reals are periods with an appropriate integrand 1 or -1. Examples of periods are in Equation (1.10) and their geometric interpretations in Figure 1.2: an algebraic $\sqrt{2}$, a transcendental $\ln(x)$ for algebraic x, and a transcendental π.

$$\sqrt{2} = \int_{0 \le t \wedge 2t^2 \le 1} t\, dt, \quad \ln(x) = \int_{1}^{x} 1/t\, dt = \int_{t \le x \wedge s \cdot t \le 1} 1\, ds\, dt, \quad \pi = \int_{x^2+y^2 \le 1} 1\, dx\, dy \tag{1.10}$$



(a) $\sqrt{2}$ (b) $\ln(2)$ (c) π

Figure 1.2: Geometric interpretation of periods

Many interesting open questions evolve around periods; for instance, whether there exists an algorithm that decides whether they are equal or not given two representations of periods [KZ01, Problem 1]; asking for an "explicit" example of a real number that is not a period [KZ01, Problem 3]. Inner-mathematical candidates are $1/\pi$ and $e = \sum_n 1/n!$, but proving so seems infeasible with the current methods. The families of semi-algebraic domains and of periods are countable, non-periods are abundant.

In fact, every period is computable in the sense of Recursive Analysis [Tur37, Wei00]; therefore any incomputable reals such as the Halting Problem encoded in binary or random reals [BDC01] like Chaitin's Ω cannot be periods. More precisely each period is of *lower elementary* complexity in Grzegorczyk's Hierarchy [Yos08, TZ10, SWG12]. Note that such improved upper complexity bounds give rise to more candidates of non-periods.

Problem 4. *Characterize the computational complexity of periods!*

Moreover efficient algorithms for computing, i.e., producing guaranteed high-precision approximations to periods enable Experimental Mathematics [KZ01, Problem 2]; cmp. [Bai17].

We present a more concise algorithm based on our previous algorithms [CPZ18]. We prove it correct; describe its implementation in discrete computation unlike our previous algorithms in the convenient Exact Real Computation paradigm; estimate their cost in the unit cost model; and empirically analyze and compare its behaviour to our previous algorithms in terms of the output precision and the degree of the polynomial involved.

Section 1.1 recalls central notions, properties, and practice of real computation; Section 1.2 puts things in perspective to related work. Section 1.3 specifies the problem this thesis cope with. Section 1.4 compares this thesis with our previous publication. Our algorithm is presented, and proven correct, in Chapter 2. In Chapter 3, we introduce our implementation, performance measurements, and their evaluation and interpretation. In Chapter 4, we show the problem of computing periods specified in Section 1.3 is NP-hard. Chapter 5 expands on future work.

1.1 Real Computation

Regular floating-point arithmetic incurs rounding errors that accumulate over time and hamper reliable computations. One example is m iterations of logistic map $f_r^m(x) : [0;1] \to [0;1]$ defined by

$$f_r^m(x) = \begin{cases} rx(1-x) & \text{if } m = 1 \\ f_r^1(f_r^{m-1}(x)) & \text{otherwise} \end{cases} \tag{1.11}$$

m	Floating-point arithmetic						Exact Real Computation	
	32-bit		64-bit		128-bit			
	$f^m_{3.75}(0.5)$	Time	$f^m_{3.75}(0.5)$	Time	$f^m_{3.75}(0.5)$	Time	$f^m_{3.75}(0.5)$	Time
30	0.715239	8	0.718096	0	0.718096	0	0.718097	77
40	0.847816	0	0.416349	0	0.416349	0	0.416349	13
85	0.642700	0	0.550527	0	0.816883	0	0.816938	28
100	0.936589	0	0.936749	0	0.816798	0	0.888294	32
200	0.578099	1	0.220493	1	0.513209	16	0.823557	77
500	0.833184	2	0.786115	2	0.829312	2	0.276754	297
1,000	0.914432	4	0.340803	4	0.662663	5	0.791747	714
10,000	0.222593	68	0.276689	39	0.223618	47	0.824205	130,653
100,000	0.860895	516	0.239181	434	0.521347	511	0.666947	32,783,502
500,000	0.728425	2,155	0.274556	2,155	0.925446	2,681	0.451203	1,245,317,065

Table 1.1: Comparison of computing m iterations of logistic map $f^m_r(x)$ defined in Equation (1.11) with $r = 3.75$ at $x = 0.5$, up to six decimal places in floating-point arithmetic and Exact Real Computation (time in μs)

Table 1.1 shows the result of computing $f^m_r(x)$ where $r = 15/4 = 3.75$ and $x = 1/2 = 0.5$ in floating-point arithmetic. It illustrates rounding errors accumulating over iterations and any fixed-precision floating-point arithmetic is unreliable. Note that higher fixed-precision floating-point arithmetic does not guarantee more precise results.

Interval arithmetic deals with an interval $[a; b]$ where the actual value resides instead of a single floating-point approximation to keep track of the error bounds. The error bound, however, may blow up beyond use. In case of the aforementioned iterations of logistic map, interval arithmetic results an interval $[0; 1]$ after a few iterations, which is useless, due to the chaotic behaviour of logistic map. Moreover, a comparison of two numbers may be meaningless when two intervals representing the numbers overlap.

Exact Real Computation is an approximation of infinite computations of Type-2 machines by finite computations with arbitrary precision [Wei00, p. 16]. A Type-2 machine M is a Turing machine with k one-way input tapes $Y_1, \cdots, Y_k$ and a one-way output tape Y_0 together with a type specification $(Y_1, \cdots, Y_k, Y_0)$ with $Y_i \in \{\Sigma^*, \Sigma^\omega\}$, giving the type for each input tape and the output tape [Wei00, Definition 2.1.1]. Computational complexity of Type-2 machine naturally involves the output length n. Time complexity of Type-2 machine M on input y is the number of steps which the machine M on input y needs until it has printed the n-th output symbol [Wei00, Definition 7.1.1]. In other words, we define computable real numbers, computable sequences of real numbers, and computable real-valued functions of real variables as follows:

Definition 5 (computable real numbers)**.** *An algorithm A computes a real number r if it outputs a dyadic approximation $a/2^n$ of r up to error 2^{-n}:*

$$\left| r - \frac{a_n}{2^n} \right| < 2^{-n} \tag{1.12}$$

given any output precision $n \in \mathbb{N}$ in unary word. A real number r is computable if there exist an algorithm A which computes r.

There are only countable computable real numbers because there are only countable algorithms by the Cantor's diagonal argument. An example of incomputable real number is obtained by embedding the halting problem H in a real number. The halting problem H is a set of pairs $\langle M, x\rangle$ of a Turing machine M and an instance x where the Turing machine M on input x halts [Tur37]. Let h be a real number such that

$$h := \sum_{\langle n\rangle \in H} 4^{-n} \tag{1.13}$$

Suppose h is computable. Then, we decide the halting problem by computing $n \in \mathbb{N}$ such that $\langle n\rangle = \langle M, x\rangle$, computing h up to error 2^{-2n-2}, then testing the $2n$-th binary place is 1 given a Turing machine M and an instance x. This is a contradiction because the halting problem is undecidable [Tur37].

Definition 6 (computable sequences of real numbers)**.** *An algorithm A computes a sequence of real numbers (r_j) if it outputs a dyadic approximation $a/2^n$ of r_j up to error 2^{-n} given any output precision $n \in \mathbb{N}$ in unary word and any index $j \in \mathbb{N}$. A sequence of real numbers (r_j) is computable if there exist an algorithm A which computes (r_j).*

Definition 7 (computable real-valued functions of real variables)**.** *An algorithm A computes a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ if it outputs a dyadic approximation $a/2^n$ of $f(x)$ up to error 2^{-n} given any output precision $n \in \mathbb{N}$ in unary word and oracles $X_1, \cdots, X_d$ which computes $x_1, \cdots, x_d$, respectively, for any $x_1, \cdots, x_d \in \mathbb{R}$. A real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ is computable if there exist an algorithm A which computes f.*

For example, addition $x + y$ is computable: given n, compute dyadic approximations $a/2^{n+1}$ and $b/2^{n+1}$ by oracles X and Y of x and y, respectively, up to error 2^{-n-1}; then, output the sum of approximations $(a+b)/2^n$. Also, multiplication $x \cdot y$ is computable: given n, compute dyadic approximations $a/2^{2n}$ and $b/2^{2n}$ by oracles X and Y of x and y, respectively, up to error 2^{-2n}; then, output the product of approximations $ab/2^n$. Note that the domain of a computable real-valued function is not restricted to the computable real numbers.

It is well-known from Recursive Analysis that equality of real numbers is incomputable. In other words, the sign of a real number, $\mathrm{sign} : \mathbb{R} \to \{-1, 0, 1\}$, is incomputable [Wei00, Exercise 4.2.9]. It is enough to show that there exists a computable sequence (r_j) where the sequence $(\mathrm{sign}(r_j))$ is incomputable. Let $\langle j\rangle = \langle M, x\rangle$ and let $T \subseteq \mathbb{N}$ such that $n \in T$ if the Turing machine M on input x halts in n steps. Now, we define the sequence (r_j) as follows:

$$r_j = \sum_{n \in T} 4^{-n} \tag{1.14}$$

The sequence (r_j) is computable by simulating the Turing machine M on input x for n steps and computing the partial sum:

$$\frac{a}{2^n} = \sum_{m \in T, 2m \le n} 4^{-m} \tag{1.15}$$

The sequence $(\mathrm{sign}(r_j))$ is incomputable because $r_j = 0$ if and only if the Turing machine M on input x does not halt. Given an oracle of the sequence $(\mathrm{sign}(r_j))$, therefore, we can solve the halting problem H by finding the corresponding j; querying the oracle about the sign of r_j; then, deciding whether the sign is non-zero: $H \le_{\mathrm{T}} \{j \mid r_j \ne 0\}$.

When $r \ne 0$, a naïve algorithm computes the sign of r: compute a dyadic approximation $a/2^n$ for some n; report the sign of a if $|a/2^n| > 2^{-n}$; otherwise, increment n and repeat. Indeed, this algorithm freezes when $r = 0$. Given an oracle of H, we can compute the sign of a real number r by querying the oracle about this algorithm on input r. Therefore, equality of real numbers is Turing equivalent to the complement of the halting problem: $H \equiv_{\mathrm{T}} \{j \mid r_j \neq 0\}$.

iRRAM is a C++ exact real computation library [Mül01, MZ14], providing real numbers as abstract data type with exact operations and partial comparison: A test "$r > 0$" freezes in case $r = 0$. Table 1.1 shows the result of computing $f_r^m(x)$ in Equation 1.11 where $r = 15/4 = 3.75$ and $x = 1/2 = 0.5$ in Exact Real Computation with iRRAM. Exact Real Computation requires more time than floating-point arithmetic to achieve a correct approximation because iRRAM employs flexible-precision floating-point; thus, the bit-cost of a floating-point operation is not constant.

To write total programs in spite of comparison being partial, a *parallel disjunction* is provided: calling the non-deterministic or *multivalued* 'function' choose$(x_1 > 0, \ldots, x_k > 0)$ returns *some* integer j such that $x_j > 0$ holds, provided such j exists.

Subject to this modified semantics of tests, *Exact Real Computation* allows to conveniently process real arguments and intermediate results as entities, naïvely without precision considerations. On the other hand, the output or return value of a function in iRRAM merely needs to be provided in approximation up to absolute error 2^p for any parameter $p \in \mathbb{Z}$ passed. The following algorithm demonstrates this paradigm with the *trisection* method for finding the promised unique and simple root of a given continuous function $f : [0; 1] \to [-1; 1]$ while avoiding the sign test "$0 < f(a) \cdot f(b')$" to fail in case b' already happens to be a root:

Algorithm REAL Trisection(INTEGER p, REAL → REAL f)

```
REAL $a := 0$; REAL $b := 1$
while choose$\big( b - a > 2^{p-1}$ , $2^p > b - a \big) == 1$ do
    REAL $a' := \frac{2}{3}a + \frac{1}{3}b$;    REAL $b' := \frac{1}{3}a + \frac{2}{3}b$
    if choose ( $0 > f(a) \cdot f(b')$ , $0 > f(a') \cdot f(b)$) == 1 then
        $b := b'$
    else
        $a := a'$
    end if
end while
return $a$
```

Figure 1.3: Pseudocode of root finding algorithm with trisection

Remark 8. *A caveat, in* Exact Real Computation *the bit — as opposed to unit — cost of each operation may well depend on the value (and internal precision) of the data being processed. For instance a sign test "$x > 0$" will take time between linear and quadratic in $n \approx \log_2(1/|x|)$: to obtain the dyadic approximation $a_n/2^n$ of x up to error 2^{-n} and verify it to be strictly larger than 2^{-n}. Similarly, a parallel test choose$(x_1 > 0, \ldots, x_k > 0)$ will take time roughly $k \cdot \min_{j<k} \log(1/|x_j|)$.*

1.2 Periods and their Computational Complexity

It has been shown that periods are of *lower elementary* complexity [TZ10, SWG12]. The present section recalls this and related notions with their connections to resource-bounded complexity.

In Grzegorczyk's Hierarchy, *Lower Elementary* means the smallest class of total multivariate functions $f : \mathbb{N}^d \to \mathbb{N} = \{0, 1, 2, \cdots\}$ containing the constants, projections, successor, modified difference $x \dot{-} y = \max\{x - y, 0\}$, and is closed under composition and bounded summation $f(\vec{x}, y) = \sum_{z=0}^{y} g(\vec{x}, z)$.

We write $\mathcal{M}^2 = \mathcal{E}^2$ for the smallest class of such f containing the constants, projections, successor, modified difference, binary multiplication, and closed under both composition and bounded search $\mu(f)(\vec{x}, y) = \min\{z \leq y : f(\bar{z}, z) = 0\}$.

A real number r is *lower elementary* if there exist lower elementary integer functions $f, g, h : \mathbb{N} \to \mathbb{N}$ with $\left|r - \frac{f(N)-g(N)}{h(N)}\right| < 1/N$ for all $N > 0$; similarly for a real number in $\mathcal{M}^2$.

A real number r is computable in *time* $t(n)$ and *space* $s(n)$ if a Turing machine can, given $n \in \mathbb{N}$ and within these resource bounds, produce some $a_n \in \mathbb{Z}$ with $|r - a_n/2^n| \leq 2^{-n}$ [Ko91].

Fact 9. *a) All functions from $\mathcal{M}^2$ are lower elementary; and the latter functions grow at most polynomially in the value of the arguments. In terms of the binary input length and with respect to bit-cost, lower elementary functions are computable using a linear amount of memory for intermediate calculations and output, that is, they belong to the complexity class* $\mathsf{FSPACE}(n)$.

b) $\mathsf{FSPACE}(n)$ *is closed under bounded summation and therefore coincides with the class of lower elementary functions. The 0/1-valued functions (that is, decision problems) in $\mathcal{M}^2$ exhaust the class* $\mathsf{SPACE}(n)$ [Rit63, §4]*; cmp.* [Kut87].

c) π and $e = \sum_n 1/n!$ and Liouville's transcendental number $L = \sum_n 10^{-n!}$ and the Euler-Mascheroni Constant $\gamma = \lim_n \big(-\ln(n) + \sum_{k=1}^{n} 1/k\big)$ are all lower elementary [Sko08, §3].

d) The set of lower elementary real numbers constitutes a real closed field: Binary sum and product and reciprocal of lower elementary real numbers, as well as any real root of a non-zero polynomial with lower elementary coefficients, are again lower elementary [SWG12, Theorem 2].

e) Arctan, natural logarithm and exponential as well as Γ and ζ function map lower elementary reals to lower elementary reals [TZ10, §9].

f) Natural logarithm maps periods to periods; $\zeta(s)$ is a period for every integer $s \geq 2$ [KZ01, §1.1].

g) Periods are lower elementary [TZ10, Corollary 6.4].

h) Given a Boolean expression $\varphi(x_1, \cdots, x_m)$ as well as the degrees and coefficients of the polynomials p_j defining its constituents S_{p_j}, deciding whether the semi-algebraic set $\varphi\big(S_{p_1}, \cdots, S_{p_m}\big)$ is non-empty/of given dimension [Koi99] *is complete for the complexity class* $\mathsf{NP}^0_{\mathbb{R}} \supseteq \mathsf{NP}$.

Item a) follows by structural induction. Together with b) it relates resource-oriented to Grzegorczyk's structural Complexity Theory. Note that the hardness Result h) does not seem to entail a lower bound on the problem of approximating a fixed volume; in fact many of the usual reductions among real algebraic decision problems [Mee06] fail under volume considerations. Common efficient and practical algorithms tailored for approximating L, e, γ, or the period π do so up to absolute error $1/N := 2^{-n}$ within time polynomial in the binary precision parameter $n = \log_2 N$ [Kan03]; whereas the best runtime bound known for $\mathsf{SPACE}(n)$ is only exponential [Pap94, Problem 7.4.7]. On the other hand exponential-time algorithms may well be practical [FG06, KF10].

1.3 Problem Statement

In this background, we specify the problem of computing periods as follows:

Definition 10 (computing periods)**.** *Given polynomial inequalities $p_{s,t} > 0$ with rational coefficients in $\mathbb{Q}[x_1, \cdots, x_d]$ in a disjunctive normal form and an output precision $n \in \mathbb{N}$ in a unary word, produce a dyadic approximation $a/2^n$ of the volume of domain Δ from Equation (1.5) satisfying the polynomial inequalities bounded by the unit cube up to error 2^{-n}:*

$$\left|\mathrm{vol}(\Delta) - \frac{a}{2^n}\right| \le 2^{-n} \quad \textit{where} \quad \Delta = \bigcup_{s=1}^{S} \bigcap_{t=1}^{T_s} \{\vec{x} \in \mathbb{R}^d \mid p_{s,t}(\vec{x}) > 0\} \tag{1.16}$$

1.4 Comparison of this Thesis and our Previous Publication

In this section, we compare this thesis and our previous publication "Computing Periods..." in WALCOM 2018 [CPZ18]. In the previous publication, we considered the sum of the squares of polynomials in the analyses:

$$p = \sum_{j=1}^{M} p_j^2 \tag{1.17}$$

The roots of p, however, captures the simultaneous roots of $p_1, \cdots, p_M$ only, not the union of the roots of each polynomial. Therefore, the analyses are wrong because non-simultaneous roots of a polynomial can cause an error. With the product of polynomials $p_1, \cdots, p_M$, the analyses are correct:

$$p = \prod_{j=1}^{M} p_j \tag{1.18}$$

Because the sign of a real number is incomputable, we introduced three algorithms to avoid a root when computing the sign of the function value of the given polynomials at a point in each sub-cube: randomized, deterministic, and *transcendental* algorithms. The randomized algorithm evaluates the sign at a real random point based on Fact 2. The deterministic algorithm evaluates the signs at more than one points in parallel based on Fact 3. Finally, the transcendental algorithm evaluates the sign at an algebraically independent point based on Fact 11. Note that $\vec{x} \in \mathbb{R}^d$ is algebraically independent if and only if it is not a root of any polynomials with rational coefficients in $\mathbb{Q}[x_1, \cdots, x_d]$. Fact 11 is the *Lindemann-Weierstrass Theorem*, applied to linear independence of prime square roots over rationals.

Fact 11. *For $p_1, \ldots, p_d$ pairwise distinct primes, $\vec{x} := \left(e^{\sqrt{2}}, e^{\sqrt{3}}, \ldots, e^{\sqrt{p_d}}\right)$ is algebraically independent: $q(\vec{x}) \neq 0$ for every non-zero d-variate polynomial q with integer coefficients; and more generally $q\big(A \cdot \vec{x} + \vec{b}\big) \neq 0$ for any vector $\vec{b} \in \mathbb{A}^d$ with algebraic coefficients and invertible $d \times d$-matrix $A \in \mathrm{GL}_d(\mathbb{A})$.*

Because the image of the restriction of polynomials with rational coefficients to $\mathbb{Q}$ is subsets of $\mathbb{Q}$, we choose a rational point in each sub-cube, namely the center of sub-cube, and evaluate the signs of the polynomials at the rational points in this thesis. This results a more concise algorithm than the previous three algorithms.

Moreover, the running time of computing the sign of function value at a point depends on the function value. Thus, it seems infeasible to analyze the previous three algorithms in terms of bit-cost. On the other hand, bit-cost analysis of the new algorithm is feasible because the algorithm involves only discrete computations although it is left as a future work.

Chapter 2. Our Algorithm and its Analysis

In the view of the problem statement in Section 1.3, this section devises and analyzes an algorithm that approximates the volume of the set of solutions $\vec{x}$ to d-variate polynomial inequalities with rational coefficients in disjunctive normal form up to guaranteed absolute error 2^{-n}.

The common basic idea underlying all of our algorithms including our previous algorithms is to divide $[0;1]^d$ into sub-cubes

$$Q_{\vec{c},N} := \left[\tfrac{\vec{c}}{N};\tfrac{\vec{c}+\vec{1}}{N}\right) = \prod_{i=1}^{d}\left[\tfrac{c_i}{N};\tfrac{c_i+1}{N}\right) \quad \text{for all} \quad \vec{c}\in[N]^d \tag{2.1}$$

where $[N] := \{0,\ldots,N-1\}$ and $\vec{1} := (1,\ldots,1)$; then determine the signs of the polynomials $p_{s,t}$ at the center points of the sub-cubes $Q_{\vec{c},N}$;

$$\vec{x}_{\vec{c},N} := \frac{2\vec{c}+\vec{1}}{2N} \in Q_{\vec{c},N} \tag{2.2}$$

and count those, where the constraints $p_{s,t}(\vec{x}_{\vec{c},N}) > 0$ are met, with uniform weight $\operatorname{vol}(Q_{\vec{c},N}) = N^{-d}$.

However, when a polynomial's sign may vary within a sub-cube $Q_{\vec{c},N}$, the above approach can incur an error. Here are two observations: only sub-cubes containing a root produce an error; and the total error converges to zero as N increases. Therefore, with appropriate N, the above approach results an approximation up to guaranteed absolute error 2^{-n}. We show an appropriate N and prove its correctness in Section 2.1 and we summarize our complete algorithm in pseudocode in Section 2.2.

2.1 Recap on Real Algebraic Geometry

Let k be the maximum degree of the product p of polynomials $p_{s,t}$. Then, the union of the roots of polynomials $p_{s,t}$ is exactly the roots of p.

$$p = \prod_{s=1}^{S}\prod_{t=1}^{T_s} p_{s,t} \quad \text{and} \quad k = \operatorname{maxdeg}(p) \tag{2.3}$$

1D case In the case of $d = 1$, a change of sign can occur in and affect at most k of the sub-intervals $Q_{c,N}$ because p can have at most k roots. Hence, taking $N > k\cdot 2^n$ guarantees the required error bound 2^{-n}. A multivariate polynomial on the other hand may have infinitely many roots, which however can form only a bounded number of connected components. This allows us to generalize the analysis of 1D case as Lemma 12 and 14.

2D case By Lemma 12, taking N satisfying the following guarantees the required error bound 2^{-n}:

$$\left(1+\frac{(k-1)(k-2)}{2}+2(N+1)\cdot k\right)\cdot N^{-2} < 2^{-n} \tag{2.4}$$

Lemma 12 (2D case)**.** *In case of $d = 2$ and for any fixed non-zero polynomial $p\in\mathbb{R}[x,y]$ of maximum degree k, at most $1+(k-1)\cdot(k-2)/2+2(N+1)\cdot k$ of the $N\times N$ sub-squares $Q_{\vec{c},N}$ can contain roots of p.*

Proof. By Fact 13 below, the roots of p can form at most $c \le 1+(k-1)\cdot(k-2)/2$ connected components in $\mathbb{R}^2$. Of course, such a component may extend through more than one of the sub-squares $Q_{\vec{c},N}$; however, in order to do so, it must cross one of the $N+1$ horizontal lines or one of the $N+1$ vertical lines forming the sub-division of $[0;1]^2$. More precisely for component C to extend to $M_C \in \mathbb{N}$ of the N^2 sub-squares, it must intersect at least $M_C - 1$ of the $2(N+1)^2$ segments of the $2(N+1)$ aforementioned lines; and for all c components to extend to a total of M of the N^2 sub-squares, they have to intersect these lines in at least $M - c$ points, which are distinct since connected components do not meet. However, p restricted to any of the $2(N+1)$ lines boils down to a univariate polynomial either in x or in y of degree at most k, and hence can have at most k roots on each such line: requiring $M - c \leq 2k \cdot (N+1)$. □

Fact 13 (Harnack's Curve Theorem)**.** *Let $p \in \mathbb{R}[x,y]$ denote a bivariate polynomial of maximum degree k. Then the number of connected components of $\{(x,y) \in \mathbb{R}^2 \mid p(x,y) = 0\}$ is at most $1+(k-1)\cdot(k-2)/2$ [PP16, Theorem 48.1].*

General case By Lemma 14, taking N satisfying the following guarantees the required error bound 2^{-n},

$$\left(k^d + k^{d-1} \cdot d \cdot (N+1)\right) \cdot N^{-d} < 2^{-n} \tag{2.5}$$

namely by Lemma 16:

$$N > 3 \max(k,d) \cdot 2^{n/(d-1)} \tag{2.6}$$

Lemma 14 (general case)**.** *For $d \geq 3$ and any fixed non-zero polynomial $p \in \mathbb{R}[x_1, \ldots, x_d]$ of maximum degree k, at most $k^d + k^{d-1} \cdot d \cdot (N+1)$ of the N^d sub-(hyper)cubes $Q_{\vec{c},N}$ can contain roots of p.*

Proof. Similarly to the proof of Lemma 12, the roots of p can form at most $c_d \leq k^d$ connected components according to Fact 15. For any such component C to extend to $M_C \in \mathbb{N}$ of the N^d sub-(hyper)cubes, it must intersect at least $M_C - 1$ of the $d \cdot (N+1)^d$ facets of the overall subdivision induced by the $d \cdot (N+1)$ hyperplanes; and for all c_d components to extend to a total of M of the N^d sub-cubes, they have to intersect these hyperplanes in at least $M - c_d$ different components. However, p restricted to any of the $d \cdot (N+1)$ hyperplanes boils down to a $(d-1)$-variate polynomial of maximum degree at most k, whose roots can form at most $c_{d-1} \leq k^{d-1}$ connected components according to Fact 15: $M - c_d \leq c_{d-1} \cdot d \cdot (N+1)$. □

Fact 15 (generalization of Harnack's Curve Theorem due to Milnor and Thom)**.** *If $\Delta \subseteq \mathbb{R}^d$ is the zero set of one polynomial of maximum degree k, then it has at most k^d connected components; if it is the conjunction of (any number of) such sets, then it has at most $k \cdot (2k-1)^d$ connected components [HRR90, Theorem 9].*

Lemma 16. *It holds $k^d + k^{d-1} \cdot d \cdot (N+1) \leq N^d \cdot 2^{-n}$ for all $N \geq 3k \cdot 2^{n/(d-1)}$ and $k \geq d \geq 3$.*

Proof. Apply inequality $|x|^d + |y|^d \leq (|x| + |y|)^d$ to $x^d := 2^n \cdot (k^d + d \cdot k^{d-1}) \leq 2k^d \cdot 2^{n \cdot d/(d-1)}$ and $y^d := N \cdot d \cdot k^{d-1} \cdot 2^n$, taking into account $\sqrt[d]{2} + \sqrt[d-1]{d} \leq 3$ for all $d \geq 3$. □

2.2 Our Algorithm and Pseudocode

In order to guarantee absolute error bound 2^{-n}, our algorithm uses binary search to find the least $N \in \mathbb{N}$ satisfying Equation (2.4) or (2.5) rather than apply the concise but asymptotic bound in Equation (2.6). In summary, our complete algorithm in pseudocode is in Figure 2.1. From Equation (2.6), the running time of our algorithm in unit-cost is $O(N^d) = O(k^d \cdot 2^{nd})$, exponential to n and d, but only polynomial to k.

Algorithm `BOOL` IsEnoughN(`INTEGER` n, `INTEGER` d, `INTEGER` N)

```
if $d == 1$ then
    return $N > k \cdot 2^n$
else if $d == 2$ then
    return $1 + (k-1) \cdot (k-2)/2 + 2(N+1) \cdot k < N^2 \cdot 2^{-n}$
else
    return $k^d + k^{d-1} \cdot d \cdot (N+1) < N^d \cdot 2^{-n}$
end if
```

Algorithm `RATIONAL` Period(`INTEGER` n, `INTEGER` d, `POLYNOMIAL` $p_{s,t} \dots$)

```
INTEGER $S$ := the number of conjunctions
INTEGER $T_s$ := the number of polynomial inequalities in $j$-th conjunction for each $1 \leq s \leq S$
POLYNOMIAL $p := \prod_{s=1}^{S} \prod_{t=1}^{T_s} p_{s,t}$
INTEGER $k$ := maxdeg($p$)
REAL $l := 0$; REAL $r := 1$
while $\neg$IsEnoughN($n+1$, $d$, $r$) do
    $r := 2r$
end while
while $l \leq r$ do
    INTEGER $N := (l+r)/2$
    if IsEnoughN($n+1$, $d$, $N$) then
        $r := N - 1$
    else
        $l := N + 1$
    end if
end while
INTEGER $N := l$
INTEGER $C := 0$
while INTEGER $\vec{c} \in [N]^d$ do
    RATIONAL $\vec{x} = (2\vec{c} + \vec{1})/(2N)$
    if $\exists 1 \leq s \leq S.\ \forall 1 \leq t \leq T_s.\ p_{s,t}(\vec{x}) > 0$ then
        $C := C + 1$
    end if
end while
RATIONAL $a := C/N^d$ rounded off to $n$-th binary place.
return $a$
```

Figure 2.1: Pseudocode of our algorithm

Chapter 3. Implementation and Evaluation

The aforementioned algorithm was implemented without multithreading in C++ with GMP, the GNU Multiprecision Arithmetic Library, involving only discrete computations. Meanwhile, the three algorithms introduced in [CPZ18] were implemented in *Exact Real Computation* based on the `iRRAM` C++ library [Mül01, MZ14]. The randomized and transcendental algorithms had the best performance among the three algorithms. In this chapter, we discuss the result of the new algorithm from this thesis on the same experiment in [CPZ18]. For convenience and to avoid confusion, we refer the new algorithm as *Thesis* algorithm and the randomized algorithm as *CPZ18* algorithm. Note that both Thesis and CPZ18 algorithms have the same unit-cost.

However, this analysis only counts the number of operations, that is, referring to an algebraic or unit-cost measure as opposed to the more realistic bit-cost measure taking into account aspects of internal or working precision. For the CPZ18 algorithm, the latter seem hard to estimate, though, since they depend not only on the output precision n and the polynomial degree k, but also on the coefficients of the polynomial constraints p under consideration: which in the worst-case may give rise to unbounded running times. On the other hand, for the Thesis algorithm, the bit-cost analysis is now feasible because it involves only discrete computations although it is left as a future work. For a more realistic assessment we have thus implemented, empirically evaluated and compared the practical performance of the both algorithms on the following kinds of benchmark polynomials:

The bivariate polynomials representing the transcendental periods π and $\ln(2)$; recall Equation (1.10); and for $d = 2$ and $d \geq 3$, the multivariate scaled *Wilkinson-type* Polynomials I:

$$p_{n,d} := \prod_{i=1}^{d} \prod_{j=1}^{k} \left(x_i - \tfrac{j}{k} \right) \tag{3.1}$$

deliberately placing roots at points on a grid that the deterministic algorithm will try to determine their signs in and *Wilkinson-type* Polynomials II:

$$p_{n,d} := \prod_{j=1}^{k} \left(\prod_{i=1}^{d} x_i - \tfrac{j}{k} \right) \tag{3.2}$$

3.1 Computing Environment

Their source codes and the experiment results are available for download from the urls `https://github.com/junheecho/period` and `https://github.com/junheecho/iRRAM`. We have executed and timed them on a computer with Intel® Core™ i7-6950X processor with 10 cores, 20 threads running at 3.00 GHz and 64 GB of RAM. Less than 20 processes ran at the same time so that they do not impede each other. The source code is compiled with g++ 5.4.0 on Ubuntu 16.04.3 LTS. Also, GMP 6.1.0 is used. We focus on CPU time; memory was never a problem.

3.2 Performance Results

We have measured, for each of the two algorithms and each of the above benchmark polynomials as input, the CPU time in dependence on n; for the Wilkinson-type Polynomials also on k. We have

Input	Algorithm	Regression	R^2 score
π	CPZ18	$\exp(1.31n - 9.51)$	0.99
	Thesis	$\exp(1.27n - 8.40)$	0.95
(the ratio of Thesis over CPZ18)		$\exp(0.07n - 0.14)$	0.12
$\ln(2)$	CPZ18	$\exp(1.38n - 11.74)$	1.00
	Thesis	$\exp(1.19n - 8.81)$	0.93
(the ratio of Thesis over CPZ18)		$\exp(0.00n - 0.68)$	0.01
Wilkinson-type Polynomials I	CPZ18	$\exp(1.38n - 12.32) \cdot k^{3.33}$	0.89
		$\exp(1.01k + 1.37n - 11.95)$	0.49
	Thesis	$\exp(1.32n - 11.49) \cdot k^{3.39}$	0.99
		$\exp(1.02k + 1.31n - 11.12)$	0.94
(the ratio of Thesis over CPZ18)		$\exp(0.02n - 0.11) \cdot k^{0.19}$	0.01
		$\exp(0.04k + 0.02n + 0.04)$	-0.02
Wilkinson-type Polynomials II	CPZ18	$\exp(1.25n - 10.39) \cdot k^{2.80}$	0.98
		$\exp(0.73k + 1.24n - 9.82)$	0.76
	Thesis	$\exp(1.29n - 10.70) \cdot k^{3.05}$	0.98
		$\exp(0.75k + 1.26n - 9.62)$	0.91
(the ratio of Thesis over CPZ18)		$\exp(0.05n - 0.34) \cdot k^{0.21}$	0.24
		$\exp(0.04k + 0.04n - 0.19)$	0.18

Table 3.1: Parameter Regression of CPU time

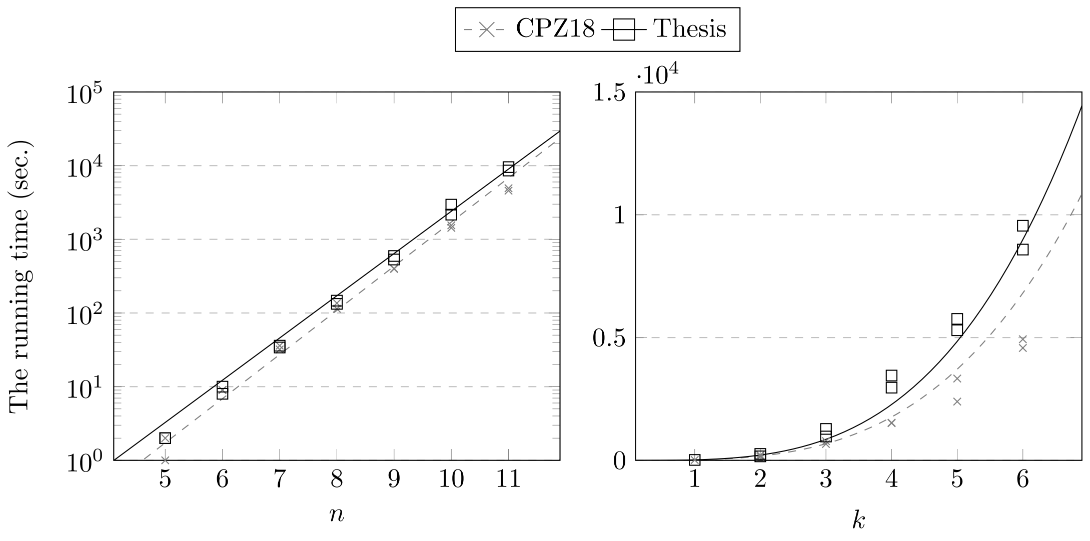
then fitted the results to the model or ansatz $\exp(n \cdot \beta - \alpha)$ — for the Wilkinson-type Polynomials to $\exp(n \cdot \beta - \alpha) \cdot k^{\gamma}$ — and plotted both. We have also fitted the result for the Wilkinson-type Polynomials to $\exp(k \cdot \gamma + n \cdot \beta - \alpha)$ but the R^2 scores show the former model is more suitable; see Table 3.1. We have also plotted and fitted the *ratios* of the algorithms' respective performances to the aforementioned models; see Figure 3.1.

3.3 Interpretation

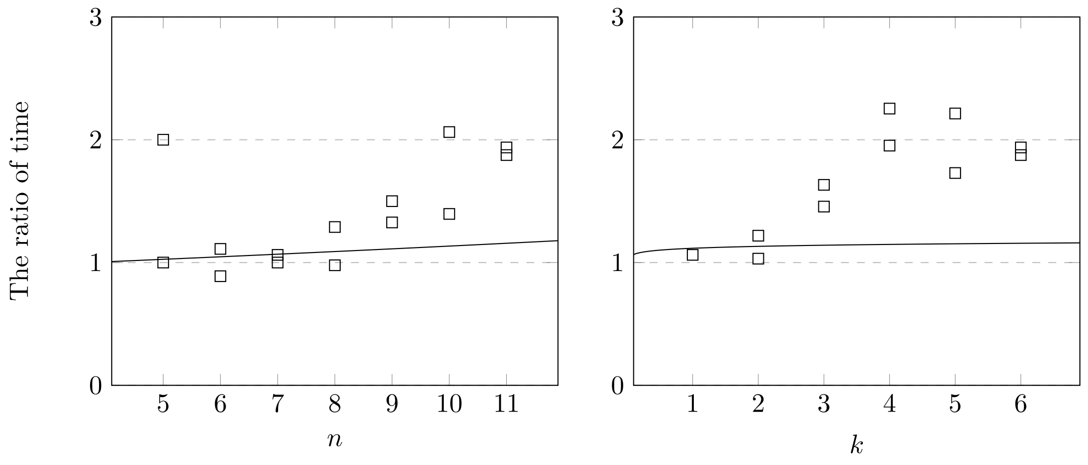
Our measurements confirm the predicted running times polynomial in k and exponential in n. The CPZ18 algorithm runs faster than the Thesis algorithm even though we did not capture that with the regression models. Therefore, we conclude their performance only vary by a constant factor. However, the CPZ18 implementation has a limitation on capability due to `iRRAM`'s nature.

The practical advantage of the CPZ18 algorithm comes from intermediate interval computations, i.e., `iRRAM` does not compute the exact value of rationals, but compute approximations by an interval with multiprecision floating points. When the precision of floating points is not sufficient, it increases the precision and reiterate the program: a bottom-up approach. Hence, when the absolute value of the function value of a polynomial is large, it is possible to decide the sign with low precision floating points. On the other hand, the Thesis algorithm should compute the exact value; thus, running slow.

`iRRAM`'s bottom-up approach together with parallel disjunction, namely `choose` function, causes a limitation on space complexity. Even though `iRRAM` takes a bottom-up approach, one of its primary goal is to hide reiterations from users. When it comes to parallel disjunction at `choose` function call,



(a) The running time and the fit, $k = 6$ (left) and $n = 11$ (right)



(b) The ratio of Thesis over CPZ18, $k = 6$ (left) and $n = 11$ (right)

Figure 3.1: CPU time (sec.) of computing periods via Wilkinson-type Polynomials I

it has to return the same value from the function call for all reiterations. Due to different timing in multithreading or randomization, `iRRAM` caches which branch it took at each `choose` call and follows the same branch afterwards. This makes the space complexity linearly depend on the time complexity.

In our case, the space complexity does not depend on the number of sub-cubes. However, `iRRAM`'s cache for `choose` calls makes the space complexity depend on the number of sub-cubes. Although we did not run out of physical memory during the experiment, the program crashed; for example, the CPZ18 implementation crashed when computing π up to error 2^{-13}. Perhaps, `iRRAM` has its own limit on the cache size. From this perspective, we suggest the following principle:

Principle 17. *Avoid real computations whenever possible.*

Chapter 4. NP-hardness

In this chapter, we show the general problem of computing periods from Definition 10 is NP-hard by a polynomial reduction from *3-satisfiability problem* (3-SAT). 3-SAT is a decision problem to decide the satisfiability of a Boolean formula in conjunctive normal form where each clause has at most 3 literals. Let x_i be a Boolean variable for $1 \leq i \leq N$. Then, a literal l is x_i or $\neg x_i$ for some $1 \leq i \leq N$, a clause C_j is a disjunction of $N_j \leq 3$ literals: $C_j = \vee_{k=1}^{N_j} l_{j,k}$ for $1 \leq j \leq M$. Formally, 3-SAT is a set of $\wedge_{j=1}^{M} C_j$ which is satisfiable. Without loss of generality, we assume each clause has exactly three literals by duplicating a literal. For a polynomial-time many-one reduction, we identify a decision problem (n,q)-PERIOD:

$$\text{(n,q)-PERIOD} := \{\Delta \mid \mathrm{vol}(\Delta) > q + 2 \cdot 2^{-n}\} \tag{4.1}$$

Given an instance of 3-SAT, we construct an instance of (n,q)-PERIOD as follows:

1. Construct a real variable $y_i \in [0; 1/6] \cup [5/6; 1]$ for each Boolean variable x_i so that

$$x_i = \texttt{true} \quad \Leftrightarrow \quad y_i \in [5/6; 1] \tag{4.2}$$
$$x_i = \texttt{false} \quad \Leftrightarrow \quad y_i \in [0; 1/6] \tag{4.3}$$

2. Construct $\bar{l}$ for each literal l:

$$l = x_i \quad \mapsto \quad \bar{l} = y_i \tag{4.4}$$
$$l = \neg x_i \quad \mapsto \quad \bar{l} = 1 - y_i \tag{4.5}$$

   so that

$$l = \texttt{true} \quad \Leftrightarrow \quad \bar{l} \in [5/6; 1] \tag{4.6}$$
$$l = \texttt{false} \quad \Leftrightarrow \quad \bar{l} \in [0; 1/6] \tag{4.7}$$

3. Construct $p_j = \bar{l}_{j,1} + \bar{l}_{j,2} + \bar{l}_{j,3} - 2/3$ for each clause $C_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ so that $C_j(\vec{x}) = \texttt{true}$ if and only if $p_j(\vec{y}) > 0$ for $\vec{y}$ corresponding to $\vec{x}$. If both literals x_i and $\neg x_i$ for some $1 \leq i \leq N$ appear in C_j, we do not construct $\overline{C_j}$ because C_j is always satisfied.

4. Finally, construct a system of polynomial inequalities P and the set of its solutions Δ:

$$P := \left(\wedge_{j=1}^{M} p_j(\vec{y}) > 0\right) \wedge \left(\wedge_{j=1}^{N} 0 < y_j < 1\right) \wedge \left(\wedge_{j=1}^{N} (y_j - 1/2)^2 - 1/9 > 0\right) \tag{4.8}$$
$$\Delta := \left\{\vec{y} \in \mathbb{R}^N \mid P(\vec{y})\right\} \tag{4.9}$$

   for the Boolean formula $\wedge_{j}^{M} C_j$.

This construction is a one-to-one correspondence between Boolean value assignments and sub-cubes at vertices in n-dimensional unit cube. Each sub-cube corresponds to a `false` assignment is excluded by some polynomial inequality $p_j(\vec{y}) > 0$, but each sub-cube corresponds to a `true` assignment is not. Given a `NO` instance of 3-SAT, all sub-cubes are excluded; thus,

$$\mathrm{vol}(\Delta_{\texttt{NO}}) = 0 \tag{4.10}$$

Likewise, given a YES instance of 3-SAT, a sub-cube is included; thus,

$$\mathrm{vol}(\Delta_{\texttt{YES}}) \geq 1/6^N \tag{4.11}$$

Let $q = 6^{-N}/2$ and $n \in \mathbb{N}$ be the smallest satisfying $2^{-n} < 6^{-N}/2$.

$$\mathrm{vol}(\Delta_{\texttt{NO}}) = 0 < q - 2 \cdot 2^{-n} < q < q + 2 \cdot 2^{-n} < 6^{-N} \leq \mathrm{vol}(\Delta_{\texttt{YES}}) \tag{4.12}$$

Therefore, this construction is a polynomial-time many-one reduction from 3-SAT to (n,q)-PERIOD.

$$\wedge_{j=1}^{M} C_j \in \textsf{3-SAT} \Leftrightarrow P \in \textsf{(n,q)-PERIOD} \tag{4.13}$$

$$\textsf{3-SAT} \leq_{\mathrm{p}} \textsf{(n,q)-PERIOD} \tag{4.14}$$

Indeed, some fixed periods have efficient algorithms; for example, π can be efficiently computed by arithmetic-geometric mean; and $\ln(x)$ can also be efficiently computed. We leave whether every fixed period can be computed in time polynomial in n as a future work. Furthermore, given a polynomial inequalities in a disjunctive normal form, can we compute the period in time polynomial in n?

Chapter 5. Conclusion and Perspectives

We have present an algorithm rigorously computing periods up to guaranteed error 2^{-n} which involves only discrete computations given polynomial inequalities with rational coefficients in disjunctive normal form. It takes time exponential in the output precision n and the dimension d, but only polynomial in the maximum degree of the product of the polynomials k. It performs almost identical to our previous algorithms, but it is more concise. Furthermore, its bit-cost analysis is feasible while bit-cost analyses of our previous algorithms seem infeasible.

For now we have focused on the case of two (and three) variables. Future work will extend in that, and the following directions:

1. We will try to analyze the time and space complexity of the algorithm in bit-cost in terms of the bit length of input.
2. Picking up on the first paragraph of Chapter 3, we will try to identify reasonable parameters of the polynomials $p \in \mathbb{Q}[x_1, \cdots, x_d]$ under consideration, in addition to their maximal degree bound k, to devise a refined rigorous parameterized bit-cost analysis.
3. The question remains open as of whether every fixed period can be computed in time polynomial in n although we have seen the general problem of computing periods is NP-hard.
4. In the spirit of *Experimental Mathematics*, we plan to algorithmically search for new (candidate, linear or algebraic) relations among periods.

Bibliography

[Bai17] David H. Bailey. Jonathan borwein: Experimental mathematician. *Experimental Mathematics*, 26(2):125–129, 2017.

[BDC01] Verónica Becher, Sergio Daicz, and Gregory Chaitin. A highly random number. In *Combinatorics, computability and logic*, Springer Ser. Discrete Math. Theor. Comput. Sci., pages 55–68, London, 2001. Springer.

[Can74] Georg Cantor. Ueber eine eigenschaft des inbegriffs aller reellen algebraischen zahlen. *Journal für die reine und angewandte Mathematik (Crelle's Journal)*, 1874(77), 1874.

[CPZ18] Junhee Cho, Sewon Park, and Martin Ziegler. Computing periods .... In M. Sohel Rahman, Wing-Kin Sung, and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*, volume 10755 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2018.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[HRR90] Joos Heintz, Tomás Recio, and Marie-Françoise Roy. Algorithms in real algebraic geometry and applications to computational geometry. In Jacob E. Goodman, Richard Pollack, and William Steiger, editors, *Discrete and Computational Geometry: Papers from the DIMACS Special Year*, volume 6 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 137–164. DIMACS/AMS, 1990.

[Ifr00] Georges Ifrah. *The Universal History of Numbers: From Prehistory to the Invention of the Computer.* John Wiley & Sons, 2000.

[Kan03] Yasumasa Kanada. 計算機による円周率計算 (特集 円周率 π). *J. Mathematical Culture*, 1(1):72–83, 2003.

[KF10] Dieter Kratsch and Fedor V. Fomin. *Exact Exponential Algorithms.* EATCS. Springer, 2010.

[Ko91] Ker-I Ko. *Complexity Theory of Real Functions.* Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991.

[Koi99] Pascal Koiran. The real dimension problem is NP_R-complete. *Journal of Complexity*, 15(2):227–238, 1999.

[Kut87] Miroslaw Kutylowski. Small Grzegorczyk classes. *Journal of the London Mathematical Society*, 36(2):193–210, 1987.

[KZ01] Maxim Kontsevich and Don Zagier. Periods. In Björn Engquist and Wilfried Schmid, editors, *Mathematics unlimited — 2001 and beyond*, pages 771–808. Springer, 2001.

[Mee06] Klaus Meer. Optimization and approximation problems related to polynomial system solving. In *Proc. 2nd Conference on Computability in Europe (CiE'06)*, pages 360–367, 2006.

[Mül01] Norbert Th. Müller. The iRRAM: Exact arithmetic in C++. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis*, volume 2064 of *Lecture Notes in Computer Science*, pages 222–252, Berlin, 2001. Springer. 4th International Workshop, CCA 2000, Swansea, UK, September 2000.

[MZ14] Norbert Th. Müller and Martin Ziegler. From calculus to algorithms without errors. In Hoon Hong and Chee Yap, editors, *Proc. 4th International Congress on Mathematical Software (ICMS)*, volume 8592 of *Lecture Notes in Computer Science*, pages 718–724. Springer, 2014.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[PP16] Patric Popescu-Pampu. *What is the Genus?*, volume 2162 of *Lecture Notes in Mathematics*. Springer, 2016.

[Rit63] Robert W. Ritchie. Classes of predictably computable functions. *Trans. Amer. Math. Soc.*, 106(1):139–173, 1963.

[Sko08] Dimiter Skordev. On the subrecursive computability of several famous constants. *Journal of Universal Computer Science*, 14(6):861–875, 2008.

[SWG12] Dimiter Skordev, Andreas Weiermann, and Ivan Georgiev. M^2-computable real numbers. *J. Logic Comput.*, 22(4):899–925, 2012.

[Tur37] Alan M. Turing. On computable numbers, with an application to the "Entscheidungsproblem". *Proceedings of the London Mathematical Society*, 42(2):230–265, 1937.

[TZ10] Katrin Tent and Martin Ziegler. Computable functions of reals. *Münster J. Math.*, 3:43–65, 2010.

[VS17] Juan Viu-Sos. A semi-canonical reduction for periods of kontsevich-zagier. *arXiv*, 1509.01097, 2017.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[Yos08] Masahiko Yoshinaka. Periods and elementary real numbers. *arXiv*, 0805.0349, 2008.

Acknowledgement

First of all, I thank my parents and my sister, who are always on my side and support, believe, and love me so that I can pursuit my dreams through adventures. I take this time to say that I love you all again. This humble thesis is dedicated to you.

This thesis would have never been possible without the thoughtful advice of Prof. Dr. Martin Ziegler nor the fruitful discussion with lab members and all visitors. It was an extraordinary experience studying, doing a research, and even hosting a conference in this lab for two and a half years since I was an undergraduate intern.

It is a shame that I cannot list all the people I want to thank, but I would like to make this humble thesis as short as possible. Nevertheless, I would like to thank Prof. Dr. Sukyoung Ryu, my advisor during my undergraduate, and Prof. Dr. Kyung-Yong Chwa for helping me to get to ACM-ICPC World Finals and for inviting me to Adama, Ethiopia. It was an invaluable time at KAIST with good people, professors and friends. It will be always remembered as good old days.

Last but not least, I could have never overcome hurdles in my life without all the love from Sumin. I believe we will win through anything together.

Curriculum Vitae

Name	:	Junhee Cho
Date of Birth	:	December 12, 1989
E-mail	:	junheecho@kaist.ac.kr
Website	:	https://junheecho.com

Educations

Sep. 2016 – Aug. 2018	M.S. in Computer Science, KAIST
Feb. 2008 – Aug. 2016	B.S. in Computer Science and in Mathematics, KAIST

Publications

1. **Junhee Cho**, Sewon Park, and Martin Ziegler, "Computing periods . . . ," In M. Sohel Rahman, Wing-Kin Sung, and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation - 12th International Conference, WALCOM 2018, Dhaka, Bangladesh, March 3-5, 2018, Proceedings*, volume 10755 of *Lecture Notes in Computer Science*, pages 132-143. Springer, 2018.
2. **Junhee Cho** and Sukyoung Ryu, "JavaScript Module System: Exploring the Design Space," *Proceedings of the 13th International Conference on Modularity*, April 2014.
3. **Junhee Cho**, "Rewriting JavaScript Module System," *Proceedings of the 12th Annual International Conference Companion on Aspect-oriented Software Development*, March 2013. ACM Student Research Competition (Undergraduate) 1st.
4. Hongki Lee, Sooncheol Won, Joonho Jin, **Junhee Cho**, Sukyoung Ryu, "SAFE: Formal Specification and Implementation of a Scalable Analysis Framework for ECMAScript," *2012 International Workshop on Foundations of Object-Oriented Languages*, October 2012.
5. **Junhee Cho** and Sukyoung Ryu, "JavaScript Module System: Exploring the Design Space (Extended with Formalization)," Technical Report, KAIST, July 2013.

Career

Feb. 2018 – Jun. 2018	T.A of Design and Analysis of Algorithms (CS500), KAIST School of Computing
Sep. 2016 – Dec. 2016	T.A of Operating Systems and Laboratory (CS330), KAIST School of Computing
Jul. 2016 – Aug. 2016	Volunteer teaching algorithms and problem solving for ACM-ICPC, Adama Science and Technology University (ASTU), Adama, Ethiopia
Nov. 2014	Republic of Korea Cyber Commander's Commendation
Jul. 2013 – Apr. 2015	Software Engineer, Republic of Korea Army Sergeant, Republic of Korea Cyber Command
Jul. 2012 – Nov. 2012	Software Maestro Trainee, Republic of Korea Ministry of Knowledge Economy
Feb. 2012	T.A of Programming Languages (CS320), KAIST Department of Computer Science
Jan. 2010 – Jan. 2011	T.A of KAIST Course Management System, KAIST
27 May 2011	13th Place, The 35th Annual ACM-ICPC World Finals
12 Dec. 2010	4th Place, The 2010 ACM-ICPC Asia Tokyo Regional Contest
29 Oct. 2010	1st Place, The 2010 ACM-ICPC Asia Daejeon Regional Contest
5 Nov. 2009	2nd Place, The 2009 ACM-ICPC Asia Seoul Regional Contest