universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Zur Berechnungskomplexität analytischer Funktionen und der Faktorisierung ganzer Zahlen

verfasst von | submitted by

Moisej Plistiev BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von | Supervisor:	emer. o. Univ.-Prof. Dr. Arnold Neumaier

Abstract

The computation of various mathematical functions in a broad sense is a fundamental problem in applied mathematics. This field is vast, and in this work, we will focus on some basic operations, methods for computing analytic functions, and partially touch upon certain operators acting on them. In conclusion, we will briefly describe the current state of factorization and demonstrate the connection between the problems of computing certain analytic functions and finding divisors of integers.

Die Berechnung verschiedener mathematischer Funktionen im weiten Sinne ist ein fundamentales Problem der angewandten Mathematik. Dieses Gebiet ist äußerst umfangreich, und in dieser Arbeit konzentrieren wir uns auf einige grundlegende Operationen, Methoden zur Berechnung analytischer Funktionen und streifen teilweise bestimmte Operatoren, die auf diese wirken. Abschließend beschreiben wir kurz den aktuellen Stand der Faktorisierung und zeigen den Zusammenhang zwischen der Berechnung bestimmter analytischer Funktionen und dem Auffinden von Teilern ganzer Zahlen auf.

Contents

Acknowledgements

First and foremost, I would like to thank Professor Arnold Neumaier, who not only agreed to supervise me but also guided me with great patience throughout this work. I am also deeply grateful to my second examiner, Professor Martin Ehler, for his readiness to support me in many ways. My sincere thanks also go to Professors Marc Mezzarobba, Manuel Kauers, Christoph Koutschan, Chee Yap, and Dr. Fredrik Johansson, whose valuable suggestions greatly contributed to my understanding and to the development of the overall picture. I would also like to thank everyone else who directly or indirectly supported me along this journey.

1 Introduction

In mathematics, there are surprising connections between problems from different fields — for example, Fermat's Last Theorem and the properties of elliptic curves. In this work, we describe some basic computational operations, discuss the computation of real functions and certain operators, and examine the connection between two problems in the domains of computational mathematics and complexity theory — namely, the computation of analytic functions and the problem of factorization.

Analytic functions are a class of functions that can be represented as a convergent Taylor series in the neighborhood of a point. This makes them easily computable locally, but this property becomes less effective at greater distances from the point of expansion.

The task of computational mathematics is to compute numerical values of mathematical objects, whereas the role of complexity theory is to evaluate how efficiently this can be done in terms of computational resources.

We can roughly divide algorithms for basic problems in computer arithmetic and algebra into *simple* — those that run in polynomial time, and *hard* — those requiring substantially more time (although there are also fast algorithms in narrow sense, such as fast multiplication). A common early disappointment in this field is realizing that computational complexity is measured not by the size of the number itself, but by the size of its input — for example, the complexity of handling a number x is determined by $\approx \log(x)$, i.e., the number of digits in its representation.

In the course of this work, an analytic function family $f(x)$ was found whose zeros on the interval from 2 to some number t correspond to the divisors of t. Similar functions had been independently described and studied in earlier work, [**Mateus and Vieira 2011**], but only from the perspective of finding its zeros using methods from complex analysis.

This work proposes a different approach to root-finding, one that operates entirely within the real domain. The method is based on *transformation and modulating the analytic function* — that is, constructing an auxiliary function derived from the original one, with specially tailored properties. This modulation enables the creation of a function whose values can be analyzed to effectively locate the zeros of the original function $f(x)$.

Moreover, the proposed method is versatile: it can be applied to any analytic function, including those in several variables, to find roots over a given interval. This makes the method potentially useful not only for factorization, but also in a broader class of computational problems involving root analysis in both real and complex settings.

In the final part of the work, we reformulate the problem of finding the divisors of a number as the problem of finding the zeros of a function and provide a rough estimate of its complexity. However, to do this, we must first develop the entire framework necessary for this reformulation.

In the second chapter, we provide a description of the computational model based on the Turing machine and introduce basic definitions related to computational complexity.

In Chapter 3, we define floating-point arithmetic, which is essential for real computations, as well as basic algorithms for addition, subtraction, and multiplication.

Chapter 4 presents fundamental algorithms for mathematical functions that do not require Taylor series expansions. We begin with Newton's method and the general requirements for its application. We then discuss the use of Newton's method for computing reciprocals, inverse functions, and roots. After that, we introduce the method of argument reduction, which will be necessary later. This method, for example, is used in computing sine values, as the sine function is periodic, allowing for input data reduction. We then consider the arithmetic-geometric mean method, which enables the computation of logarithm values with very high speed and, through well-known relationships and previously described methods, the computation of many elementary functions.

Chapter 5 first introduces essential facts from complex analysis that will be needed later. It then covers fundamental aspects of Taylor series, their computation, and their role in approximation theory and practical computations. The chapter concludes with a section on holonomic functions—a special class of analytic functions with interesting algebraic properties that can be computed efficiently under certain conditions.

Chapter 6 provides fundamental information on root-finding algorithms and numerical integration, as these tasks are combined in the root-finding method based on the Argument Principle. The chapter discusses key quadrature formulas that guarantee results for a broad class of functions and can be applied to the oscillatory function introduced in Chapter 7. Then. we propose a method for finding the zeros of a broad class of analytic functions. It is based on modulating the original function with another analytic function, in our case, $\operatorname{erfc}(x)$. Integration is also required in this approach.

Chapter 7 begins with an overview of existing methods for finding divisors. The fastest methods, in one way or another, are related to Fermat's method and sieve methods. However, even these are not polynomial-time algorithms. We then reformulate the problem of finding divisors as the problem of finding the roots of an oscillatory function, similar to the one previously discussed in [**Mateus and Vieira 2011**]. A natural approach to solving this problem is the root-finding method based on the Argument Principle. However, in this case, we cannot guarantee that the problem will be solved in polynomial time, as the function has complex conjugate zeros that must remain outside the integration contour, and the complexity of integration depends on the maximum value of the function at some distance from the contour. Consequently, if the contour approaches the poles, the maximum grows extremely rapidly. The original paper first introduced this approach [**Mateus and Vieira 2011**] demonstrates that it is unknown how close the complex zeros are to the real axis, and therefore, it is unclear how close the contour should be to the poles, as the argument Principle method requires zeros to be treated as poles. After that, we analyze the implementation of the proposed method from the Chapter 6 for finding the zeros of the mentioned function. Based on this data, it does not appear to be a polynomial-time method. However, it is free of poles, may provide new functional equations, and offers general insights that may be useful in future applications. Moreover, this chapter presents analytic reduction: the reduction of the problem of integer factorization to the problem of computation of analytic functions.

2 A basic notion of complexity

As a real number is an infinite object, we have to do something about it before use in the finite bias of computation theory

For our purposes, firstly we must define computational complexity and, in general terms, a model of computation. We will adhere to the concept of bit complexity, using the formalism of a **Turing machine**. This abstract machine is defined as a device that serves as a model of computation, which includes the following components:

- an alphabet—a set of symbols available for use, the set of symbols is finite;
- one or more tapes consisting of discrete bits—the minimal units where the symbols of the alphabet are recorded;
- a device with one or more heads that can read symbols from the tape and write symbols to the tape;
- and the device itself, which can be in different states and, depending on its state, can perform operations such as writing, reading, erasing symbols, moving the head one (and only one) cell at a time, or transitioning to a new state, the set of states is finite.

This is a somewhat generalized description of a Turing machine used in complexity analysis, which is sufficient for our purposes[**Hopcroft, Motwani, and Ullman 2006, Chapter 8**]. Sometimes other models are useful, such as Random Access Machine (RAM), which is also an abstract machine, using in some areas of complexity analysis, differ in certain details, such as the number of tapes, the set of commands, and memory access. Specifically, a RAM can instantly access any memory cell, while a Turing machine moves only one cell per step. However, for our purposes, the Turing machine is used. Now we define complexity itself.

Time complexity is the amount of time (operations of the Abstract machine) required to solve an instance of the computational problem as a function of characteristics of the input or output. It is the time required by an algorithm until it executes completely [**Sipser 2013, Chapter 7.1**].

The function of time dependency on the input size can be, for example, n^2 or n^2+n, but we are more interested in the general behavior of algorithms as the amount of data increases. Therefore, in computational theory, the concept of asymptotic complexity is used, for which we need to define the **Big O notation**.

The function $f(n)$ is said to be **Big O** of $g(n)$, denoted as $f(n) = O(g(n))$, if there exists a positive constant c and a constant n_0 such that for all $n \geq n_0$, the inequality $0 \leq f(n) \leq c \cdot g(n)$ holds. We call $g(n)$ an asymptotic upper bound for $f(n)$ [**Balcázar and Gabarro 1989, Chapter 3**].

Analogously, the function $f(n)$ is said to be Ω of $g(n)$, denoted as $f(n) = \Omega(g(n))$, if there exists a positive constant c and a constant n_0 such that for all $n \geq n_0$, the inequality $0 \leq c \cdot g(n) \leq f(n)$ holds. We call $g(n)$ an asymptotic lower bound for $f(n)$. [**Balcázar and Gabarro 1989, Chapter 3**]

Thus, both algorithms, having n^2 and n^2+n steps, will have the same asymptotic complexity. An important note is that we use worst-case complexity—the amount of time required for an algorithm to execute under any circumstances (including both favorable and unfavorable conditions, such as the initial point in Newton's method) and regardless of other factors. The complexity is not the only way to categorize a problem and corresponding algorithm. Clearly, problems such as "is the product of real numbers a and b an integer?" and "what is the product of a and b?" are not the same. We can distinguish (at least) 5 classes of problems and corresponding algorithms:

- **Decisional**, which, for a given input set A, answer "yes" or "no."
- **Functional**, which compute some function.
- **Counting**, which count the number of possible solutions.
- **Searching**, which find elements that satisfy some relation.
- **Optimizational**, which seek the best possible solutions (satisfying some relation).

[Trevisan 2010, Chapter 1] **[Rich 2007, Section 28.10]** This allows us to construct two most important classes, we will use further:

- **P**: Class of decisional problems computable in polynomial time $O(n^a)$
- **FP**: Class of functional problems computable in polynomial time

3 Floating point representation

Since a real number is an infinite object, we need to do something to it before using it within the finite basis used in computation theory. To achieve this, a set F is chosen, which is countable and dense in $\mathbb{R}$. A standard choice is $F = \mathbb{Q}$ or $F = \mathbb{D}$. Here, $\mathbb{D}$ is the set $\mathbb{Z}\left[\frac{1}{2}\right] = \{m2^n : m, n \in \mathbb{Z}\}$, consisting of bigfloats (or dyadic numbers).

If $\tilde{a}$, a are real numbers such that $|\tilde{a}-a| \leq 2^{-i}$, we say $\tilde{a}$ is an i-bit **absolute approximation** of a; if $|\tilde{a} - a| \leq |a| \cdot 2^{-i}$, we say $\tilde{a}$ is an i-bit **relative approximation** of a. **[Yap 2009, Chapter 1]** We define $size(\tilde{a})$ as number of bits in $\tilde{a}$

$$\text{size}(\tilde{a}) = \lceil \log_2(|m| + 1) \rceil + |n|$$

[J. v. d. Hoeven 2016, Chapter 1]. For a complex number $y = x + iy$, $size(z) = size(x) + size(y)$. Later we will use a rough approximation $size(\tilde{a}) = d \approx \log_2(d)$.

For practical realizations we use finite approximations and agreements, one of the most commonly used is the standard IEEE 754.

Floating-point number x in the standard is represented as **[R. P. Brent and Zimmermann 2010, p. 3.1]**:

$$x = (-1)^s \cdot m \cdot \beta^e,$$

where

- $(-1)^s, s \in \{0, 1\}$, is the sign,
- $m \geq 0$ is the significand,
- β is the radix. The most common choice is $\beta = 2$.
- and the integer e is the exponent of x.

In addition, a positive integer n defines the precision of x, which means that the significand m contains at most n significant digits in radix β.

An important special case is $m = 0$ representing zero. In this case, the sign s and exponent e are irrelevant and may be used to encode other information. Normalization in IEEE 754 ensures that floating-point numbers are represented with a leading 1 in the mantissa (implicitly stored), providing a unique, precise, and efficient representation.

The IEEE 754 standard for floating-point representation includes special values to denote exceptional or special cases:

- **Positive and Negative Infinity ($+\infty$ and $-\infty$)**: Used to represent results that exceed the range of representable numbers (overflow).
- **Not a Number (NaN)**: Indicates the result of undefined or invalid operations, such as $0/0$ or the square root of a negative number.
- **Positive Zero and Negative Zero ($+0$ and -0)**: Allow distinguishing the direction of approach to zero in computations, especially in division and reciprocal functions.
- **Subnormal Numbers**: Numbers smaller than the smallest positive normalized number. They enable the representation of values close to zero with reduced precision.

There are several precision standards in IEEE 754, most useful binary are presented in the table:

Format	Bits	Exp. B.	Man. B.	Bias	App. Max Value	Min Normal Value	Min Subnormal Value
Half Precision (16)	16	5	10	15	6.5504×10^{4}	6.10352×10^{-5}	5.96046×10^{-8}
Single Precision (32)	32	8	23	127	3.402823×10^{38}	1.175494×10^{-38}	1.401298×10^{-45}
Double Precision (64)	64	11	52	1023	1.797693×10^{308}	$2.225074 \times 10^{-308}$	$4.940656 \times 10^{-324}$
Quad Precision (128)	128	15	112	16383	$1.189731 \times 10^{4932}$	$3.362103 \times 10^{-4932}$	$6.475175 \times 10^{-4966}$

Table 1: Summary of IEEE 754 Formats with Ranges and Precision.

IEEE 754 defines several rounding modes for floating-point operations:

1. **Round to Nearest (ties to even)**: Rounds to the nearest representable value. If the number is exactly halfway between two values, it rounds to the one with an even least significant digit.

2. **Round Toward Negative Infinity**: Rounds the result toward the largest representable number not greater than the actual value (rounding down).

3. **Round Toward Positive Infinity**: Rounds the result toward the smallest representable number not less than the actual value (rounding up).

4. **Round Toward Zero**: Discards the fractional part, moving toward zero (truncation).

Although the IEEE 754 standard is commonly used for **fixed-precision numbers**, we will instead work with the **arbitrary-precision numbers** described above. In this setting, the length of the number is limited only by the memory capacity of the computing device, which is irrelevant in theoretical constructions. Accordingly, all the algorithms presented below are applicable in arbitrary-precision arithmetic.

3.1 Addition and Subtraction

Now let us move on to basic operations. In many ways, operations with floating-point numbers can be reduced to operations with integers, with their own specific features. The complexity of the integer addition is $O(n)$ for two n-bits numbers, since every bit has to be processed, including handling the carry from the least significant bit to the most significant bit. Accordingly, if two numbers of lengths m and n bits are added, where $n > m$, the addition will have a time complexity of $O(n)$, as the process must handle all bits of the larger number (n).

Floating-point addition and subtraction are more difficult to implement than integer addition/subtraction for two reasons:

- Scaling due to the exponents requires shifting the significands before adding or subtracting them. In principle, we could perform all operations using only integer operations, but this might require huge integers, for example, when adding 1 and 2^{-1000}.
- As carries are propagated from the least to the most significant digits, we may have to examine arbitrarily low input digits to guarantee correct rounding.

Despite this fact they have the same asymptotic complexity, since the algorithm performs the following steps:

1. Extract exponent and fraction bits.
2. Prepend a leading 1 to form the mantissa.
3. Compare exponents.
4. Shift the smaller mantissa if necessary.
5. Add mantissas.
6. Normalize the mantissa and adjust the exponent if necessary.

7. Round the result.

8. Assemble the exponent and fraction back into a floating-point number.

Each of these steps requires at most $O(n)$ operations, which makes the overall complexity of floating-point addition or subtraction also $O(n)$ [**D. M. Harris and S. L. Harris 2013, Chapter 5.3**].

3.2 Multiplication

The exact product of two floating-point numbers $m\beta^e$ and $m'\beta^{e'}$ is $(mm')\beta^{e+e'}$. Therefore, if no underflow or overflow occurs, the problem reduces to the multiplication of the significands m and m', which can then be rounded to fit the precision of the floating-point representation. Now we must consider the main algorithms for integers. The complexity of computing the product of two n-digit numbers is denoted as $M(n)$, and for different algorithms, it has different values. Say, we multiply m and m' with $size(m) \geq size(m')$.

3.2.1 Schoolbook long multiplication

In the schoolbook method, the multiplication is performed digit by digit:

1. Each digit of m' is multiplied by all the digits of m.

2. The resulting partial products are shifted appropriately (by powers of the base) and summed.

If m has n digits and m' has k digits, with $n \geq k$, then:

- Each digit of m' requires $O(n)$ operations to multiply with the digits of m.
- Since m' has k digits, the total complexity is: $O(nk)$.

For the special case where $size(m) = size(m') = n$, the complexity becomes $O(n^2)$, which seen as the worst - case complexity of the method. For a long time, there existed the so called n^2-hypothesis, which asserted that the best possible time for multiplying two numbers is of order $O(n^2)$.

3.2.2 Karatsuba multiplication

The chapter below is close to [**A.Okhotin 2019**].

This method should be examined in detail for several reasons. First, it was the first method to disprove the n^2 hypothesis, which stated that the best complexity for computing a product is $O(n^2)$. Second, this method was generalized into the Toom-Cook algorithm and, in essence, is a special case of it, sufficient for understanding the principles of the algorithm.

Third, it is the first algorithm based on the **divide-and-conquer principle**, or **binary splitti**ng, which have been actively developed since then and will be mentioned later in this work.

For simplicity, let us take two 2-digit numbers. To quickly multiply two 2-digit numbers, by definition, four products must be calculated, then summed, and the carry must be handled.

$$\begin{array}{cccc} & & a_1 & a_0 \\ \times & & b_1 & b_0 \\ \hline & & a_1b_0 & a_0b_0 \\ & a_1b_1 & a_0b_1 & \\ \hline a_1b_1 & a_1b_0 + a_0b_1 & a_0b_0 & \end{array}$$

3.2.2.1 The first idea of Karatsuba's method

The idea is that two 2-digit numbers can be multiplied not with four, but with three multiplication operations: First, the products a_0b_0 and a_1b_1 are calculated as usual. Then, unexpectedly, the numbers $a_1 + a_0$ and $b_1 + b_0$ are multiplied. The result is:

$$(a_1 + a_0)(b_1 + b_0) = a_1b_1 + a_0b_1 + a_1b_0 + a_0b_0,$$

from which the existing products a_0b_0 and a_1b_1 can be subtracted, yielding the desired sum $a_1b_0 + a_0b_1$.

3.2.2.2 The second idea

According to this idea, the same approach can be applied to multiply two n-digit numbers: They are represented as:

$$a = a_1 \cdot 2^{n/2} + a_0, \quad b = b_1 \cdot 2^{n/2} + b_0,$$

where $a_0, a_1, b_0, b_1 \in \{0, \ldots, 2^{n/2} - 1\}$.

The same approach applies here: three products of numbers of length $n/2$ need to be calculated, along with $O(n)$ bit additions (including carry handling).

3.2.2.3 The third idea

The essence is simple: to compute each of the three products of numbers of half the original length, the same method should be applied recursively. At each level of recursion, the problem size is halved, resulting in a recursion depth of $\log_2 n$.

3.2.2.4 Time complexity of the multiplication algorithm

Recursive calls in the multiplication process form a tree. At each call, a certain number of calculations are performed, along with three recursive calls. The total runtime in this case is the sum of the time spent on calculations at all nodes (vertices) of the tree, while the direct overhead of calling the procedures is much smaller.

The subproblem of size n is just one. The subproblems of size $\frac{n}{2}$ are exactly 3. The subproblems of size $\frac{n}{2^i}$ are exactly 3^i. The runtime for each subproblem is linear relative to its size, and for a subproblem of size $\frac{n}{2^i}$, it takes $O\left(\frac{n}{2^i}\right)$ steps. The total number of operations is calculated as follows:

The subproblem of size n is just one. The subproblems of size $\frac{n}{2}$ are exactly 3. The subproblems of size $\frac{n}{2^i}$ are exactly 3^i. The runtime for each subproblem is linear relative to its size, and for a subproblem of size $\frac{n}{2^i}$, it takes $O\left(\frac{n}{2^i}\right)$ steps. The total number of operations is calculated as follows:

$$\sum_{i=0}^{\log_2 n} 3^i \frac{n}{2^i} = n \sum_{i=0}^{\log_2 n} \left(\frac{3}{2}\right)^i = n \cdot \frac{\left(\frac{3}{2}\right)^{1+\log_2 n} - 1}{\frac{3}{2} - 1} = O\left(3^{\log_2 n}\right) = O\left(n^{\log_2 3}\right).$$

3.2.3 Later improvements

For the large numbers some other, but more difficult in construction (that means that they become more efficient only for large numbers) and description methods are developed.

3.2.3.1 Toom - Cook algorithm

This algorithm can be viewed as a generalization of Karatsuba method: it splits the integers m and m' on k parts (Karatsuba method k=2) and performs the operations on the parts. The complexity of 3-way Toom-Cook is $O(n^{\log(5)/\log(3)})$ [**Bilardi and De Stefani 2019, Chapter 1**].

3.2.3.2 Schönhage - Strassen

This algorithm became an important milestone, and essentially its "basic" version is one of the fastest algorithms that remain practical, while the faster ones presented below are, in practice, its modifications. Let α and β be two n-digit integers. Represent them as polynomials:

$$\alpha(x) := a_n x^{n-1} + a_{n-1} x^{n-2} + \cdots + a_1, \tag{3.1}$$

$$\beta(x) := b_n x^{n-1} + b_{n-1} x^{n-2} + \cdots + b_1 \tag{3.2}$$

where a_i and b_i are the decimal digits of α and β, respectively. Then:

$$\alpha = \alpha(10), \quad \beta = \beta(10) \tag{3.3}$$

To compute the product $\alpha \cdot \beta$, we use the **Fast Fourier Transform (FFT)** to efficiently compute the coefficients of the product polynomial:

$$\gamma(x) := \alpha(x) \cdot \beta(x) \tag{3.4}$$

The overall asymtotic complexity of the method is $O(n \log n \log\log n)$. A more thorough exposition can be found in [**J. M. Borwein and P. B. Borwein 1998, Section 6.3**].

3.2.3.3 Harvey-van der Hoeven

The methods building upon the approaches introduced by Schönhage and Strassen have continued to evolve. The first significant improvement was made by Fürer, who reduced the complexity further. Later, Harvey and van der Hoeven achieved an even better result, bringing the complexity down to $O(n \log n)$. However, this algorithm becomes efficient only for numbers with $size(m) \geq 2^{1729^{12}}$. For a comprehensive description, refer to [**Harvey and J. v. d. Hoeven 2021**].

4 Evaluation of elementary functions without Taylor series

Here we give some basic methods for evaluation of the algebraic and transcendental functions. Although the most intuitive approach is the Taylor series method, we must first review auxiliary techniques, largely based on Newton's method, which serve for computing inverse functions, reciprocals, roots, and, more generally, any transformation of a function that can be expressed as an equation. We should also introduce the method of argument reduction, which is applied to many functions, including in combination with the Taylor series method itself. Secondly, it is necessary to present the method of functional equations, which allows one to express certain functions in terms of others, and finally to demonstrate the Arithmetic-Geometric Mean (AGM) method, which is employed for computing the logarithm.

Then, the entire framework we have developed so far can be applied to the computation of elementary functions: by means of the AGM one can compute the logarithm, then obtain the exponential of the same number as its inverse function, and using the values of the exponential we can compute the trigonometric functions through functional equations, while applying argument reduction to minimize the number of operations.

4.1 Newton's method and elementary functions

Newton's method is an important tool used for computing quotients, roots, and inverse functions. Newton's method is a numerical approach to approximate a simple zero ζ of a differentiable function $f(x)$. A simple zero satisfies $f(\zeta) = 0$ and $f'(\zeta) \neq 0$.

The method is based on constructing successive linear approximations to $f(x)$ near ζ. Assuming an initial guess x_0 close to ζ and using Taylor's theorem, we can expand $f(x)$ as:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2} f''(\xi),$$

where ξ lies between x and x_0. Setting $f(\zeta) = 0$,

$$0 = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2}f''(\xi).$$

If x_0 is close to ζ, the term $\frac{(x-x_0)^2}{2}f''(\xi)$ is small and can be ignored in the first approximation. This simplifies the equation to:

$$0 \approx f(x_0) + (x - x_0)f'(x_0).$$

Rearranging for x, we obtain:

$$x \approx x_0 - \frac{f(x_0)}{f'(x_0)}.$$

This leads to the first approximation:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

[**Burden and Faires 2010, Chapter 2.3**].

This iterative process is generalized as:

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}, \quad j = 0, 1, 2, \ldots.$$

If the initial guess x_0 is sufficiently close to ζ, the sequence $\{x_n\}$ converges to ζ. The order of convergence is quadratic in case of simple zero, as the error at the $(n+1)$th step satisfies:

$$|e_{n+1}| \leq K|e_n|^2,$$

where $e_n = x_n - x$ is the error at the nth iteration and K is a constant independent of n.

A more detailed analysis shows:

$$e_{n+1} = \frac{f''(x)}{2f'(x)}e_n^2 + O(e_n^3),$$

provided $f \in C^3$ near x_0, $f''(x) \neq 0$, and $f'(x)$ is sufficiently nonzero. This confirms the quadratic convergence of Newton's method [**R. P. Brent and Zimmermann 2010, Chater 4.2**].

4.2 Newton's method for inverse roots

To compute inverse roots, consider the function:

$$f(x) = y - x^{-m},$$

where m is a positive integer, and y is a positive constant. It's clear that The goal is to find x such that $f(x) = 0$, which implies:

$$x^{-m} = y \quad \text{or equivalently} \quad x = y^{-1/m}.$$

Using Newton's method, the iteration formula becomes:

$$x_{j+1} = x_j + \frac{x_j(1 - x_j^m y)}{m}.$$

This iteration converges to $\zeta = y^{-1/m}$, provided the initial approximation x_0 is sufficiently close to ζ.

An interesting observation is that this formula does not involve division, except for the division by the constant integer m. This makes it computationally efficient in scenarios where division is expensive.

Special cases of this method include: - Reciprocals ($m = 1$): This corresponds to computing $1/y$. - Reciprocal square roots ($m = 2$): This corresponds to computing $1/\sqrt{y}$.

These cases are significant enough to warrant separate discussions in the following subsections. But let us make an important remark first. If the quadratic convergence guaranteed, every step 2 the number of correct steps doubles. Than it needs $log(n)$ steps with $F(n)$ operations, $F(n)$ is a complexity of computation of $\frac{f(x_0)}{f'(x_0)}$. But we can do better, since if the initial precision of x_0 is m, $m < n$, we only need $F(m)$ at the firs step, $F(2m)$ at the next step and only the final step has complexity $F(n)$. The efficiency of Newton's method with variable precision relies on the superlinear growth of $F(n)$, the cost of evaluating $f(x)$ and $f'(x)$ with n-digit precision. If $F(n)$ grows superlinearly, the cost of earlier iterations with lower precision $(F(n/2), F(n/4))$ becomes negligible compared to $F(n)$. Thus, total cost is dominated by the final iteration:

$$T_n = O(F(n)).$$

On the next section $F(n) = M(n)$ and $M(n)$ is superliner according to the previous chapter. Therefore the complexity of this type of methods is $M(n)$.

This subsection is largely based on **[J. M. Borwein and P. B. Borwein 1998, Chapter 6.4]** and **[Richard P. Brent 1975, Section 2]**.

4.3 Newton's method for reciprocals

To compute the reciprocal $1/y$, we take $m = 1$ in the general formula for Newton's method for inverse roots. This gives the iteration:

$$x_{j+1} = x_j + x_j(1 - x_j y).$$

This formula is expected to converge to $1/y$, provided the initial guess x_0 is sufficiently close to the true value. To analyze convergence, we define the error term:

$$u_j = 1 - x_j y.$$

The error u_j measures how close x_j is to $1/y$, and $u_j \to 0$ as $x_j \to 1/y$. Multiplying both sides of the iteration by y, we find:

$$1 - u_{j+1} = (1 - u_j)(1 + u_j).$$

Simplifying, this becomes:

$$u_{j+1} = u_j^2.$$

Thus, the error at each step is squared, which means the method exhibits quadratic convergence:

$$u_j = (u_0)^{2^j}.$$

This shows that the error decreases exponentially, provided the initial error $|u_0| < 1$, which corresponds to the condition $x_0 y \in (0, 2)$ for real x_0 and y.

The initial approximation x_0 is determined using a lookup table, where the table is indexed by the first few bits of y.

Since the order of convergence is two, the number of correct bits approximately doubles with each iteration. This allows us to predict in advance how many iterations are required to achieve a desired level of accuracy.

However, this process relies on the assumption that the lookup table is correctly initialized. At first glance, the iteration formula

$$x_{j+1} = x_j(2 - x_j y)$$

may appear simpler than the original form

$$x_{j+1} = x_j + x_j(1 - x_j y).$$

While these two forms are mathematically equivalent, they are not computationally identical. The difference lies in the precision requirements for intermediate calculations.

1. Precision of Error Term: In the original formula, if x_j approximates $1/y$ to within $n/2$ bits, the term $1 - x_j y$ is accurate to $O(2^{-n/2})$. This allows its product with x_j to be computed with reduced precision.

2. Precision in Simplified Form: In the simplified form, the term $2 - x_j y$ evaluates as $1 + O(2^{-n/2})$, but the final product $x_j(2 - x_j y)$ must be computed with full precision of n bits to achieve the desired accuracy for x_{j+1}.

[R. P. Brent and Zimmermann 2010, Section 4.2.2]

4.4 Newton's method for functional inverses

The exposition in this subsection is largely based on **[R. P. Brent and Zimmermann 2010, Section 4.2.5]**. For a given function $g(x)$, its functional inverse $h(x)$ satisfies:

$$g(h(x)) = x,$$

and is denoted as $h(x) = g^{-1}(x)$. Examples of such pairs include: - $g(x) = \ln x$ and $h(x) = \exp x$, - $g(x) = \tan x$ and $h(x) = \arctan x$.

Using Newton's method, we can compute the inverse function $h(y)$ by finding the root ζ of the equation:

$$f(x) = y - g(x).$$

This leads to the iteration formula:

$$x_{j+1} = x_j + \frac{y - g(x_j)}{g'(x_j)}.$$

The iteration only involves the computation of $g(x)$ and $g'(x)$, making it efficient if these evaluations can be performed quickly. The complexity of this method is comparable to that of evaluating $g(x)$ itself.

For instance, if $\ln(x)$ is efficiently implemented, this approach can also be used to compute $\exp(x)$ with similar efficiency. For the specific case where $f(x) = y - \ln(x)$, the iteration simplifies to:

$$x_{j+1} = x_j + x_j(y - \ln(x_j)).$$

4.5 Argument reduction and functional equations

4.5.1 General introduction

In numerical mathematics, a **functional equation** is an analytically established relation between the values of one or several functions at different arguments, which can be exploited for computation. Such relations are fundamental tools for **argument reduction**, **expressing functions in terms of others**, and for constructing efficient **recurrence-based algorithms**.

These equations are used for:

1. **argument reduction** - to reduce the evaluation of a function at "complicated" arguments to "simpler" ones;
2. **recurrent application** - to compute a whole range of values of a function from a limited set of initial ones;
3. **defining functions through other elementary functions** - which simplifies and unifies numerical algorithms.

Some important identities are:

$$\arctan(x) = \Im\left(\ln(1 + ix)\right),$$
$$\cos(x) + i\sin(x) = e^{ix}$$

[**R. P. Brent and Zimmermann 2010, Section 4.8.5**].

$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}, \qquad \cos(z) = \frac{e^{iz} + e^{-iz}}{2}$$

[**Abramowitz and Stegun 1964, Formulas 4.3.1-4.3.2**].

$$\sinh(z) = \frac{e^{z} - e^{-z}}{2}, \qquad \cosh(z) = \frac{e^{z} + e^{-z}}{2}$$

[**Abramowitz and Stegun 1964, Formula 4.5.1-4.5.2**].

4.5.2 Argument reduction

The subsection below is based on [**R. P. Brent and Zimmermann 2010, Chapter 4.3.**] Argument reduction, again, is a classical technique used to simplify the evaluation of mathematical functions by reducing the input argument to a domain where the function is easier to compute. The process consists of three main steps:

- **Reduction**: Transform the input x into a reduced argument x'.
- **Evaluation**: Compute the function $f(x')$ for the reduced argument.
- **Reconstruction**: Use a functional identity to compute the original $f(x)$ from $f(x')$.

In some situations, reduction and reconstruction are straightforward. For instance, for $x = 2^k$ in base 2, $f(x) = kf'(1)$, or for periodic functions like $\sin(x)$, periodicity allows reduction to $\sin(x \bmod 2\pi)$. In general, reduction formulas are based on functional identities:

- $\exp(x + y) = \exp(x)\exp(y)$,
- $\log(xy) = \log(x) + \log(y)$,
- $\sin(x + y) = \sin(x)\cos(y) + \cos(x)\sin(y)$,
- $\tan(x + y) = \frac{\tan(x)+\tan(y)}{1-\tan(x)\tan(y)}$.

These identities enable transformations such as: Doubling formulas (e.g., $\exp(2x) = \exp(x)^2$), tripling formulas (e.g., $\sin(3x) = 3\sin(x) - 4\sin^3(x)$).

Types of Argument Reduction:

- Additive Reduction: Applicable when $x' = x - kc$, where c is a constant and k is an integer, common for periodic functions (e.g., $\sin(x), \cos(x)$).
- Multiplicative Reduction: Applicable when $x' = x/c$, where c is a constant, Often used for functions like $\exp(x)$, leveraging doubling formulas.

Not all functions have argument reduction formulas. Some functions, like the Gamma function $\Gamma(x)$, require different techniques where the argument is increased (instead of reduced) to a region where simpler approximations (e.g., Stirling's formula) are effective.

We will observe an important particular case.

4.5.3 Argument reduction of sine and cosine

Argument reduction transforms a general real argument x of $\sin(x)$ or $\cos(x)$ into a smaller argument $y \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, where series approximations are stable and efficient, especially for arbitrary-precision computations. Here and later we accept that the value of π is known up to the some $n' > n$. The cost of computation of π will be discussed later.

4.5.3.1 Key identities

$$\begin{aligned}
&\sin(x + 2\pi k) = \sin(x), && \cos(x + 2\pi k) = \cos(x),\\
&\sin(-x) = -\sin(x), && \cos(-x) = \cos(x),\\
&\sin(\pi - x) = \sin(x), && \cos(\pi - x) = -\cos(x),\\
&\sin\left(\frac{\pi}{2} - x\right) = \cos(x), && \cos\left(\frac{\pi}{2} - x\right) = \sin(x).
\end{aligned}$$

4.5.3.2 Algorithm steps

Input: $x \in F$, $F = \mathbb{Q}$ or $F = \mathbb{D}$, precision n (in bits).

Output: $y \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, sign and function indicator.

1. **Range reduction:** Compute $r = x \bmod 2\pi$, with $r \in [0, 2\pi)$.
   In arbitrary-precision arithmetic, use high-precision division and multiplication.
2. **Reduction to symmetric interval:**
   If $r > \pi$, replace $r \leftarrow r - 2\pi$, so $r \in (-\pi, \pi]$.
3. **Reduction to core interval:**
   Write $r = k\frac{\pi}{2} + t$, with $k \in \{-2, -1, 0, 1, 2\}$ and $t \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.
   This is achieved by setting $k = \text{round}(2r/\pi)$, then $t = r - k\frac{\pi}{2}$.
4. **Apply symmetries:**
   Use the table below to express $\sin(x)$ or $\cos(x)$ in terms of $\sin(t)$ or $\cos(t)$:

$k \bmod 4$	$\sin(x)$	$\cos(x)$
0	$+\sin(t)$	$+\cos(t)$
1	$+\cos(t)$	$-\sin(t)$
2	$-\sin(t)$	$-\cos(t)$
3	$-\cos(t)$	$+\sin(t)$

5. **Get result:**
   $y \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.

The foregoing exposition follows [**Ng 1992, Section 1 and 2**].

4.5.3.3 Asymptotic Complexity

- **Modulo Reduction:** For large x at n bits of precision, this step requires $O(M(n))$ operations, where $M(n)$ is the complexity of n-bit multiplication.
- **Other Steps:** Require only $O(M(n))$, which is negligible compared to the modulo reduction.

Therefore, the overall complexity is $M(n)$

4.6 The arithmetic-geometric mean

The other important method based on the functional equations, but much more difficult is Arithmetic - geometric mean (AGM). This chapter follows closely [**R. P. Brent and Zimmermann 2010, Chapter 4.8**]

The arithmetic-geometric mean (AGM) iteration, introduced by Gauss and Legendre, is one of the fastest known methods for high-precision computations. It is a non-linear recurrence with a complexity of $O(M(n) \ln n)$, where $M(n)$ is the cost of multiplication. Although the implicit constant can be large (making it less efficient for small n), AGM becomes highly effective for large n.

Given initial values a_0 and b_0, the AGM iteration is defined as:

$$(a_{j+1}, b_{j+1}) = \left(\frac{a_j + b_j}{2}, \sqrt{a_j b_j} \right).$$

For simplicity, we consider only real, positive starting values a_0 and b_0. This method allows to compute the value of logarithm $\ln(x)$, using special statring value of AGM iteration, depending on x. The AGM possesses several noteworthy features:

1. **Quadratic Convergence**:
   - The AGM iteration converges quadratically to a limit denoted as $\mathrm{AGM}(a_0, b_0)$.
   - This means the number of correct digits approximately doubles with each iteration, requiring $O(\log n)$ iterations for n-digit precision.
2. **Cost Per Iteration**:
   - Each iteration involves computing an arithmetic mean and a geometric mean (square root).
   - Using Newton's method, the cost of a square root is $O(M(n))$, so the total cost of one iteration is also $O(M(n))$.
   - Can be used in complex case. But when z is complex, the evaluation of functions like $\exp(z)$ or $\ln(z)$ introduces additional hidden costs in the O-notation due to the increased complexity of arithmetic operations.

- In contrast with Newton method, AGM is not self correcting, which makes us to compute $M(n)$ with the precision n on the every step, therefore composite complexity is $O(M(n)log(n))$

3. **Applications**:

   - Computation of logarithms
   - Computation of other elementary functions, such as exponentials and trigonometric functions using Newton's method and functional
   - Computation of important constants such as π and $\ln 2$

4.6.1 Main theory and first AGM algorithm

The theory of the arithmetic-geometric mean (AGM) is closely connected to the theory of elliptic integrals. There are two main types of complete elliptic integrals:

1. The First Kind:

$$K(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}} = \int_0^1 \frac{dt}{\sqrt{(1 - t^2)(1 - k^2 t^2)}}.$$

2. The Second Kind:

$$E(k) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2 \theta}\, d\theta = \int_0^1 \sqrt{\frac{1 - k^2 t^2}{1 - t^2}}\, dt.$$

[J. M. Borwein and P. B. Borwein 1998, pp. 1.3.1–1.3.2] Here: $k \in [0, 1]$ is called the modulus, and $k' = \sqrt{1 - k^2}$ is the complementary modulus.

It is common notation to use $K'(k)$ and $E'(k)$ for $K(k')$ and $E(k')$, respectively.

Gauss discovered a profound connection between the AGM and elliptic integrals. Specifically:

$$\frac{1}{\mathrm{AGM}(1, k)} = \frac{2}{\pi} K'(k). \tag{G}$$

This relationship allows for efficient computation of the elliptic integral $K(k)$ using the AGM iteration.

Properties of $K(k)$:

1. For $|k| < 1$, the integral $K(k)$ has a series expansion that converges rapidly:

$$K(k) = \frac{\pi}{2} \left(1 + \frac{k^2}{4} + O(k^4)\right).$$

2. Similarly, for $K'(k)$, we have:

$$K'(k) = \frac{2}{\pi} \ln\left(\frac{4}{k}\right) - \frac{k^2}{4} + O(k^4).$$

From the previously established formulas, we can derive:

$$\frac{\pi}{2\,\mathrm{AGM}(1,k)} = \ln\left(\frac{4}{k}\right)\left(1+O(k^2)\right).$$

Thus, if $x = 4/k$ and k is small, the natural logarithm of x can be expressed as:

$$\ln(x) = \frac{\pi}{2\,\mathrm{AGM}(1,4/x)}\left(1+O\left(\frac{1}{x^2}\right)\right).$$

If $x \geq 2^{n/2}$, we can compute $\ln(x)$ to n-digit precision using the AGM iteration. The process takes approximately $2\log(n)$ iterations to converge, assuming $x \in [2^{n/2}, 2^n]$.

To use this formula, we need the constant π, which computation will be briefly described later.

If x is not large enough, we can **expand the argument** by repeatedly doubling it. For example:

$$\ln(2^l x) = l\ln 2 + \ln x,$$

where $\ln 2$ is known. Alternatively, if $x > 1$, squaring x multiple times allows us to compute:

$$\ln(x^{2^l}) = 2^l \ln(x).$$

This approach, with $x = 2$, provides an **efficient way to compute** $\ln(2)$, assuming π is already known.

The error term $O(k^2 \ln k)$ in formula of logarithm can slow down convergence. A rigorous bound for this error is:

$$\left|\frac{\pi}{2\,\mathrm{AGM}(1,k)} - \ln\left(\frac{4}{k}\right)\right| \leq 4k^2(8-\ln k),$$

valid for all $k \in (0,1]$. This bound can be further sharpened for $k \in (0,0.5]$ to:

$$0.37k^2(2.4-\ln k).$$

The presence of the error term $O(k^2\,|\ln k|)$ makes the use of a larger value for x, such as $x = 4/k$, with k chosen smaller than $2^{n/2}$. However, there exists an exact formula that avoids these issues entirely. Before introducing it, it is necessary to define theta functions, which are used to parameterize the AGM iteration.

4.6.2 Theta function and important identities

To proceed with the AGM iteration, we need the theta functions $\theta_2(q)$, $\theta_3(q)$, and $\theta_4(q)$, defined for $|q| < 1$ as follows:

1. Function $\theta_2(q)$:

$$\theta_2(q) = \sum_{n=-\infty}^{\infty} q^{(n+1/2)^2} = 2q^{1/4} \sum_{n=0}^{\infty} q^{n(n+1)}.$$

2. Function $\theta_3(q)$:

$$\theta_3(q) = \sum_{n=-\infty}^{\infty} q^{n^2} = 1 + 2\sum_{n=1}^{\infty} q^{n^2}.$$

3. Function $\theta_4(q)$:

$$\theta_4(q) = \theta_3(-q) = 1 + 2\sum_{n=1}^{\infty} (-1)^n q^{n^2}.$$

The defining power series for theta functions are sparse (nonzero terms only for specific powers), making it computationally simple to evaluate $\theta_2(q)$ and $\theta_3(q)$ for small q.

Theta functions are tied to several classical identities. Two particularly relevant identities are:

$$\frac{\theta_3^2(q) + \theta_4^2(q)}{2} = \theta_3^2(q^2) \quad \text{and} \quad \theta_3(q)\theta_4(q) = \theta_4^2(q^2).$$

The second identity can also be expressed as:

$$\sqrt{\theta_3^2(q)\theta_4^2(q)} = \theta_2^2(q^2),$$

which highlights the connection to the arithmetic-geometric mean (AGM).

Using the above, the AGM iteration can be rewritten in terms of theta functions:

$$\mathrm{AGM}(\theta_3^2(q), \theta_4^2(q)) = \mathrm{AGM}(\theta_3^2(q^2), \theta_4^2(q^2)) = \ldots (\theta_3^2(q^{2^k}), \theta_4^2(q^{2^k})) \cdots = 1$$

for any $|q| < 1$ The limit of this iteration is 1, as q^{2^k} converges to 0, causing both θ_3 and θ_4 to approach 1. The AGM iteration is therefore parameterized by $(\theta_3^2(q^{2^k}), \theta_4^2(q^{2^k}))$ for $k = 0, 1, 2, \ldots$.

Since $\mathrm{AGM}(\theta_3^2(q), \theta_4^2(q)) = 1$ and $\lambda \cdot \mathrm{AGM}(a, b) = \mathrm{AGM}(\lambda a, \lambda b)$ scaling gives $\mathrm{AGM}(1, k') = \frac{1}{\theta_3^2(q)}$ if $k' = \frac{\theta_4^2(q)}{\theta_3^2(q)}$.

Equivalently, since $\theta_2^4 + \theta_4^4 = \theta_3^4$ (Jacobi identity), we have:

$$k = \frac{\theta_2^2(q)}{\theta_3^2(q)}.$$

Furthermore, from the equation (G) shown earlier with $k \to k'$, we have:

$$\frac{1}{\mathrm{AGM}(1, k')} = \frac{2K(k)}{\pi}$$

and so

$$K(k) = \frac{\pi}{2}\,\theta_3^2(q). \tag{S1}$$

Thus, theta functions have a direct connection to elliptic integrals. The parameter q is often referred to as the nome associated with the modulus k.

The relationship $k = \frac{\theta_2^2(q)}{\theta_3^2(q)}$ allows us to compute k in terms of q. Conversely, there is an inverse formula that expresses q in terms of k:

$$q = \exp\left(-\pi \frac{K'(k)}{K(k)}\right),$$

or equivalently:

$$\ln\left(\frac{1}{q}\right) = \pi \frac{K'(k)}{K(k)}. \tag{S2}$$

By substituting earlier results (G) and (S1) into (S2), **Sasaki and Kanada** derive an intriguing expression for the logarithm:

$$\ln\left(\frac{1}{q}\right) = \frac{\pi}{\mathrm{AGM}(\theta_2^2(q), \theta_3^2(q))}.$$

This formula provides an efficient algorithm for computing $\ln(x)$ via AGM.

4.6.3 Second AGM algorithm

Suppose x is large. Define $q = 1/x$ and compute $\theta_4^4(q)$ and $\theta_3^4(q)$ using their defining series. Sasaki and Kanada's formula, modified by replacing q^4 with q to avoid $q^{1/4}$ in $\theta_2(q)$, gives:

$$\ln(x) = \frac{\pi/4}{\mathrm{AGM}(\theta_2^2(q^4), \theta_3^2(q^4))}.$$

There is a trade-off between increasing x by squaring or multiplying by a power of 2 and taking longer to compute $\theta_2(q^4)$ and $\theta_3(q^4)$ from their series.

It appears optimal to increase x until $q = 1/x$ is sufficiently small, such that higher-order terms can be ignored. In this case:

$$\theta_3(q^4) = 2\left(q + q^9 + q^{25} + O(q^{49})\right),$$

$$\theta_4(q^4) = 1 + 2\left(q^4 + q^{16} + O(q^{36})\right).$$

This method requires $x \geq 2^{n/36}$, which is significantly better than the $x \geq 2^{n/2}$ requirement for the first AGM algorithm. This optimization saves approximately four AGM iterations at the expense of a few multiplications.

4.6.4 Computation of number π

π is polynomial time computable constant - it is an important consequence of the current chapter. The aprroached called The *Gauss-Legendre algorithm* uses the arithmetic-geometric mean to compute the number π up to a certain digit. The basement of this algorithm is *Legendre's relation*: for $0 < k < 1$,

$$E(k)K'(k) + E'(k)K(k) - K(k)K'(k) \;=\; \frac{\pi}{2}.$$

A special case, obtained by taking $k = k' = 1/\sqrt{2}$, is

$$\left(2E(1/\sqrt{2}) - K(1/\sqrt{2})\right) K(1/\sqrt{2}) \;=\; \frac{\pi}{2}.$$

The iterations are defined as follows:

$$\begin{aligned} a_0 &= 1, \\ b_0 &= \frac{1}{\sqrt{2}}, \\ t_0 &= \frac{1}{4}, \\ p_0 &= 1. \end{aligned}$$

For each iteration $n \geq 0$:

$$\begin{aligned} a_{n+1} &= \frac{a_n + b_n}{2}, \\ b_{n+1} &= \sqrt{a_n b_n}, \\ t_{n+1} &= t_n - p_n(a_n - a_{n+1})^2, \\ p_{n+1} &= 2p_n. \end{aligned}$$

After several iterations,

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

The algorithm exhibits **quadratic convergence** - the number of correct digits roughly doubles with each step. The asymptotic complexity is $O(M(n) \log n)$. The first eight iterations produce $0, 3, 8, 19, 41, 84, 171$, and 344 digits, which agrees extraordinarily well with the asymptotic [**J. M. Borwein and P. B. Borwein 1998, p. 49**]. Detailed description of these computations can be found in [**Richard P. Brent 1975, Section 8**].

5 Complex analytic functions and Taylor series representation

Next, we turn to the Taylor series method. Since it is inherently connected with the theory of analytic functions, we begin by presenting several fundamental results from this theory.

Then, in Section 5.2, we examine the application of Taylor series to computation and analyze its complexity. Section 5.3 is devoted to the Bit-Burst algorithm for solving differential equations, which can also be used to evaluate analytic functions. One of its steps is the so-called Taylor method, a generalization of the Taylor expansion that allows the series to be shifted along the domain and is applied in solving differential equations with analytic solutions.

5.1 Classical results from complex analysis

Below we give classical results needed for the purposes of analytic functions from complex analysis. Most of them can be found in every book on complex analysis.

A function $f(z)$ is called **analytic (holomorphic)** in a domain $D \subset \mathbb{C}$ if it has complex derivative at every point of D, which is equivalent to having continuous partial derivatives at every point of D, and satisfies the Cauchy-Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x},$$

where $f(z) = u(x, y) + iv(x, y)$, as shown at **[Ahlfors 1979, Chapter 1.2]**. A function analytic everywhere on $\mathbb{C}$ called entire.

If a function $f(z)$ is analytic in a domain D, then it coincides with its **Taylor series expansion** around any point $z_0 \in D$, defined as:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(z_0)}{n!}(z - z_0)^n.$$

Specifically:

$$f(z) = \sum_{n=0}^{\infty} \frac{f^{(n)}(z_0)}{n!}(z - z_0)^n \quad \text{for all } z \in D.$$

[Ahlfors 1979, Chapter 5.1.2]

A Member of the series

$$\frac{f^{(n)}(z_0)}{n!}$$

is called **(n-th) Taylor coefficient**.

Next we will define points, where the function fails to be analytic.

Isolated singular point is a point $a \in \overline{\mathbb{C}}$ of a function f if there exists a punctured neighborhood of this point (i.e., the set $\{0 < |z - a| < r\}$, if the point a is finite, or the set $\{R < |z| < \infty\}$, if $a = \infty$) in which the function f is holomorphic **[Shabat 1969, Chapter 24, Definition 1]**.

An isolated singular point a of a function f is called a **pole** if there exists

$$\lim_{z \to a} f(z) = \infty.$$

Essential singularity is an isolated singular point a of a function f if f has neither a finite nor an infinite limit as $z \to a$ [**Shabat 1969, Chapter 24, Definition 2**].

A function $f(z)$ is called **meromorphic** in a domain $D \subset \mathbb{C}$ if $f(z)$ is holomorphic in D except at a discrete set of isolated points $\{p_k\}$, and At each p_k, $f(z)$ has a pole of order $m \geq 1$, such that:

$$f(z) = \frac{g(z)}{(z - p_k)^m},$$

where $g(z)$ is holomorphic at p_k and $g(p_k) \neq 0$.

If a function $f(z)$ is meromorphic in an annular region $r < |z - z_0| < R$, then it can be represented by a **Laurent series:**

$$f(z) = \sum_{n=-\infty}^{\infty} a_n (z - z_0)^n,$$

where the coefficients a_n are given by:

$$a_n = \frac{1}{2\pi i} \int_\gamma \frac{f(w)}{(w - z_0)^{n+1}}\, dw.$$

The following properties of Laurent series have to be mentioned.

1. The Laurent series converges absolutely and uniformly in the annular region $r < |z - z_0| < R$.

2. If $f(z)$ is meromorphic, it has at most a finite number of negative powers in the series. That is, $f(z)$ can be written as:

$$f(z) = \sum_{n=-m}^{\infty} a_n (z - z_0)^n,$$

where $m \geq 1$ is the order of the pole at z_0, or $m = 0$ if z_0 is not a pole.

3. If $f(z)$ is meromorphic in a punctured disk $0 < |z - z_0| < R$, the Laurent series uniquely represents $f(z)$ in this domain.

4. An isolated singular point a of a function f is an *essential singularity* if and only if the principal part of the Laurent expansion of f in a neighborhood of the point a contains infinitely many nonzero terms [**Shabat 1969, Chapter 24, Theorem 3**].

The definitions and results on meromorphic functions and Laurent series are given in [**Ahlfors 1979, Chapter 5.1.3**].

The residue at a pole z_0 of a meromorphic function [**Shabat 1969, Chapter 25, Theorem II**] $f(z)$ is the coefficient a_{-1} in its Laurent series expansion:

$$f(z) = \sum_{n=-\infty}^{\infty} a_n (z - z_0)^n, \quad \operatorname{Res}(f, z_0) = a_{-1}.$$

In particular, as shown at [**Orlof 2018, Chapter 8.4.1-8.4.2**] for a *simple pole* at z_0, the residue is

$$\text{Res}(f, z_0) = \lim_{z \to z_0} (z - z_0) f(z).$$

and for a *pole of order* m at z_0, the residue is

$$\text{Res}(f, z_0) = \frac{1}{(m-1)!} \lim_{z \to z_0} \frac{d^{m-1}}{dz^{m-1}} \left[(z - z_0)^m f(z) \right].$$

Next, we give an important result, needed later.

Theorem (Residue Theorem)**.** *Let $f(z)$ be a meromorphic function in a domain D, and let γ be a positively oriented simple closed contour in D such that $f(z)$ has no poles on γ. If $\{z_k\}$ are the poles of $f(z)$ inside γ, then:*

$$\frac{1}{2\pi i} \int_\gamma f(z)\, dz = \sum_k Res(f, z_k),$$

[Orlof 2018, Chapter 8.5] □

We now introduce a number of definitions from complex geometry necessary for the formulation and understanding of the **argument principle**.

The argument $\arg(z)$ of a complex number $z = x + iy$ is the angle θ such that:

$$z = |z| e^{i\theta}, \quad \arg(z) = \theta + 2n\pi,\ n \in \mathbb{Z},$$

as defined at [**Shabat 1969, Chapter 1**].

This form of representing complex numbers clearly shows an important consequence: certain functions, such as $log(x)$, are **multi-valued**. This observation motivates the next definition.

A branch of a multi-valued function is a single-valued subset of the function, defined by restricting the domain of the function. A branch that is designated to be "the main" one is called the **principal branch**, with the corresponding **principal argument**. For example, the principal branch of $\log(z)$ is:

$$\log(z) = \ln|z| + i \arg(z), \quad \arg(z) \in (-\pi, \pi].$$

This indicates that certain nontrivial features may arise. Specifically, we encounter the notion of a **branch cut**. A branch cut can be interpreted as a line of discontinuity introduced into the complex plane to prevent the analytic continuation of a multi-valued function across it. It is the locus along which the function "branches out" into different values, hence the name. For $\log(z)$, a common branch cut is the negative real axis. For $\sqrt{z}$, a branch cut is often chosen along the negative real axis.

Now it is time to present an important result known as the **argument principle**. The connection between the following theorem and the previous definitions will become clear once we take into account that

$$\log f(z)' = \frac{f'(z)}{f(z)},$$

Theorem (Argument Principle)**.** *Let γ be a simple closed curve, oriented counterclockwise, and let $f(z)$ be analytic on and inside γ, except for finitely many isolated points. Suppose that:*

- *$z_1, z_2, \ldots, z_m$ are the poles of $f(z)$ inside γ,*
- *$z_1', z_2', \ldots, z_n'$ are the zeros of $f(z)$ inside γ,*
- *$mult(z_k')$ denotes the multiplicity of the zero at z_k',*
- *$mult(z_k)$ denotes the order of the pole at z_k.*

Then:

$$\int_\gamma \frac{f'(z)}{f(z)}\,dz = 2\pi i \left(\sum_{k=1}^{n} mult(z_k') - \sum_{j=1}^{m} mult(z_j) \right).$$

Proof. **[Orlof 2018, Theorem 11.1]**

Suppose z_0 is a zero of $f(z)$ of multiplicity m. Then near z_0:

$$f(z) = (z - z_0)^m g(z),$$

where $g(z)$ is analytic and $g(z_0) \neq 0$. Differentiating:

$$f'(z) = m(z - z_0)^{m-1} g(z) + (z - z_0)^m g'(z).$$

Thus:

$$\frac{f'(z)}{f(z)} = \frac{m}{z - z_0} + \frac{g'(z)}{g(z)}.$$

The residue at z_0 is:

$$\operatorname{Res}\left(\frac{f'(z)}{f(z)}, z_0 \right) = m = \operatorname{mult}(z_0).$$

Suppose z_0 is a pole of $f(z)$ of order m. Then near z_0:

$$f(z) = \frac{g(z)}{(z - z_0)^m},$$

where $g(z)$ is analytic and $g(z_0) \neq 0$. Differentiating:

$$f'(z) = -\frac{mg(z)}{(z - z_0)^{m+1}} + \frac{g'(z)}{(z - z_0)^m}.$$

Thus:

$$\frac{f'(z)}{f(z)} = -\frac{m}{z - z_0} + \frac{g'(z)}{g(z)}.$$

The residue at z_0 is:

$$\operatorname{Res}\left(\frac{f'(z)}{f(z)}, z_0 \right) = -m = -\operatorname{mult}(z_0).$$

By the Residue Theorem:

$$\int_\gamma \frac{f'(z)}{f(z)}\,dz = 2\pi i \sum \operatorname{Res}\left(\frac{f'(z)}{f(z)}, z\right).$$

Summing over all zeros z'_k and poles z_k, we obtain:

$$\int_\gamma \frac{f'(z)}{f(z)}\,dz = 2\pi i \left(\sum_{k=1}^{n} \operatorname{mult}(z'_k) - \sum_{j=1}^{m} \operatorname{mult}(z_j)\right).$$

[Orlof 2018, Theorem 11.1] □

Next we introduce **maximum Modulus Principle** which states that if a function f is holomorphic in a domain D, then the modulus $|f(z)|$ cannot attain a (local) maximum at an interior point of D, unless f is constant. Or, equivalently:

If f is holomorphic in a domain D and continuous on the closure $\overline{D}$, then $|f|$ attains its maximum on the boundary ∂D. Which follows to the condition for the minimum:

If f is holomorphic and non-vanishing in D, then $|f|$ can attain a (local) minimum inside D only if f is constant. The results on maximum modulus principle are from **[Shabat 1969, Chapter 35]**.

Next important result is famous **Cauchy's Integral formula**, that states that

$$f^{(n)}(z) = \frac{n!}{2\pi i}\int_C \frac{f(w)}{(w-z)^{n+1}}\,dw, \quad n = 0, 1, 2, \ldots \tag{3}$$

where C is a simple closed curve, oriented counterclockwise, z is inside C, and $f(w)$ is analytic on and inside C **[Orlof 2018, Theorem 4.5]**.

And on the top of it we introduce **Cauchy's Estimates**.

Theorem (Cauchy's Estimates)**.** *Suppose f is holomorphic on a neighborhood of the closed ball $B(z_0, R)$, and suppose that*

$$M_R := \max\{|f(z)| : |z - z_0| = R\}. \quad (< \infty)$$

Then

$$|f^{(n)}(z_0)| \le \frac{n!\,M_R}{R^n}.$$

Proof. According to the Cauchy Integral Formula, we have

$$f^{(n)}(z_0) = \frac{n!}{2\pi i}\int_{|z-z_0|=R} \frac{f(z)}{(z-z_0)^{n+1}}\,dz.$$

Then

$$|f^{(n)}(z_0)| = \left|\frac{n!}{2\pi i}\int_{|z-z^*|=R} \frac{f(z)}{(z-z_0)^{n+1}}\,dz\right| \le \frac{n!}{2\pi}\int_{|z-z_0|=R} \left|\frac{f(z)}{(z-z_0)^{n+1}}\right| |dz|.$$

Thus,

$$|f^{(n)}(z_0)| \leq \frac{n!}{2\pi} \int_{|z-z_0|=R} \frac{|f(z)|}{|z-z_0|^{n+1}} \, |dz| \leq \frac{n!}{2\pi} \int_{|z-z^*|=R} \frac{M_R}{R^{n+1}} \, |dz|.$$

Since $|dz| = 2\pi R$ along the contour, we get

$$|f^{(n)}(z_0)| \leq \frac{n!}{2\pi} \cdot \frac{M_R}{R^{n+1}} \cdot 2\pi R = \frac{n! \, M_R}{R^n}.$$

[Orlof 2018, Theorem 4.15] □

5.2 Taylor series representation

Here we present one of the oldest and most common tools for computing analytic functions called **Taylor polynomials**. This method involves a truncated representation of the function up to order m. It is important to note that this approach is mostly suitable only for local computations, so the first steps of such computations is often argument reduction.

5.2.1 Complexity of Taylor series in general (complex) case

First, we define z_0 as point of decomposition. The method is primarily suitable for local computations in terms of distance $|z - z_0|$.

Since Cauchy estimate states that

$$|f^{(k)}(z_0)| \leq \frac{k! \, M_R}{R^k} \tag{5.1}$$

with

$$M_R := \max\{|f(z')| : |z' - z_0| = R\}, \quad (< \infty)$$

then using approximation up to the order m

$$f_a(z) = \sum_{n=0}^{m} \frac{f^{(k)}(z_0)}{k!}(z - z_0)^k,$$

assuming $\frac{|z-z_0|}{R} < 1$ and using formula for geometric progression we get

$$\left| f_m(z - z_0)^m + f_{m+1}(z - z_0)^{m+1} + \dots \right|$$

$$\leq \frac{M_R|z - z_0|^m}{R^m} + \frac{M_R|z - z_0|^{m+1}}{R^{m+1}} + \cdots = \frac{RM_R}{R - |z - z_0|} \left(\frac{|z - z_0|}{R} \right)^m$$

for the tail of the Taylor series of f in z_0, when evaluating in z up to order m. Here we see the known fact, that the error might be greater, if the point of evaluation is farther.

Now we want to compute the value m for given precision. We usually want to compute the value of the function at least up to the first digit after comma, i. e. with the precision $|f(z) - f_a(z)| \leq \epsilon$ with $\epsilon = 2^{-n}$. Now let

$$\tau = \frac{R}{(z - z_0)}, \quad C_1 = \frac{\log 2}{\log \tau}, \quad C_2 = \frac{\log(2M_R) - \log(1 - \tau^{-1})}{\log \tau} \tag{5.2}$$

i. e.

$$C_1 = \log_\tau(2), C_2 = \log_\tau(\frac{2M_R}{1 - \tau^{-1}}) \tag{5.3}$$

then

$$|f(z) - \sum_{k=0}^{m-1} \frac{f^{(k)}(z_0)}{k!}(z - z_0)^k| = |\sum_{k=m}^{\infty} \frac{f^{(k)}(z_0)}{k!}(z - z_0)^k| \leq \frac{1}{2}\frac{2M_R\tau^{-m}}{1 - \tau^{-1}} =$$

$$\frac{1}{2}\tau^{\log_\tau(\frac{2M_R\tau^{-m}}{1-\tau^{-1}})} = \frac{1}{2}\tau^{\log_\tau(\frac{2M_R}{1-\tau^{-1}}\tau^{-m})} = \frac{1}{2}\tau^{\log_\tau(\frac{2M_R}{1-\tau^{-1}})-m} = \frac{1}{2}\tau^{C_2-m}$$

then

$$\frac{1}{2}\tau^{C_2-m} \leq \frac{\epsilon}{2}$$

if

$$\log(\tau^{C_2-m}) \leq \log(\epsilon) \Longleftrightarrow (C_2 - m)\log(\tau) \leq log(\epsilon)$$

$$\Longleftrightarrow m \geq -\frac{log(\epsilon)}{\log(\tau)} + C_2 = \frac{log(\epsilon^{-1})}{\log(\tau)} + C_2,$$

then to guarantee the error

$$\left|f(z) - \left(f_0 + \cdots + f_{m-1}(z_0 - z)^{m-1}\right)\right| \leq \frac{\varepsilon}{2},$$

and therefore with the precision $\leq \epsilon$ the condition

$$m \geq C_1 n + C_2 \tag{5.4}$$

has to be satisfied. **[J. v. d. Hoeven 1999, Chapter 3.1]** This estimate requires analysis to choose R appropriately and is used in the computation of complex-valued functions.

5.2.2 Complexity of Taylor's series in real-valued case

For real-valued functions with real variabl, there is a simpler estimate known as **Taylor's inequality**: we want to compute $f(x)$ using the decomposition in a point x_0, the derivative $f^{(m)}(x) \leq M_{R,x}$ for all $x \in R_{\mathbb{R}}$, $R_{\mathbb{R}} = [x_0, x]$

$$\left|f(x) - \sum_{n=0}^{m} \frac{f^{(k)}(x_0)}{k!}(x - z_0)^k\right| \leq M_{R,x}\frac{|x - x_0|^{m+1}}{(m+1)!}.$$

[**Conrad n.d.**]

Note, M_R is a maximum of the derivative in a single point $f^{(m)}(z)$, wehereis $M_{R,x}$ is a maximum for the whole interval $R_{\mathbb{R}} = [x_0, x]$

5.3 Holonomic functions and bit - burst algorithm

5.3.1 General facts about holonomic functions

A **holonomic function** (also known as a *D-finite function*) is a class of analytic functions that satisfy a linear ordinary differential equation (ODE) with polynomial coefficients. Specifically, a function $f(x)$ is holonomic if there exists an ODE of the form

$$p_p(z)f^{(p)}(z) + p_{p-1}(z)f^{(p-1)}(z) + \cdots + p_1(z)f'(z) + p_0(z)f(z) = 0, \tag{5.5}$$

where $p_0(z), p_1(z), \ldots, p_p(z)$ are polynomials in z.

They have a set of very useful **properties:**

1. **Analyticity and Finite Determination:** Holonomic functions are analytic on their domain of definition and can be locally represented by *power series whose coefficients satisfy linear recurrence relations with polynomial coefficients* called **P-Recursive, D-finite or holonomic sequence [D. Chudnovsky and G. Chudnovsky 1990, Page 115]**. This finite data (the ODE and initial conditions) fully determines the function.
2. **Closure Properties:** The class of holonomic functions is closed under several algebraic and analytic operations:
   - **Addition and Multiplication:** If $f(x)$ and $g(x)$ are holonomic, then $f(x)+g(x)$ and $f(x) \cdot g(x)$ are also holonomic.
   - **Differentiation and Integration:** The derivative of a holonomic function is holonomic, and indefinite integrals of holonomic functions are also holonomic.
   - **Composition:** A composition of the functions $f(g(x))$, where $g(x)$ is algebraic and $f(x)$ is holonomic is again holonomic.
3. **Singularity distribution:** Singularities of a holonomic function can occur only at the zeros of the polynomial $p_q(z)$ in (5.5). This allows us to predict both how and where the Taylor series will converge. Moreover, there exist so-called **false singularities**: these are zeros of (5.5) which, for the given initial conditions, do not actually produce a singularity at that point **[J. v. d. Hoeven 1999]**. More formally, the points where the function may fail to be analytic are called singular points.

Moreover, many well-known special functions—such as the exponential function, error function, Airy function, and Bessel functions—are holonomic. Their defining ODEs fall into the holonomic framework, enabling systematic study and algorithmic manipulation.

5.3.2 Fast evaluation technique

The presented method consists of two parts — **computing the Taylor series** using the binary splitting technique (a variant of which is mentioned in connection with the Toom-Cook algorithm), and **performing analytic continuation** at various points between the values of $f(z)$ and $f(z_0)$.

This algorithm was discovered by the Chudnovsky brothers in the late 1980s [**D. V. Chudnovsky and G. V. Chudnovsky 1987**][**D. Chudnovsky and G. Chudnovsky 1990**], and later independently rediscovered by Joris and van der Hoeven in the late 1990s [**J. v. d. Hoeven 1999**]. The most complete analysis can be found in the work of Marc Mezzarobba [**Mezzarobba 2011**]. Here we will only provide a brief outline and the most essential results on complexity, following the approach and notation of van der Hoeven.

The first point to keep in mind is that this is an algorithm designed for computing a large number of digits. In other words, with a fixed evaluation point z, the main goal is to increase the number of output digits. This is the key aspect in assessing the asymptotic complexity.

Before the start we set $\mathbb{K} = \mathbb{F} = \mathbb{Q}(i)$ and every $z \in \mathbb{F}$ and the usual Euclidean norm for vectors V with q entries:

$$\|V\| = \sqrt{V_1^2 + \cdots + V_q^2},$$

and the usual operator norm for matrices M:

$$\|M\| = \sup_{\|V\|=1} \|MV\|.$$

The starting point of the algorithm is the differential equation as above and a vector of initial conditions

$$F(z) = \begin{pmatrix} f(z) \\ \vdots \\ f^{(p-1)}(z) \end{pmatrix}.$$

The members $f^k, \ldots f^{k+q-1}$, as well as $f^k(z-z_0)^k, \ldots f^{k+q-1}(z-z_0)^{k+q-1}$ can be derived from the equation 5.5 using recurrence relation

$$Q_q(k) f_{k+q} + \cdots + Q_0(k) f_k = 0,$$

or

$$Q_q(k) f_{k+q}(z-z_0)^{k+q} + \cdots + \left[(z-z_0)^q Q_0(k)\right] f_k (z-z_0)^k = 0,$$

with $Q_0, \ldots, Q_{q-1} \in \mathbb{K}[k]$, which can be directly obtained from (5.5) [**J. v. d. Hoeven 1999, Proposition 2**], (as shown, for example, in [**Elias 2025, Chapter 7.2**]. Now, we will assume, that $(z - z_0) = 1$

Now we define

$$N_k = \begin{pmatrix} 0 & 1 & & 0 \\ & \ddots & \ddots & \\ 0 & & 0 & 1 \\ -\frac{Q_0(k)}{Q_0(k)} & -\frac{Q_1(k)}{Q_0(k)} & \cdots & -\frac{Q_{q-1}(k)}{Q_0(k)} \end{pmatrix}. \tag{5.6}$$

$$\Phi_k = \begin{pmatrix} f_k \\ \vdots \\ f_{k+q-1} \end{pmatrix}.$$

Hence

$$\Phi_{k+1} = N_k \Phi_k$$

and

$$\Phi_k = N_{k-1} \cdots N_0 \Phi_0.$$

5.3.2.1 Binary splitting

Next, we will show that the summation of the terms of the series can be replaced by a product of matrices. First, we define the vector $\Sigma_{k,l}$, which represents the sum of l terms of the sequence. Then we introduce the matrices M and N, and, using their modified versions M' and N', the computation of the sum of the sequence terms will be carried out by means of binary splitting.

The term **binary splitting** refers to the technique of reorganizing the product into a **balanced tree** of subproducts of matrices recursively, as shown later. The height of the tree is $\log(m)$, where m is number of member of decomposition needed to reach the precision. Instead of computing the blocks sequentially, one after another — which results in multiplying very small blocks by very large ones (in terms of the number of digits involved) — we construct subblocks of approximately equal length, splitting all computations in half at each step.

Now let us introduce

$$\Sigma_{k,l} = \begin{pmatrix} f_k + f_{k+1} + \cdots + f_{k+l-1} \\ f_{k+1} + f_{k+2} + \cdots + f_{k+l} \\ \vdots \\ f_{k+q-1} + f_{k+q} + \cdots + f_{k+q+l-2} \end{pmatrix}.$$

We claim that for all $k \geq 0$ and $l \geq 1$, there exist matrices $M_{k,l}$ and $N_{k,l}$ with coefficients in $\mathbb{F}$ such that

$$\Sigma_{k,l} = M_{k,l} \Phi_k, \qquad \Phi_{k+l} = N_{k,l} \Phi_k.$$

[J. v. d. Hoeven 1999, (11)] Indeed, if $l = 1$, we take $M_{k,1} = N_{k,1} = N_k$. For $l \geq 2$, we compute $M_{k,l}$ and $N_{k,l}$ using binary splitting: let $l = l_1 + l_2$ with $l_1 = \left\lfloor \frac{l}{2} \right\rfloor$. Then we define:

$$M_{k,l} = M_{k,l_1} + M_{k+l_1,l_2} N_{k,l_1},$$
$$N_{k,l} = N_{k+l_1,l_2} N_{k,l_1}.$$

[**J. v. d. Hoeven 1999, (12)**] In order to avoid computations with rational numbers, it is more efficient to write $M_{k;l} = q_{k;l}^{-1} M'_{k;l}$ and $N_{k;l} = q_{k;l}^{-1} N'_{k;l}$, where the numbers $q_{k;l}$ are positive integers and the matrices $M'_{k;l}, N'_{k;l}$ have coefficients in $\mathbb{Z}$. The representation for $M_{k;l}$ and $N_{k;l}$ becomes

$$\begin{aligned} q_{k;l} &= q_{k+l_1;l_2}\, q_{k;l_1}, \\ M'_{k;l} &= q_{k+l_1;l_2}\, M'_{k;l_1} + M'_{k+l_1;l_2}\, N'_{k;l_1}, \\ N'_{k;l} &= N'_{k+l_1;l_2}\, N'_{k;l_1}. \end{aligned} \tag{5.7}$$

Now we have to say about the input, steps and complexity results.

Given $\varepsilon = 2^{-n}$, the algorithm computes an approximation $\tilde{f}(z)$ for $f(z)$ with $|\tilde{f}(z) - f(z)| \leq \varepsilon$. We assume that $z_0 \to z$ is a straight line with $z_0, z \in \mathbb{K}$ and in U, which is an open simply connected domain entirely contained in the circle of analyticity, where the function $f(z)$ is bounded $|f(x)| < M$ (In contrast to Sections 6.2 and 7.3, where the dependence of the maximum on the function is made explicit). Since the function is bounded on the whole domain U, and this boundedness is in turn limited only by the radius of analyticity, it makes sense to let R from 5.2 tend to its maximal possible value, namely the radius of analyticity. This is essentially what happens in Section 4.1 [**J. v. d. Hoeven 1999, Section 4.1**], including for the purpose of complexity estimates. The algorithm proceeds as follows:

Algorithm 1 Taylor series BinSplit

Require: Differential equation, initial condition $F(z_0)$, straight line $z_0 \rightsquigarrow z$, precision ε
Ensure: Approximation of $L\Phi_0$

1: Compute the integer m according to formula (5.5), and use the linear recursion to construct the matrix (5.6).
2: Compute $\Sigma_{0;m}$ using binary splitting (equation (5.7)). Let L denote the first row of $\Sigma_{0;m}$.
3: Compute an approximation $\widetilde{\Phi}_0$ of Φ_0 with entries in $\mathbb{K}$ such that

$$\|\widetilde{\Phi}_0 - \Phi_0\| \;\leq\; \frac{\varepsilon}{2\|L\|}.$$

4: **return** $L\widetilde{\Phi}_0$

Denote $s = \mathrm{size}(z_0) + \mathrm{size}(z)$, $d(z)$ is the distance between a point $z_0 \in U$ and the boundary of U and

$$\tau = \frac{d(z)}{|z - z_0|}.$$

Then $f(z)$ can be evaluated up to precision 2^{-n} in time

$$O\big(M\big(n(s + \log n)\big)\, \log n\, \log^{-1} \tau\big)\,,$$

uniformly in z and z_0, provided that $\log \log \tau = O(n)$. [**J. v. d. Hoeven 1999, Theorem 3**]

5.3.2.2 Analytic continuation and bit-burst algorithm

The conclusions of the previous chapter apply to the case where the entire domain U lies within the radius of convergence. However, a significant number of holonomic functions, including some elementary ones, possess singularities-for instance, the arctangent at i and $-i$. We may, therefore, wish to evaluate such functions outside the radius of convergence. To do this, we must resort to the **analytic continuation** of the function, embodied in the so-called **Taylor's method** for differential equations.

The essence of this method is that we successively construct Taylor series expansions at the points $z_0, z_1, \ldots, z$, transferring the expansion from one point to the next. That is, we compute the expansion at z_0 in such a way that the next point z_1 lies within its disk of convergence, and then evaluate an approximation of the function at this point. We then shift the expansion to z_1 and repeat the process. The same procedure is carried out for the subsequent points, until we finally reach z. In the context of the arctangent example, this method can be used to bypass singularities—we choose the path so that each singularity lies outside the disk associated with every individual expansion point.

In our case, we are dealing with holonomic functions, which are solutions of **linear differential equations**. This allows the process to be simplified. In particular, we can express the equation in matrix form, transforming it in a system of equations.

The solutions of a holonomic ODE of order p form a vector space of dimension p, whose basis vectors are given by the linearly independent solutions of the equation [**Elias 2025, Theorem 4.5**]. We may then construct a $p \times p$ matrix Δ,

$$\Delta_z \;=\; \big(F_0(z)\; F_1(z)\; \cdots\; F_{p-1}(z)\big) \;=\; \begin{pmatrix} f_1(z) & f_2(z) & \cdots & f_{p-1}(z) \\ f_1'(z) & f_2'(z) & \cdots & f_{p-1}'(z) \\ \vdots & \vdots & \ddots & \vdots \\ f_1^{(p-1)}(z) & f_2^{(p-1)}(z) & \cdots & f_p^{(p-1)}(z) \end{pmatrix},$$

where the first entry is the solution itself and the following entries are its derivatives up to order $p-1$. This matrix is called the *fundamental matrix of solutions*. From this matrix we can construct the *state transition matrix*, which transports initial conditions from one point to another,

$$F(z) = \Delta_{z_0 \rightsquigarrow z}\, F(z_0).$$

Moreover, these matrices satisfy

$$\Delta_{z_0 \rightsquigarrow z_1 \rightsquigarrow z} = \Delta_{z_1 \rightsquigarrow z}\, \Delta_{z_0 \rightsquigarrow z_1}\,,$$

And, therefore,

$$F(z) \;=\; \Delta_{z_{m-1} \rightsquigarrow z} \cdots \Delta_{z_1 \rightsquigarrow z_2}\, \Delta_{z_0 \rightsquigarrow z_1}\, F(z_0).$$

We now describe the algorithm itself with significant simplifications, providing only a brief outline. Our current task is to compute an approximation $\tilde{f}(\tilde{z})$ of $f(z)$ during multiplication of the approximations of the transition matrices $\tilde{\Delta}_{z_{m-1} \rightsquigarrow z} \cdots \tilde{\Delta}_{z_1 \rightsquigarrow z_2}\, \tilde{\Delta}_{z_0 \rightsquigarrow z_1}$, and for these,

in turn, the matrices of series summation. In both cases, binary splitting will generally be applied. It is also important to note that the matrix norm can be used to control the truncation error arising from the application of the matrix, which is exploited in the algorithm. For further details, see [**Mezzarobba 2011, Chapter 8.2-8.4**].

Bit burst is a technique of computing $f(z)$ through analytic continuation along the path

$$z_0 \to z_1 \to \cdots \to z_m = \tilde{z},$$

consisting of approximations of z whose accuracy increases geometrically. Let z_ℓ be the truncation of the binary expansion of z to $2^\ell + 1$ digits after the decimal point. Then

$$|z_{\ell+1} - z_\ell| \le 2^{-2^\ell} \quad \text{and} \quad size(z_\ell) = O(2^\ell), \tag{5.8}$$

In other words, the number of significant digits taken doubles at each step.

The algorithm takes a non-singular broken line path $z_0 \rightsquigarrow z$ and a rational number $\varepsilon > 0$ on input, and computes an approximation $\bar{F}(z)$ for $F(z)$ with $\|\bar{F}(z) - F(z)\| \le \varepsilon$. We assume that $\mathbb{K}$ is an algebraic number field and all polinomials $p_i(z)$ in the differential equation (5.5) for this computations are in $\mathbb{K}[z]$.

Algorithm 2 Analytic continuation with bit burst

Require: Differential equation, initial condition $F(z_0)$, set of points $z_0, z_1 \dots z$, precision ε
Ensure: Approximation $\widetilde{F}(z)$ such that $\|\widetilde{F}(z) - F(z)\| \le \varepsilon$

1: Compute all constants needed to control the error. Split the path to satisfy condition (5.8) and compute the intermediate points $z_\ell \in \mathbb{K}$.
2: Compute the product of the approximations of the transition matrices $\widetilde{\Delta}_{v_\ell \rightsquigarrow v'_{\ell'}}$ using binary splitting. Compute the approximations of single-step matrices using the previous algorithm for summation of power series, following the error conditions.
3: Compute $\widetilde{F}(z) = \widetilde{\Delta}_{z_0 \rightsquigarrow z_\ell} F(z_0)$, then return such that

$$\|\widetilde{F}(z) - F(z)\| \le \varepsilon.$$

return $\widetilde{F}(z)$

The complexity of this method is

$$T_n = O\big(M\big(n \log^2 n \, \log\log n\big)\big)$$

[**J. v. d. Hoeven 1999, Theorem 4**]

5.3.2.3 Later development

First of all, the upper bound has been improved in the case of functions with a finite radius of convergence

$$T_n = O\big(M\big(n \log^2 n\big)\big)$$

The general framewor algorithm was extended in a series of works by Joris van der Hoeven. The latest versions allow the algorithm to be used in a broader range of cases—most importantly, they handle various types of singularities using the method of *expedito-summation*, a

variant of *accelero-summation* originally developed for applications in physics. As a result of the most recent paper, it can be stated that the uniform computational complexity of holonomic functions is bounded by $O(M(n)(\log n)^3)$ [**J. v. d. Hoeven 2016, Theorem 1**].

6 Numerical integration and root finding

Now we have to talk about the problems of integration and root finding, which are connected, especially in the case of analytic functions, we want to observe. We will only touch upon general classical methods to introduce the reader to the main approaches. Even within the relatively strict class of analytic functions, there exists a wide variety of highly specialized techniques.

We can roughly divide algorithms along a spectrum from local to global. Local algorithms, such as the previously discussed family of Newton's methods, converge very quickly but require excellent initial approximations to ensure convergence. Global algorithms, on the other hand, typically exhibit slower convergence but are much less dependent on initial approximations. An example of a global method is the Argument Principle method, which will be discussed later. Moreover, global methods can often be combined with local ones.

Next, we will consider classical methods that form the foundation for many other approaches and can be extensively modified to suit specific situations. We begin with binary search. The simplest variation, which will be described below, relies on detecting a sign change over an interval to guarantee the presence of a root. However, this method can undergo numerous modifications — for example, to handle the case of a second-order zero, where the function touches the axis without changing sign. In such situations, the standard sign-change criterion fails, and alternative selection rules, such as choosing the subinterval where the absolute value of the function decreases, are applied.

In what follows, we will present the most basic version of binary search and later use it in a slightly modified form, employing a different convergence criterion.

6.1 Root finding

6.1.1 Binary search

This method finds a zero of a monotonic continuous function $f(x)$ (i.e., a function that is strictly increasing or decreasing). It works by iteratively narrowing the search interval where the zero lies.

1. The function $f(x)$ must be continuous and monotonic on the interval $[a, b]$.

2. The function values at the endpoints must satisfy:

$$f(a) \cdot f(b) < 0,$$

ensuring that a zero exists in the interval (by the Intermediate Value Theorem).

1. Start with an interval $[a, b]$ where $f(a) \cdot f(b) < 0$.

2. Compute the midpoint:

$$m = \frac{a+b}{2}.$$

3. Evaluate $f(m)$: - If $f(m) = 0$, m is the zero. - If $f(a) \cdot f(m) < 0$, update the interval to $[a, m]$. - Otherwise, update the interval to $[m, b]$.

4. Repeat until the interval is sufficiently small.

Binary search converges linearly with a rate proportional to the interval halving. The error at step n satisfies:

$$\text{Error after } n \text{ iterations: } |b_n - a_n| = \frac{|b-a|}{2^n}.$$

Thus, the method requires $O(\log(\frac{|b-a|}{2^{-n}}))$ iterations to achieve a precision n what makes the complexity $O(T(n)\log_2(\frac{|b-a|}{2^{-n}})$, where T is a time to compute $f(x)$ with the precision to the n digits if we compute the function up to the n digits every step.

This method can also be used in a somewhat different way: if we have a certain criterion that allows us to determine the presence or absence of a zero within a known interval, we can subdivide the interval in which we assume a zero to be located and perform the search using this method. One such criterion is the result of complex integration, described below.

6.1.2 Root Finding Method based on Argument Principle:

We take an analytic function $f(x)$ and compute

$$N = \frac{1}{2\pi i} \int_\gamma \frac{f'(z)}{f(z)}\, dz,$$

where N gives the difference between the number of zeros and poles of $f(z)$ within the closed contour γ, counted with multiplicities.

If $f(z)$ is analytic inside the curve and has only the roots of multiplicity 1, this method determines the exact number of roots within the region. By iteratively refining the contour γ by binary search and combining numerical integration with contour deformation, or using Newton identities the location of the roots can be isolated.

There are many methods based on this approach [**Chen 2022, Chapter 1**], and the described before is the most simple or even primitive one, but all of them are as computationally complex at least as evaluating the integral $\int_\gamma f(z)\, dz$. We finally reached the problem of integration.

6.2 Quadrature (numerical) integration

6.2.1 General notes

We need to evaluate numerical methods for application in root-finding techniques. Note, that

$$\frac{f'(z)}{f(z)} = \frac{d}{dz} \ln f(z),$$

which leads us to the necessity of monitoring the variation of the argument in order to correctly account for the branch changes of the logarithm.

We can proceed in two ways. One approach is to divide the contour into discrete segments and compute the argument increment step by step. This is inherently difficult—both in choosing appropriate step sizes and in terms of computational cost—because the segments must be sufficiently small to capture the variation of the argument accurately. This becomes especially challenging for highly oscillatory functions (which we will examine later), since achieving high accuracy effectively requires accounting for every crest and trough, which is difficult when the period is not constant **[Delves and Lyness 1967, Chapter 5]**.

Alternatively, we may apply quadrature integration techniques in different ways as, for example, in **[Delves and Lyness 1967, Chapter, 3, 4, 6]**, which uses the trapezoidal rule, being a special case of the Newton–Cotes method, or others employed in different approaches to root-finding via integration **[Ioakimidis 1987]**. Therefore, the quadrature methods themselves should be discussed in general, with attention to the simplest and most common families

First of all, the definitions:

An n-Point quadrature formula is

$$I = \int_a^b f(x)w(x)\,dx = \sum_{i=1}^{n} w_i f(x_i) + R_n.$$

We also denote

$$I_n = \sum_{i=1}^{n} w_i f(x_i)$$

The w_i and x_i are called quadrature **weights** and **nodes**, $w(x)$ is a **weight function** and R_n the **remainder** or **error [Kahaner, Moler, and Nash 1989, Cahpter 5.2]**. We now will define Bernstein ellipse.

The **Bernstein ellipse** E_ρ with focil ± 1, center $z = 0$, and semi-axes

$$a_\rho = \frac{\rho + \rho^{-1}}{2}, \quad b_\rho = \frac{\rho - \rho^{-1}}{2}, \tag{6.1}$$

for some parameter $\rho > 1$ called the elliptical radius. Note that $a_\rho + b_\rho = \rho$. This ellipse has Cartesian equation

$$E_\rho = \left\{ z : \frac{(\operatorname{Re} z)^2}{a_\rho^2} + \frac{(\operatorname{Im} z)^2}{b_\rho^2} = 1 \right\},$$

and parametric equation

$$E_\rho = \left\{ z = \frac{\rho e^{i\theta} + \rho^{-1} e^{-i\theta}}{2} : \theta \in [0, 2\pi) \right\}.$$

[**Demanet and Ying 2010**]. For all the estimates below, we adopt the convention that the ellipse together with its boundary lies entirely within the domain of analyticity of the functions. Notice that the convergence theorems below are applicable to integrals of analytic functions $f(x)$ on the real axis over the interval $[-1, 1]$,

$$I = \int_{-1}^{1} f(x)\, dx,$$

however, this restriction can easily be overcome by means of a certain type of parametrization that preserves analyticity of the function (provided the contour does not self-intersect, which depends only on the nature of the function itself). Specifically, we can

1. shift the center using a parameter u: $f(x) \mapsto f(x + u)$;
2. change the scale of the argument using a parameter v: $f(x) \mapsto v\, f(vx)$,

since

$$\int_{-1}^{1} v\, f(vx)\, dx = \int_{-v}^{v} f(x)\, dx.$$

Now, let us go to the quadratures. The simplest quadrature rule is known as the family of **Newton–Cotes formulas**.

6.2.2 Family of Newton-Cotes formulas

Let $x_0, x_1, \ldots, x_{n-1}$ be n distinct points lying in the real line and let C be a closed interval containing these points in its interior. Suppose that $f(x)$ is a regular function within C. Let

$$W_n(z) = \prod_{i=0}^{n-1} (x - x_i).$$

We also require x_i to be of the form

$$x_i = -1 + 2i/n \quad (i = 0, 1, \ldots, n)$$

Then the **Newton-Cotes quadrature formula** is defined as

$$\int_{-1}^{1} f(t)\, dt = \sum_{i=0}^{n-1} w_i f(z_i) + R_n(f),$$

where w_i are weights and $R_n(f)$ is the error term.

$$w_k = \int_{-1}^{1} \frac{W_{n+1}(x)}{(x - x_k) W'_{n+1}(x_k)}\, dt \qquad (k = 0, 1, \ldots, n-1)$$

[**Kambo 1970, Chapter II**].

6.2.2.1 Error estimate

Let f be analytic on E_ρ, $\rho > 1$. Then for $n \geq 1$

$$|R_n| \leq K_\rho M_\rho \left(\frac{2\rho}{\rho^2 - 1}\right)^{n+1},$$

where

$$K_\rho = \begin{cases} \frac{4}{3} \cdot \frac{\rho^2 + 1}{\rho^2 - 1}, & \text{if } n \text{ is odd,} \\ 4\left(\frac{2}{105}\right)^{1/2} \cdot \frac{\rho^2 + 1}{\rho(\rho^2 - 1)}, & \text{if } n \text{ is even,} \end{cases}$$

and

$$M\rho = \max |f(z)| \quad \text{on } E_\rho$$

[**Kambo 1970, Chapter II, Theorem 1**]. Next we will talk about **Gaussian quadrature**.

6.2.3 Family of Gaussian quadrature formula

First we define **orthogonal polynomials:** A system of polynomials $P_k(x)$, degree $[P_k(x)] = k$, is called orthogonal on the interval $a \leq x \leq b$, with respect to the weight function $W(x)$, if

$$\int_a^b w(x)\, P_k(x)\, P_m(x)\, dx = 0, \quad (k \neq m;\ n, m = 0, 1, 2, \ldots).$$

The weight function $w(x)$ $[w(x) \geq 0]$ determines the system $p_k(x)$ up to a constant factor in each polynomial [**Abramowitz and Stegun 1964, Section 22.1**].

Now we are ready to define **Gaussian type quadrature** as:

$$\int_a^b f(x)w(x)\, dx = \sum_{i=1}^{n} w_i f(x_i) + R_n,$$

where x_i are the roots of orthogonal polynomials [**Gautschi 2004, Section 1.4**]. Different formulas of the Gaussian quadrature family often differ by the *weight function* $w(x)$, interval and by the system of *orthogonal polynomials* used for interpolation. Here are some classical examples.

Among the Gaussian quadrature formulas, is the **Gauss–Legendre rule** seems to be the most suitable for our purposes since our aim is to examine a bounded, indeed compact, segment of the curve in terms of the increment of the argument, without introducing any distortion by a weight function. Therefore, in what follows we shall restrict our attention to this particular case. Here the weight function is constant, $w(x) = 1$ on $[-1, 1]$, which

Formula	Weight $w(x)$	Polynomials	Interval
Gauss–Legendre	1	Legendre $P_n(x)$	$[-1,1]$
Gauss–Chebyshev	$\dfrac{1}{\sqrt{1-x^2}}$	Chebyshev $T_n(x)$	$[-1,1]$
Gauss–Laguerre	e^{-x}	Laguerre $L_n(x)$	$[0,\infty)$
Gauss–Hermite	e^{-x^2}	Hermite $H_n(x)$	$(-\infty,\infty)$
Gauss–Jacobi	$(1-x)^\alpha(1+x)^\beta, \quad \alpha,\beta>-1$	Jacobi $P_n^{(\alpha,\beta)}(x)$	$[-1,1]$

Table 2: Classical Gaussian quadrature rules with their weight functions and orthogonal polynomials [**Abramowitz and Stegun 1964, Section 25.4**].

makes the construction especially simple: the quadrature nodes are given by the zeros of the **Legendre polynomials** $P_n(x)$.

The nth-degree Legendre polynomial P_n corresponds to the normalization $P_n(1)=1$. One has

$$k_n = \frac{(2n)!}{(2^n n!)^2}, \qquad h_n = \frac{1}{n+\frac{1}{2}}, \qquad h_n = \|P_n\|^2 = \int_{-1}^{1} \big(P_n(x)\big)^2\,dx.$$

In terms of the P_n, the recurrence relation is

$$(k+1)P_{k+1}(x) = (2k+1)\,t\,P_k(x) - k\,P_{k-1}(x), \qquad P_0(x)=1, \quad P_1(x)=x.$$

Each P_n is bounded by 1 on $[-1,1]$. [**Gautschi 2004, Chapter 1.5**]

Now, to finally obtain the Gauss–Legendre quadrature formula, we have to specify the weights, defined as

$$w_i = \frac{2}{(1-x_i^2)\,\big[P_n'(x_i)\big]^2}, \qquad i=1,2,\ldots,n,$$

where x_i are the zeros of the Legendre polynomial $P_n(x)$. In the next step we shall provide an estimate of the remainder term of the Gauss–Legendre quadrature formula.

6.2.3.1 Error estimate

Let a function f be analytic in $[-1,1]$ and analytically continuable to the open Bernstein ellipse E_ρ, where it satisfies $|f(z)|\le M_\rho$ for some M. Then $(n+1)$-point Gauss quadrature with $n\ge 1$ satisfies

$$|I-I_n| \le \frac{64 M_\rho \rho^{-2n}}{15(\rho^2-1)}.$$

[**Trefethen 2018, Chapter 19**]

6.2.4 General remark on convergence

The constant M_ρ depends on ρ, which in the case of many functions makes the situation nontrivial: if the functions involve an exponential component (such as sine or cosine), it can grow exponentially on the contour. On the other hand, one might try to take ρ as large

as possible to accelerate convergence. For those quadrature formulas in which a similar constant on the Bernstein ellipse appears, as for example in the case of **Clenshaw–Curtis** [**Trefethen 2018, Chapter 19**], the same challenge arises.

Suppose our goal is to compute the integral of $f(x)$ on the closed interval $[-v, v]$; for simplicity, let the function be entire. We aim to achieve an approximation error equal or less than $\epsilon \leq 1$. **Thus, with $f(x)$ given, we fix v and the required precision ϵ, while the number of nodes n can be varied to achieve the desired accuracy within the interval $[-v, v]$.** In order to determine the required number of nodes for a prescribed accuracy, and to assess whether this number may become excessively large with respect to the parameters involved in the computation, a rigorous analysis is necessary. Such an analysis must take into account the properties of the function, in particular its growth parameters, especially in the complex domain. such as on the Bernstein ellipse.

Consider functions involving exponential function, such as the sine or cosine. In such cases, we might expect the constant M_ρ in the error bound to grow exponentially as well. We will use maximum modulus principle and properties of Bernstein ellipse to estimate the constant M_ρ. In our example trigonometric functions are used and because

$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}$$

and

$$\cos(z) = \frac{e^{iz} + e^{-iz}}{2},$$

they both have exponential rate of growth in the complex plane, and therefore,

$$|\sin(z)| \leq |\frac{e^{iz} - e^{-iz}}{2i}|,\ |\cos(z)| \leq |\frac{e^{iz} + e^{-iz}}{2}| \Rightarrow$$

$$|\sin(z)|,\ |\cos(z)| \leq e^{a|z|} \leq M_{\rho,f} = ve^{av|z_\rho|}$$

where a is a constant parameter describing the growth of the function, v is a parameter of parametrization, and z_ρ is a point in the complex domain inside the ellipse corresponding the maximum of the function inside the Ellipse, and therefore on the Ellipse according to the maximum modulus principle, moreover, if we take z_ρ on the ellipse, it must satisfy the following condition:

$$\frac{\rho - \rho^{-1}}{2} \leq |z_\rho| \leq \frac{\rho + \rho^{-1}}{2} \tag{6.2}$$

because of (6.1). This makes us to be careful with the function involving exponential and carefully study them on the whole plane to avoid troubles. Note that using the parameter a as a growth parameter, we can bound not only a single function but also certain symmetric counterparts of functions exhibiting exponential growth. Consequently, we obtain an upper bound that, while slightly coarser, applies to a broader class of functions. Since all constants a, v, h, ρ are positive, we may write the constant M in the form

$$M_{\rho,f} = ve^{av(\rho+\rho^{-1})/2}.$$

We can observe, that every estimation of the error term we discussed before may be roughly expressed as

$$|I - I_n| \leq bM_{\rho,f}\rho^{-cn}$$

where b and c are some rational numbers. We can deduce from that the following has to be satisfied to reach the error at least equal ϵ

$$bM_{\rho,f}\rho^{-cn} \leq \epsilon \quad \Rightarrow \quad |I - I_n| \leq \epsilon,$$

$$\rho^{cn} \geq bM_{\rho,f}\epsilon^{-1} \Rightarrow \rho^{-cn} \leq \epsilon(bM_{\rho,f})^{-1}.$$

That leads us to the condition:

$$\frac{\rho^{cn}}{b} \geq \epsilon^{-1}M_{\rho,f} = \epsilon^{-1}ve^{av(\rho+\rho^{-1})/2}.$$

And therefore we need to reach

$$\frac{\rho^{cn}}{b} \geq \epsilon^{-1}M_{\rho,f}.$$

Now, we aim to define the parameter n in terms of v, ρ and f (or, equivalently, a parameter a that depends on f).

We define n as the logarithm of the product $\epsilon^{-1}M_{\rho,f} \cdot b$, scaled by a constant c:

$$n \geq \frac{\log_\rho(\epsilon^{-1} \cdot M_{\rho,f} \cdot b)}{c}. \tag{6.3}$$

Substituting the expression for M_ρ, we compute:

$$n(v,\rho,f) \geq \frac{\log_\rho \epsilon^{-1} + \log_\rho M_\rho + \log_\rho b}{c},$$

$$n(v,\rho,f) \geq \frac{\log_\rho \epsilon^{-1} + \log_\rho \left(ve^{av(\rho+\rho^{-1})/2}\right) + \log_\rho b}{c},$$

$$n(v,\rho,f) \geq \frac{\log_\rho \epsilon^{-1} + avh((\rho+\rho^{-1})/2) \cdot \log_\rho e + \log_\rho v + \log_\rho b}{c},$$

$$n(v,\rho,f) \geq \frac{\ln \epsilon^{-1}}{c\ln\rho} + \frac{av((\rho+\rho^{-1})/2) \cdot \ln e}{c\ln\rho} + \frac{\ln v}{c\ln\rho} + \frac{\ln b}{c\ln\rho}, \tag{6.4}$$

$$n(v,\rho,f) \geq \frac{\ln \epsilon^{-1} + av((\rho+\rho^{-1})/2)\ln e + \ln v + \ln b}{c\ln\rho}. \tag{6.5}$$

Now, we want to compute if we measure complexity in terms of the number of digits of precision d in case of the integer v iwth the base of 2—that is, logarithmically in v—we get:

$$d \approx \log_2 v, \qquad v \approx 2^d,$$

$$n(d,\rho,f) \geq \frac{\ln \epsilon^{-1} + a2^d((\rho+\rho^{-1})/2)\ln e + \ln 2^d + \ln b}{c \ln \rho},$$

$$n(d,\rho,f) \geq \frac{\ln \epsilon^{-1} + a2^d((\rho+\rho^{-1})/2) + d\ln 2 + \ln b}{c \ln \rho}.$$

We want to consider a very rough special case when $\epsilon = 1$, then

$$n(d,\rho,f) \geq \frac{a2^d((\rho+\rho^{-1})/2) + d\ln 2 + \ln b}{c \ln \rho}.$$

Hence, to cancel out the constant and achieve the desired precision, the required number of terms will be exponential in the number of digits d, of v even though the formal convergence rate is exponential in this model.

We have considered an especially interesting case is when the function is bounded on the real line (e.g., $\cos(x)$), but its analytic continuation on the Bernstein ellipse grows exponentially. This means that the seemingly "nice" function (bounded on $\mathbb{R}$) can still be numerically expensive to approximate, due to its behavior in the complex domain.

Moreover, the integration is used mainly for the argument principle-based root finding algorithms, which require appearance of the poles, which can lead to the unpredictable behavior of the error term nearby.

6.3 Proposed method for root finding of analytic functions

6.3.1 General outline and remarks

Further, we propose a method for finding the roots of an arbitrary function $f(x)$ on a compact interval $[X_L, X_R]$, analytic on a domain R such that $[X_L, X_R] \subset R$. The essence of the method is the construction of a **special function** in the form of a chain of analytic transformations,

$$H(x) = \int_{X_L}^{x} h\big(g(af(\xi)^2)\big)\, d\xi,$$

by means of a sequence of analytic processes, including the *initial transformation*, applied to the original function through $g(x)$, and the *modulation*, applied to the result through $h(x)$, followed by *integration*, which produces the step function $H(x)$, and the final *localization*, where the jumps of $H(x)$ are detected and associated with the roots of $f(x)$. The steps are:

1. **Initial transformation.** The function $f(x)$ is mapped to $g(af(x)^2)$. The resulting function is analytic on R, takes values between 0 and 1, and for sufficiently large a is nearly zero almost everywhere, except for sharp peaks located at the zeros of f. The parameter a controls the suppression of nonzero values.

2. **Modulation (optional).** An additional analytic function $h(\cdot)$ can be applied to modify the shape and area of the peaks, thereby enhancing their contrast.

3. **Integration.** The integral of the transformed function yields a **step–like profile**: it remains nearly constant on subintervals without zeros and exhibits localized growth in neighborhoods of the zeros of the original function.

4. **Localization.** These regions of localized growth can be detected by binary search, and the approximate root locations can then be refined by classical methods (e.g., Newton's method).

Thus, the method is realized as a composition of analytic functions, culminating in the construction of a step–like integral function whose regions of growth correspond to the zeros of $f(x)$. The approach can in principle be extended to multivariate functions, but the precise properties and behavior of the resulting construction require further investigation.

Further, we provide a more detailed description of each step together with illustrative examples. But first, let us define the **error function**:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-l^2}\, dl$$

and analogous **complementary error function**:

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$$

the functions are entire.

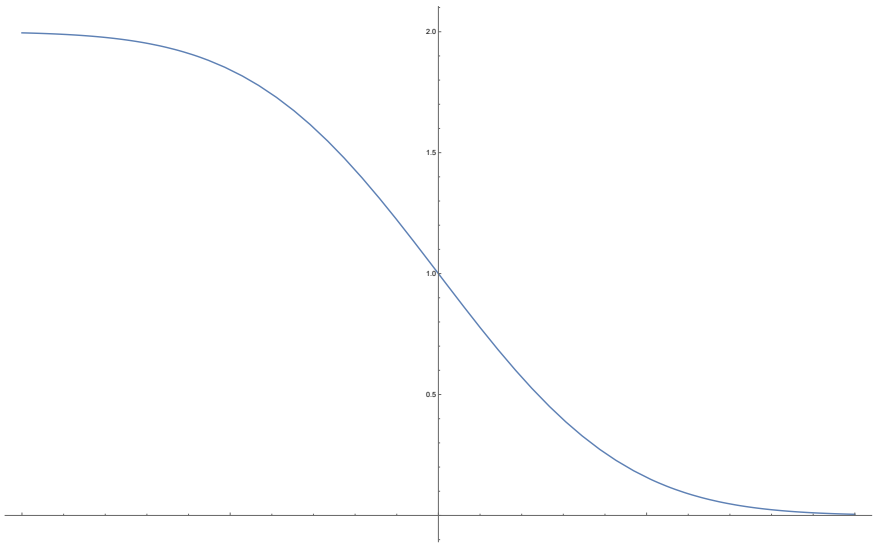

Figure 1: erfc(x)

6.3.2 Step 1 - initial transformation using the error function

This step is based on the fact that the chosen transformation function, in our case the complementary error function $\operatorname{erfc}(x)$, changes rapidly only near $x = 0$ and is practically flat elsewhere. Therefore, when applied to $f(x)$, the values near the zeros of f produce sharp peaks, while other regions are flattened. To control this effect we introduce the

parameter $a > 0$, which enhances the suppression of nonzero values of $f(x)$. As a result, the transformed function takes values close to 1 in neighborhoods of the zeros of $f(x)$ and close to 0 elsewhere, when using $\operatorname{erfc}(af(x)^2)$ with sufficiently large a. The required size of a depends on how close the nearest nonzero values of $f(x)$ are to the actual zeros.

The algorithm is as follows:

- Let $f(x)$ be analytic on the interval $[X_L, X_]$. Transform $f(x)$ using the complementary error function to obtain
$$g(x) = 1 - \operatorname{erf}\big(a \cdot f(x)^2\big).$$
This function takes values in $[0, 1]$, remaining close to 0 for most x while forming sharp peaks near the zeros of $f(x)$, with values approaching 1.
- Choose a sufficiently large parameter a. The larger a is, the sharper the peaks and the flatter the transformed function outside the neighborhoods of the zeros.

We will use the function
$$f(x) := \sin(\frac{\pi x}{2}) + \sin^2(4\pi x) + 0.1$$
as example.

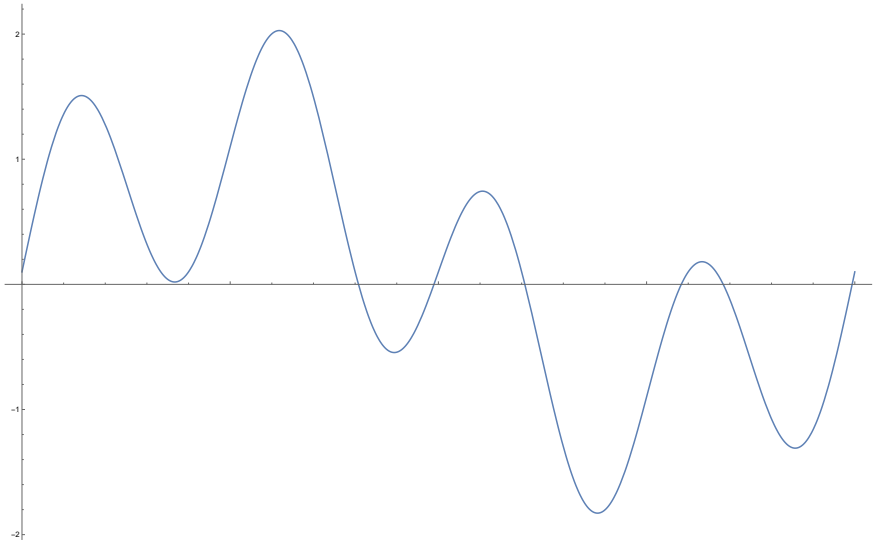

Figure 2: $f(x) := \sin(\frac{\pi x}{2}) + \sin^2(4\pi x) + 0.1$

Increasing parameter a we get more and more flat graph of the function, where all the values of $f(x)$ except x such that $f(x) = 0$ are close to zero, so for x large enough we get a picture, where only values of the function in the small neighborhood of the roots are not close to zero.

6.3.3 Step 2 (Optional) – Modulation of the transformed function

- Apply a modulation to $g(x)$ using either
$$h(x) = -\cos\big(\pi(g(x)+1)^b\big) \quad \text{or} \quad h(x) = \big(g(x)\big)^b,$$

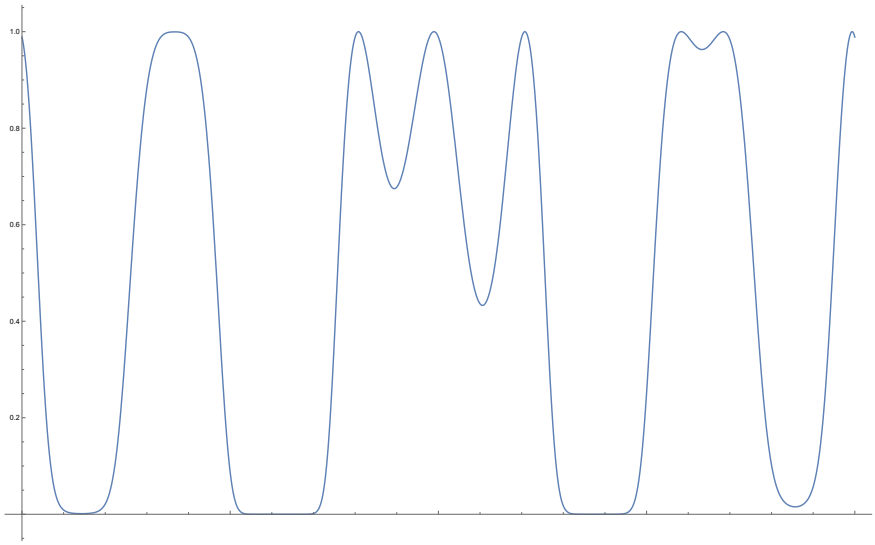

Figure 3: $g(f(x)) := \text{erfc}(\sin(\frac{\pi x}{2}) + \sin^2(4\pi x) + 0.1)$

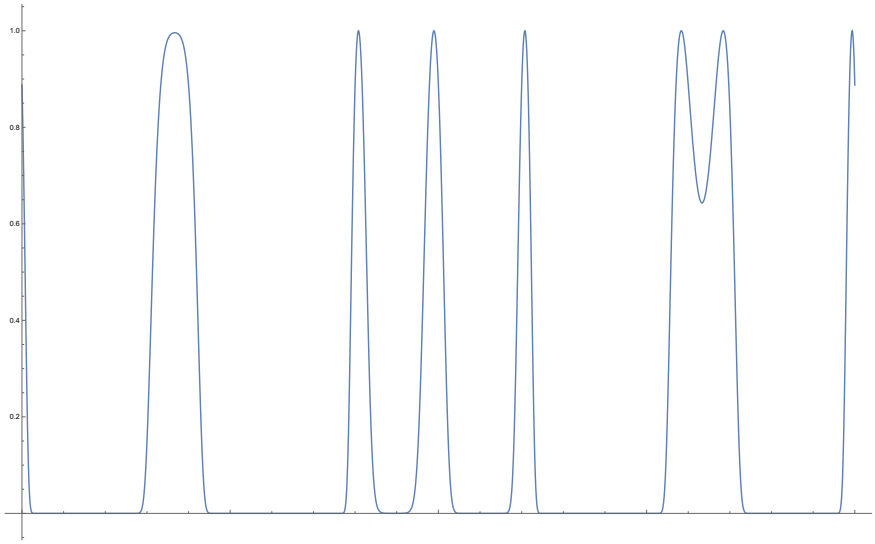

Figure 4: $g(f(x)) := \text{erfc}(10(\sin(\frac{\pi x}{2}) + \sin^2(4\pi x) + 0.1)$

where b is an even integer. The goal of this step is to obtain an integral with a more distinct "ladder" profile, where the steps correspond to the zeros of $f(x)$. As the parameter a increases, the peaks of $g(x)$ may become too narrow and the area under them may shrink. Modulation compensates for this by broadening and amplifying the peaks, thereby increasing their contribution to the integral.

- Select a suitable value of the parameter b. The parameter b controls the sharpness and width of the modulated peaks, and thus determines the clarity of the resulting step–like structure.

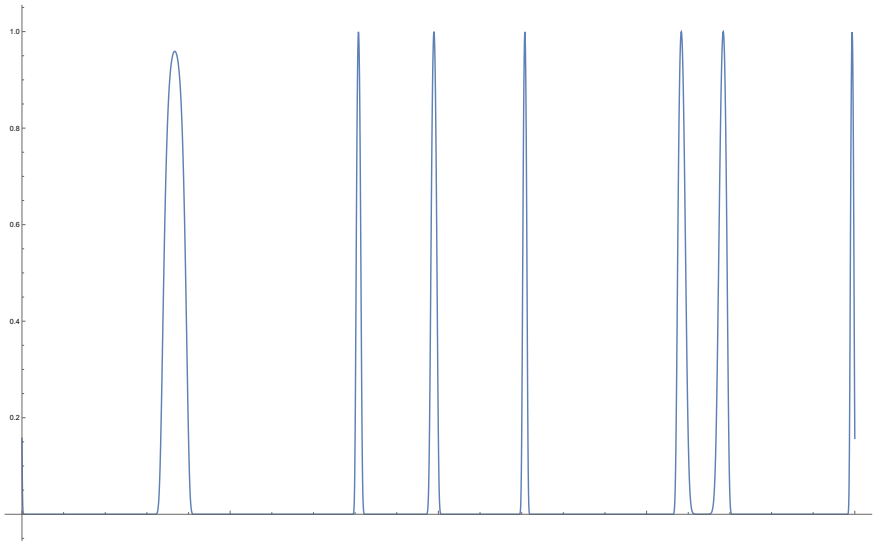

Figure 5: $g(f(x)) := \text{erfc}(100(\sin(\frac{\pi x}{2}) + \sin^2(4\pi x) + 0.1)$

6.3.4 Step 3 – Integration and binary search

- Compute the integral of the modulated function

$$H(x) = \int_{X_L}^{x} h(g(f(\eta)))\, d\eta.$$

  The resulting function is expected to display a step–like profile: flat outside neighborhoods of the zeros of $f(x)$ and increasing in their vicinity.

- Perform a binary search on $H(x)$ within the given interval to localize the regions of growth that correspond to the zeros of $f(x)$. The procedure is as follows: choose two points g_0 and g_1 such that a single zero lies between them. Take the midpoint g_2 of $[g_0, g_1]$ and compare either the *differences*

$$H(g_1) - H(g_2) \quad \text{and} \quad H(g_2) - H(g_0),$$

  or the *ratios*

$$\frac{H(g_1)}{H(g_2)} \quad \text{and} \quad \frac{H(g_2)}{H(g_0)}.$$

  Select the subinterval where the increment or ratio is larger, and repeat the process by bisecting this subinterval. Iterating this scheme yields an increasingly precise localization of the zero.

- Once the approximate position of a root is identified, refine it using a standard root–finding algorithm such as Newton's method in order to achieve higher accuracy.

6.3.5 Possible problems and limitations

- The complexity of the method might depend on the parameters a and b, and on the cost of evaluating the expressions

$$H_{a,b}(x), \quad H_{a,b}(x_1)/H_{a,b}(x_0),$$

  which are used in the search procedure and the integration might be challenging.

- The behavior of the function $f(x)$ is important for the determination of complexity. For example, in the case

$$f(x) = \sin\left(\tfrac{\pi x}{2}\right) + \sin^2(4\pi x) + 0.1,$$

the problem of *false zeros* may arise: a false zero is a point where $f(x)$ is very small but not equal to zero. Since the modulation cannot distinguish a true zero from such values, it becomes necessary to choose a large a, which increases the computational burden and may limit the efficiency of the algorithm.

Remark. *The variant of transformation*

$$h_d(x) := \frac{a}{\sqrt{\pi}} \exp\left(-af(x)^2\right)$$

can be viewed as a delta–like approximation.

Indeed, on of the representations of the Dirac delta is

$$\delta(x) = \lim_{a\to\infty} \frac{a}{\sqrt{\pi}} \exp\left(-ax^2\right),$$

which is the form used in our construction ***[Piela 2013, Appendix E]****.*

Therefore, $h_d(x)$ can be interpreted as a combination with the Dirac delta function, applied to $f(x)$ (at least in the case where $f(x)$ contains only simple zeros). The peaks of $h_d(x)$ approximate the distribution $\delta(f(x))$, highlighting the zeros of f.

7 Factorization

7.1 Current state of art

To begin, we briefly review the problem of integer factorization, that is, finding the integer factors of the integer n. First and foremost, we are working with guaranteed algorithms for the general case and the worst-case conditions, so special-purpose algorithms are excluded. Secondly, the complexity of these algorithms depends on the input data, specifically not on the integer n, but on size$(n) \approx \log(n)$. Therefore, the naive algorithm of testing all integers from 2 to $\sqrt{n}$ is not polynomial and the number of iterations is $O(n^{\frac{1}{2}})$. First, we will give a brief account of Fermat's factorization method, which is a rather old and widely known approach. Moreover, the algorithm considered the fastest — the General Number Field Sieve — and the one considered the third-fastest, the Quadratic Sieve, are, in a broad sense, its modifications. We will discuss them afterwards. We will avoid the second one based on the elliptic curves theory and non-derministic ones. It is also important to note that in many cases, the complexity estimates are heuristic in nature, meaning they are not exact [**C. Pomerance 1996**]. We will give only a brief overview of these algorithms to provide a concise insight into the current state of the art.

7.1.1 Fermat's factorization Method

Fermat's factorization method is based on the representation of an odd integer n as the difference of two squares:

$$n = x^2 - y^2 = (x + y)(x - y)$$

A number n is a **perfect square** if there exists an integer m such that $n = m^2$. The goal is to find integers x and y such that $x^2 - n = y^2$. The method proceeds as follows:

Algorithm 3 Fermat's Factorization Method

Require: Integer n (odd and composite)
Ensure: A non-trivial factorization of n
  $x \leftarrow \lceil \sqrt{n} \rceil$
  $r \leftarrow x^2 - n$
  while r is not a perfect square **do**
    $x \leftarrow x + 1$
    $r \leftarrow x^2 - n$
  end while
  $y \leftarrow \sqrt{r}$
  return $(x + y, x - y)$

The efficiency of Fermat's method depends on how quickly r becomes a perfect square. In the worst case, x can increase up to $\frac{n+1}{2}$ before r is a perfect square. Therefore, in the worst-case scenario, i.e., when the divisors are far apart, the Fermat factorization method is even more costly than the naive brute-force algorithm [**Crandall and Carl Pomerance 2005, Chapter 5.1**].

7.1.2 Dixon's factorization method

Dixon's algorithm is a generalization of Fermat's factorization method. While Fermat's method seeks integers x and y such that $x^2 - y^2 = n$, Dixon's method extends this idea by finding integers where the difference of squares is congruent modulo n.

The key concept is to find several integers x_i and y_i such that:

$$x_i^2 \equiv y_i^2 \mod n.$$

This implies that $(x_i - y_i)(x_i + y_i)$ is a multiple of n, potentially revealing a non-trivial factor of n when computing the greatest common divisor. We call a number B-**smooth** if all of its prime factors are less than or equal to B.

Algorithm 4 Dixon's Factorization Algorithm

Require: An odd composite integer n
Ensure: A non-trivial factor of n (or failure)

Select a factor base $\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$ of small primes up to bound B
Initialize an empty list of relations
while the number of collected relations $< k + 1$ **do**
    Choose a random integer x with $1 < x < n$
    Compute $r \leftarrow x^2 \bmod n$
    if r is B-smooth over $\mathcal{P}$ **then**
        Factor r over $\mathcal{P}$ and record the relation $(x, (e_1, \ldots, e_k))$
    end if
end while
Form the exponent matrix A whose rows are the exponent vectors $(e_1, \ldots, e_k)$ modulo 2
Solve the homogeneous linear system $A\mathbf{v} = 0$ over $\mathbb{Z}/2\mathbb{Z}$, seeking a non-zero solution $\mathbf{v}$
Let S be the set of indices corresponding to the nonzero entries of $\mathbf{v}$
Compute $x = \prod_{i \in S} x_i \bmod n$
Compute $y = \sqrt{\prod_{i \in S} r_i} \bmod n$ ▷ since exponents are even, this is an integer
Compute $d = \gcd(x - y,\, n)$
if $1 < d < n$ **then**
    return d
else
    return Failure; try with more relations or different random x
end if

Dixon's algorithm has a sub-exponential expected running time, specifically:

$$T_n = \exp\left(\sqrt{\ln n \ln \ln n}\right).$$

This is faster than Fermat's method for large integers because it leverages multiple congruences of squares rather than searching for a single pair (x, y). A detailed description of the algorithm can be found in [**Dixon 1981**].

7.1.3 Quadratic sieve algorithm

The Quadratic Sieve (QS) is a factorization algorithm that generalizes and optimizes Dixon's method. Like Dixon's algorithm, the QS seeks to find congruences of squares modulo the integer n to be factored. However, it introduces efficient sieving techniques to locate suitable numbers more rapidly and Focused Search: it targets values of x near $\sqrt{n}$, where $Q(x)$ is more likely to be small and hence B-smooth.

The core idea is to find a set of integers x_i such that the values $x_i^2 \bmod n$ are smooth over a chosen factor base—meaning they factor completely over a small set of primes. The steps are as follows:

Algorithm 5 Quadratic Sieve Algorithm (outline)

Require: Odd composite integer n
Ensure: Non-trivial factor of n (or failure)

Choose a factor base $\mathcal{P} = \{p_1, \ldots, p_k\}$ of small primes $\leq B$
Select a sequence of integers x close to $\sqrt{n}$
for each x in the chosen sequence **do**
    $Q(x) \leftarrow x^2 - n$
end for
Use a sieve to identify x such that $Q(x)$ is B-smooth over $\mathcal{P}$
for each such x **do**
    Factor $Q(x)$ over $\mathcal{P}$ and record exponents
end for
Form a matrix A from the exponent vectors modulo 2
Find a non-zero vector $\mathbf{v}$ such that $A\mathbf{v} = 0$ over $\mathbb{Z}/2\mathbb{Z}$
Let S be indices where $v_i = 1$
$X \leftarrow \prod_{i \in S} x_i \bmod n$
$Y \leftarrow \sqrt{\prod_{i \in S} Q(x_i)} \bmod n$
$d \leftarrow \gcd(X - Y, n)$
if $1 < d < n$ **then**
    return d
else
    return Failure
end if

The expected running time of the Quadratic Sieve is sub-exponential:

$$T_n = \exp\left(\left(\sqrt{2} + o(1)\right)\sqrt{\ln n \ln\ln n}\right).$$

This complexity measure is based on heuristic assumptions about the distribution of smooth numbers—integers that factor completely over a small set of primes (the factor base). Specifically, the algorithm's efficiency relies on the probability of randomly selected numbers near $\sqrt{n}$ being B-smooth. In practice, the Quadratic Sieve performs close to the estimated running time for numbers up to around 100 digits. For a full description of the algorithm, see [**Bläser and Saha 2012**].

7.1.4 General number field sieve (GNFS)

The General Number Field Sieve is an extension of the Quadratic Sieve algorithm and is currently the fastest known classical algorithm for factoring large integers of general form. GNFS generalizes the ideas of the Quadratic Sieve by utilizing algebraic number fields to find relations that lead to the factorization of the target integer.

The main idea of GNFS is to find a large set of relations involving smooth numbers in a number field, which can be combined to produce a congruence of squares modulo the integer n to be factored.

Algorithm 6 Number Field Sieve (Outline)

Require: Odd composite integer n
Ensure: Non-trivial factor of n (or failure)

Polynomial Selection:
Choose an irreducible polynomial $f(x)$ with integer coefficients having a root modulo n, which defines the number field $\mathbb{Q}[x]/(f(x))$
Factor Base Selection:
Select a factor base of small primes and prime ideals in the number field
Relation Collection:
for pairs of integers (a, b) in a chosen range **do**
    Compute $a - bm$ and the norm $N(a - bx)$
    if both $a - bm$ and $N(a - bx)$ are smooth over the factor base **then**
        Record the relation (a, b)
    end if
end for
Matrix Construction:
Build a sparse matrix over $\mathbb{Z}/2\mathbb{Z}$ from exponents in the collected relations
Linear Algebra:
Find nontrivial dependencies in the matrix (e.g., using Block Lanczos or Wiedemann algorithm)
Square Root Computation:
Combine corresponding relations to compute $x^2 \equiv y^2 \pmod{n}$
Factorization:
Compute $d \leftarrow \gcd(x - y, n)$
if $1 < d < n$ **then**
    return d
else
    return Failure
end if

The expected asymptotic running time of GNFS is sub-exponential and given by:

$$T_n = \exp\left(\left(\left(\frac{64}{9}\right)^{1/3} + o(1)\right)(\ln n)^{1/3}(\ln \ln n)^{2/3}\right).$$

For a detailed description, see [**Buhler, Jr., and Carl Pomerance 2006, Chapter 1 and 2**].

7.2 Reformulation of factorization

Now, let us introduce the **Factor function**

$$f(x) := \sin^2(\frac{\pi t}{x}) + \sin^2(\pi x),$$

where t is a natural number. The function has an interesting property:

Theorem 1. *On the interval* $[2,t]$ *the function* $f(x)$ *equals zero if and only if* $x \mid t$

Proof. Proof is based on 3 simple facts:

- All summands and the function is nonnegative
- $\sin^2(\frac{\pi t}{x}) = 0$ if and only if $\frac{t}{x}$ is a natural number
- $\sin^2(\pi x) = 0$ if and only if x is a natural number

□

A similar function has been already observed and earlier in the [**Mateus and Vieira 2011**]. We will write a slightly different function and expand this original function into a family of functions with similar properties.

Mentioned function has essential singularity in 0 and analytic everywhere else. It has conjugate complex zeros. Moreover, the function and its one-dimensional versions below are holonomic.

Theorem (Holonomicity)**.** *Factor function* $f(x) := \sin^2(\frac{\pi t}{x}) + \sin^2(\pi x)$ *is holonomic.*

Proof. Use closure properties form section 5.3. $\sin(x)$ is holonomic. Composition of holonomic function with rational functions $s(x) = x\pi$ and $s(x) = \frac{\pi t}{c}$ is again holonomic. The product $\sin(s(x))\sin(s(x)) = \sin^2(s(x))$ is holonomic. The sum of two holonomic $\sin^2(\frac{\pi t}{x}) + \sin^2(\pi x)$ is again holonomic. □

For other functions it can be proven analogously, taking into account the fact, that cosine is also holonomic.

It can be also formulated as maxima version: the function, which has maxima equal 2 if and only if x divides t:

$$f_M(x) := \cos^2(\frac{\pi t}{x}) + \cos^2(\pi x).$$

It can be proved the same way as in theorem 1:

- All the summands and the function is nonegative and all the suumands are equal or less than 1
- $\cos^2(\frac{\pi t}{x}) = 1$ if and only if $\frac{t}{x}$ is a natural number
- $\cos^2(\pi x) = 1$ if and only if x is a natural number

Moreover, the function $f_M(x)$ is also holonomic, since $\cos(x)$ is holonomic and hence this fact can be proven the same way as in the theorem of holonomicity. The two dimensional versions of the functions also might be introduced:

$$f(x,y) := (t - xy)^2 + \sin^2(\pi x) + \sin^2(\pi y).$$

It has the advantage, since it is entire.

The function equals 0 on the square $[2, t-1]$, $[2, t-1]$ if and only if a pair of natural x, y divides t, since:

- All summands are nonnegative
- $(t - xy)^2 = 0$ if and only if $xy = t$
- $\sin^2(\pi x) = \sin^2(\pi y) = 0$ if and only if x and y are natural numbers

7.2.1 Complexity estimation

Now we want to compute an approximation of the $f(x)$ with the precision n, $size(t), size(x) \leq n$ and excluding point 0. All the approximations of the onedimensional functions could be computed in polynomial time the following way:

1. **Approximation of the constants.** Compute an approximation of π with sufficient precision n plus some extra digits. This can be done in $O(M(n) \log n)$.
2. **Argument reduction.** Evaluate the expressions $\frac{\pi t}{x}$ and $2\pi x$, denoting these as xs_1 and xs_2, respectively. This step requires a number of operations bounded by $M(\text{size}(x))$, where $M(n)$ denotes the complexity of multiplication of n-bit integers. Apply standard argument reduction techniques to xs_1 and xs_2 to map them into a small neighborhood of the origin, into an interval $[-\pi/4, \pi/4]$, leveraging the periodicity and parity properties of the sine function $M(n)$.
3. **Evaluation of the sine function.** Compute $\sin(xs_1)$ and $\sin(xs_2)$. Suitable algorithms include the Arithmetic-Geometric Mean (AGM) method or bit-burst algorithm for holonomic functions, since $f(x)$ is holonomic, both of which allow evaluation to a given precision in quasi-linear time in the number of digits $O(M(n) \log n)$ in the case of AGM as in [**Richard P. Brent 1975, Section 12**], and $O(M(n) \log^3 n)$ in the case of general bit-burst algorithm.
4. **Post-processing.** Perform the squaring operation to obtain $\sin^2(xs_1)$ as well as $\sin^2(xs_2)$ via single multiplications, and compute the sum $\sin^2(xs_1) + \sin^2(xs_2)$ to finalize the evaluation of $f(x)$. The cost of this step is $M(n)$.

So, the overall complexity of the method is $O(M(n) \log n)$, using Arithmetic-geometric mean algorithm for step 3, with obviously a larger constant, hidden behind big-O, than in the upper steps.

7.2.2 Application of root finding

We have described the problem of finding the divisors of a number as the problem of finding the zeros of a function. Now, our task is to find the zeros of the function $f(x)$. Since the classical algorithms as binary search or Newton's methods require good initial

approximation for good result, we need something else. An obvious way to find the zeros of this type of functions, also considered in [**Mateus and Vieira 2011, Formula 11**], since the function $f(x)$ is analytic on the interval $[1, t]$ is either to compute the residue of the function $h = \frac{1}{f(t)}$ using complex integration or to apply the argument principle, which also requires integration. The article [**Mateus and Vieira 2011**] concludes that the complexity of complex integration of the function $f(z)$ is equivalent to the complexity of finding the divisors of the number t.

We have to give our perspective of the approaches using complex integration and say that it seems impossible to guarantee it polynomial-time computability. To demonstrate this, we employ the following heuristic arguments.

7.2.2.1 Heuristic arguments

First we need to see a fact about the complex zeros of the $f(x) = \sin^2(\frac{\pi t}{x}) + \sin^2(\pi x)$, which will be poles of $\frac{1}{f(z)}$ or $\frac{f`(z)}{f(z)}$.

The function has a large number of complex zeros, so one faces the task of bypassing them in such a way that **only the real roots lie inside the contour**. For the function used in the article [**Mateus and Vieira 2011**],

$$f_{MaV}(z) = 1 - \cos\left(\frac{\pi t}{z}\right)\cos(\pi z),$$

the following estimate for regions free from complex zeros holds:

Let μ be the smallest positive local minimum of $f(x)$ in the interval $(\sqrt{n}, n)$. Then $f(x + iy)$ has no zeros whenever $x \in (\sqrt{t}, t)$, $f_{MaV}(x) \geq \mu$ restricted to the real line and $-k < y < k$ with

$$k = \frac{1}{2\pi}\ln\left(1 + \frac{\mu}{2}\right)$$

[**Mateus and Vieira 2011, Theorem 2**].

Since how close the complex poles are to the real axis—and therefore how close the contour must pass to the real axis to avoid them—depends on the minimum of the function on the real axis, it is difficult, or even impossible, to provide a strict lower bound. Consequently, a pole can lie arbitrarily close to the real axis, as in the case of the function from the article $f_{MaV}(z)$. For our function $f(z)$, for which no such direct estimate is available, the situation may be even worse.

Moreover, since the modulus of the function tends to infinity as z approaches a pole, the constant in the estimate of the number of nodes grows rapidly. This might make the number of nodes non-polynomial with respect to size(t). The closer a pole (arising from a complex zero) lies to the real axis, the closer the contour must also pass to the real axis and the pole itself, thereby increasing the contribution of the constant caused by the pole.

The second difficulty concerns the constant M, considered in the context of the function alone, without accounting for the effect of poles. This constant might require an exponential number of points to be "neutralized," as discussed in Section 6.2, because sine

functions are defined via the exponential function, which grows at an enormous rate in the complex domain. Furthermore, since we want to make the contour as large as possible to perform a binary search, the constant M becomes even larger.

Because methods relying on complex integration around singular points seem inevitably subject to such issues, it is preferable to employ alternative techniques.

7.3 Application of proposed method and analytic reduction

7.3.1 Resulting functions

We must state that the direct application of the above methods does not yield polynomial complexity for factorization, but delivers another interesting result.

We observe the functions

$$g(f(x)) := \operatorname{erfc}\left(a\left(\sin^2\left(\frac{\pi t}{x}\right) + \sin^2(\pi x)\right)\right), \tag{7.1}$$

and

$$h_a(x) := \left(\operatorname{erfc}\left(a\left(\sin^2\left(\frac{\pi t}{x}\right) + \sin^2(\pi x)\right)\right) + c\right)^b \tag{7.2}$$

or

$$h_d(x) := \frac{a}{\sqrt{\pi}} \exp - \left(a\left(\sin^2\left(\frac{\pi t}{x}\right) + \sin^2(\pi x)\right)\right), \tag{7.3}$$

$$H_a(x) := \int_2^x \left(\operatorname{erfc}\left(a\left(\sin^2\left(\frac{\pi t}{\xi}\right) + \sin^2(\pi \xi)\right)\right) + c\right)^b d\xi, \tag{7.4}$$

$$H_d(x) := \int_2^x \frac{a}{\sqrt{\pi}} \exp - \left(a\left(\sin^2\left(\frac{\pi t}{\xi}\right) + \sin^2(\pi \xi)\right)\right) d\xi. \tag{7.5}$$

7.3.2 Analytic reduction and general remarks on computation

Practically, using the proposed method, **we can reduce the problem of factorization to the problem of computing certain analytic functions**. We will call this **analytic reduction**. Nevertheless, even if we could compute $H(x)$ with precision up to n bits in polynomial time for any values a and b, we still can say little about the complexity of computing the divisors t using this method, since, since the precision needed for the computation is not known.

Since the functions $g(f(x))$ are chosen in such a way that the values approach 1 as $f(x) \to 0$, the larger the value of $f(x)$, the closer $g(f(x))$ is to 0. Consequently, the choice

of the parameter a clearly depends on the "false zeros" - the values of x for which $f(x)$ is sufficiently close to zero. Accordingly, the parameter a must be chosen so as to "lift" the minima that do not correspond to actual zeros, so that they are not counted as zeros.

Next, one should estimate the values of $g(f(x))$ both far from the zeros and near the zeros; after that, the parameter b must be introduced, and the estimation performed again for the function $h_a(x)$. These steps make it possible to determine the precision required for the binary search to work.

However, the computation of the function $H(x)$ itself is a rather nontrivial task, and in what follows we will present some arguments explaining why the direct application of the methods discussed earlier to this function does not yield its computation in polynomial time.

7.3.3 AGM and numerical integration

The arithmetic–geometric mean method is based on the functional equations discussed in Chapter 4. This method works well for the functions

$$f(x) := \sin^2\left(\frac{\pi t}{x}\right) + \sin^2(\pi x), \qquad f(x) := \cos^2\left(\frac{\pi t}{x}\right) + \cos^2(\pi x),$$

but the integrals discussed above are not suitable for this type of method, since the corresponding functional equations do not apply in this case.

Classical numerical integration also appears to be unsuitable, as it may require an exponential number of nodes due to the constant M, as discussed in Section 6. More detailed estimation will be shown below.

7.3.4 Algorithm for holonomic functions

Let us discuss application of the bit-burst (or general holonomic) algorithm. SSince compositions of holonomic functions are not holonomic in the general case, we cannot apply the algorithm to $g(x)$ and $h(x)$ including $\operatorname{erfc}(x)$ or e^x. Nevertheless we have at least one holonomic candidate for $g(f(x))$:

$$g(f(x)) := (1/2(\cos^2(\frac{\pi t}{x}) + \cos^2(\pi x)))^a \tag{7.6}$$

The function essentially performs the same task as the proposed method for locating zeros. We know that its maxima correspond to the divisors of the number in the interval from 2 to t, and that the maximum value of the function is 2. Therefore, by dividing the function by 2, we obtain a function that is positive on the entire interval and whose maxima are equal to 1. Raising it to a positive power, we observe that all points of the graph, except the zeros and their neighborhoods, are shifted downward.

To see the holonomicity of the function, we use the fact that both the integral and the product of holonomic functions remain holonomic, in other words, integers powers of

holonomic functions are holonomic. Hence, since $f_M(x)$ is holonomic, its product with constant is holonomic, next, its power with integer exponent is holonomic since it is repetitive multiplication and finally, the integral of the result is holonomic. The initial point for the algorithm is the function given as differential equation. The functions $f(x) := \sin^2(\frac{\pi t}{x}) + \sin^2(\pi x)$ and $f(x) := \cos^2(\frac{\pi t}{x}) + \cos^2(\pi x)$ satisfy the differential equation:

$$
\begin{aligned}
&(2\pi^4t^4x^4 + 15\pi^2t^2x^6 + 9x^8 - 4\pi^4t^2x^8 + 15\pi^2x^{10} + 2\pi^4x^{12})f^{(5)} \\
&+ (28\pi^4t^4x^3 + 180\pi^2t^2x^5 + 90x^7 - 40\pi^4t^2x^7 + 120\pi^2x^9 + 12\pi^4x^{11})f^{(4)} \\
&+ (8\pi^6t^6 + 168\pi^4t^4x^2 + 540\pi^2t^2x^4 - 8\pi^6t^4x^4 + 180x^6 + 160\pi^4t^2x^6 + 180\pi^2x^8 \\
&\quad - 8\pi^6t^2x^8 + 72\pi^4x^{10} + 8\pi^6x^{12})f^{(3)} \\
&+ (112\pi^6t^4x^3 + 720\pi^4t^2x^5 + 360\pi^2x^7 - 160\pi^6t^2x^7 + 480\pi^4x^9 + 48\pi^6x^{11})f^{(2)} \\
&+ (32\pi^8t^6 + 672\pi^6t^4x^2 + 2160\pi^4t^2x^4 - 64\pi^8t^4x^4 + 720\pi^2x^6 + 400\pi^6t^2x^6 + 576\pi^4x^8 \\
&\quad + 32\pi^8t^2x^8 + 48\pi^6x^{10})f^{(1)} = 0.
\end{aligned}
$$

The equation is not simple, but for the equation, for which $g(f(x)) := (1/2(\cos^2(\frac{\pi t}{x}) + \cos^2(\pi x)))^a$ with even integer $a > 1$ satisfy as solution we must say that the coefficients become unacceptably large, making the function not usable for the determination of the factors, because the computation of the matrix (5.6) becomes difficult. Moreover, an additional source of complexity for this function lies in the need to evaluate $\frac{1}{2}$ raised to the power a before proceeding to the evaluation of the function itself, in the case of a direct computation without employing any simplification techniques. The **holonomic functions algorithm from the chapter 5.3 is not appropriate.**

7.3.5 Direct application of Taylor series and constant M

The direct Taylor series computation computation becomes also useless. To show this we will estimate the the number of Taylor series decomposition members m as in 5.1. Note that every member of the decomposition should be computed before the summation and therefore the exponential number of the members of decomposition lead to the at least superpolnmial complexity.

7.3.5.1 The constant M

The Taylor's theorem requires the estimation of the derivatives, which is hard to get, so we will use the method for the complex plane. Our goal is to give an upper bound for the constant M_R, which depends on the radius R. Note, that want to use binary splitting algorithm and need to make the radius large and we have to avoid essential singularity in the origin.

First, we will do it for the function $f(x)$. Apply basic trigonometric identities we get:

$$\sin^2(z) = \frac{1 - \cos(2z)}{2}, \quad \cos(z) = \frac{e^{iz} + e^{-iz}}{2}, \quad z = x + iy,$$

$$\sin^2(z) = \frac{2 - e^{2iz} - e^{-2iz}}{4} = \frac{2 - e^{i2(x+iy)} - e^{-i2(x+iy)}}{4} = \frac{2 - e^{i2x}e^{-2y} - e^{-i2x}e^{2y}}{4}.$$

Hence

$$|\sin^2(z)| = \left|\frac{2 - e^{i2x}e^{-2y} - e^{-i2x}e^{2y}}{4}\right| \leq \frac{2 + e^{-2y} + e^{2y}}{4}. \tag{S}$$

We can estimate the absolute value of the $\sin^2(\theta)$ using **maximum modulus principle** on the disc with the radius R with the center in x_0 letting all the "length" of the radius go to the complex plane as $\sin^2(x_0 + Ri)$ estimating $M_{1,R} = e^{-R}$ and $M_{2,R} = e^R$ and combining it we get

$$\left|\sin^2(\pi(x_0 + Ri))\right| \leq \frac{2 + e^{-2\pi R} + e^{2\pi R}}{4}.$$

For the next summand we will use trivial estimation because the behavior of the function is somewhat more complicated, but later we will show that consideration only the first summand also gives superpolynomial complexity. Note, that $x_0 > R$, $R > 1$ to perform binary search (will be explained later), $x_0 \geq 2$ and $x_0 - R \geq 1$

$$\left|\frac{\pi t}{x_0 + Re^{i\phi}}\right| \leq \pi t,$$

where ϕ is an angle, chosen to make the summand maximal, next

$$\left|\sin^2(\pi(\frac{t}{x_0 + Re^{i\phi}}))\right| \leq \frac{2 + e^{-2\pi t} + e^{2\pi t}}{4} \leq \frac{3 + e^{2\pi t}}{4}$$

and hence

$$|f(z)| = |\sin^2(\frac{\pi t}{z}) + \sin^2(\pi z)| \leq \frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4.}$$

$$M_{f,R} = \frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4}.$$

To compute the constant for $g(x)$, $h(x)$ and $H(x)$ we need to insert the previous result in the functions erfc(z) or e^{-z^2}, but since the function is non-negative, we omit the square. For the second type of functions we easily see, that the absolute value of the function increases exponentially in the imaginary domain, i.e.,

$$|h_d(z)| = |\frac{a}{\sqrt{\pi}}e^{-af(z)}| \leq \frac{a}{\sqrt{\pi}}e^{aM_{f,R}}.$$

For (complementary) error function we have to do some additional work. Note that in the case of transitioning from the function $g(x)$ to the function $h(x)$, we merely need to raise the constant to a power b (in the case of $h_a(x)$), however, for the sake of simplicity, we will assume that $g(x)$ and $h(x)$ are equal.

Take the definition from the section 6.3

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-l^2}\, dl$$

and

$$\operatorname{erfc}(z) = 1 - \operatorname{erf}(z).$$

But again, since the function $f(x) \geq 0$, we omit the square and define a new function as

$$\operatorname{erfn}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-\ell}\, d\ell, \qquad \operatorname{erfcn}(z) = 1 - \operatorname{erfn}(z).$$

To achieve the estimation, we apply the **ML inequality** (also known as the **Estimation Lemma**):

Theorem (ML inequality)**.** *If f is continuous on the contour Γ and if $|f(z)| \leq M$ for all $z \in \Gamma$, then*

$$\left| \int_\Gamma f(z)\, dz \right| \leq M\, \ell(\Gamma),$$

where $\ell(\Gamma)$ denotes the length of Γ.

In particular, we have

$$\left| \int_\Gamma f(z)\, dz \right| \leq \max_{z \in \Gamma} |f(z)| \cdot \ell(\Gamma).$$

Proof. Can be found at [**Saff and Snider 2003, Chapter 4.2**]. □

Since we study the function inside the circle of analyticity, the integral is path independent and we can choose the minimal possible path, which in our case is R. It should be noted that this estimate is fairly coarse.

Here and below we redefine the functions h_a from (7.2) and H_a from (7.4) using erfcn **instead of** erfc**.**

Applying it to the result we got above for $f(x)$, we get

$$|\operatorname{erfcn}(M_{f,R})| = \left| 1 - \frac{2}{\sqrt{\pi}} e^{-aM_{f,R}} \right| R \leq \left(1 + \frac{2}{\sqrt{\pi}} e^{aM_{f,R}} \right) R = M_{h_a,R}.$$

This provides an estimate for h_a and h_d,

Our next step is to transfer these results to H_a and H_d. We use the estimation lemma again and get

$$M_{H_a,R} = \left(1 + \frac{2}{\sqrt{\pi}} e^{M_{f,R}} \right) R^2$$

and

$$M_{H_d,R} = \frac{a}{\sqrt{\pi}} e^{aM_{f,R}} R.$$

7.3.5.2 Number of members of decomposition m

Recall (5.2) and (5.3):

$$\tau = \frac{R}{|z - z_0|}, \quad C_1 = \frac{\log 2}{\log \tau}, \quad C_2 = \frac{\log(2M_{H,R}) - \log(1 - \tau^{-1})}{\log \tau}$$

$$C_1 = \log_\tau(2), C_2 = \log_\tau(\frac{2M_{H,R}}{1 - \tau^{-1}})$$

combine it with (5.4)

$$m \geq C_1 n + C_2$$

We get

$$m \geq \frac{\log(2)n + \log(2M_{H,R}) - \log(1 - \frac{|z-z_0|}{R})}{\log R - \log(|z - z_0|)}$$

for $m_{H_{a,R}}$

$$m_{H_{a,R}} \geq \frac{\log(2)n + \log(2\left(\left(1 + \frac{2}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4}\right) R^2\right)) - \log(1 - \frac{|z-z_0|}{R})}{\log R - \log(|z - z_0|)}$$

Note, that

$$1 + \frac{2}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4} \geq e^{ae^1} > 1 \tag{7.7}$$

Therefore

$$1 + \frac{2}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4} \leq \frac{3}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4}$$

$$\Rightarrow \log\left(1 + \frac{2}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4}\right) \leq$$

$$\leq \log\left(\frac{3}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^{2\pi R} + e^{2\pi t}}{4}\right)$$

Therefore the following estimation for $m_{H_{a,R}}$ for the worst case scenario works

$$m_{H_{a,R}} \geq \frac{\log(2)n + \log(2\left(\left(\frac{3}{\sqrt{\pi}} \exp a\frac{5 + e^{-2\pi R} + e^2\pi R + e^{2\pi t}}{4}\right) R^2\right)) - \log(1 - \frac{|z-z_0|}{R})}{\log R - \log(|z - z_0|)}$$

$$m_{H_{a,R}} \geq \frac{n\log 2 + \log 2 + \log\left(\frac{3}{\sqrt{\pi}}\right) + a\frac{5 + e^{2-\pi R} + e^{2\pi R} + e^{2\pi t}}{4} + 2\log R - \log\left(1 - \frac{|z-z_0|}{R}\right)}{\log R - \log(|z - z_0|)}.$$

Next, we estimate $m_{H_{d,R}}$ as

$$m_{H_{d,R}} \geq \frac{\log(2)n + \log\left(2\left(\frac{a}{\sqrt{\pi}}\exp a\frac{5+e^{-2\pi R}+e^{2\pi R}+e^{2\pi t}}{4}\right)\right) - \log(1-\frac{|z-z_0|}{R})}{\log R - \log(|z-z_0|)}$$

$$m_{H_{d,R}} \geq \frac{n\log 2 \ + \ \log 2 \ + \ \log\left(\frac{a}{\sqrt{\pi}}\right) \ + \ a\frac{5+e^{-2\pi R}+e^{2\pi R}+e^{2\pi t}}{4} \ + \ \log R \ - \ \log\left(1-\frac{|z-z_0|}{R}\right)}{\log R - \log(|z-z_0|)}.$$

7.3.5.3 Setting the radius R and step size

Although one might argue that the estimate involving $\frac{3+e^{2\pi t}}{4}$ is too crude, we shall see later that even in the case of a lower bound, after discarding it, the situation in this model remains computationally challenging.

Next, we will show that in our model it is not possible to guarantee the polynomiality of m, and consequently of the entire computation using the Taylor series. The reason for this, briefly speaking, is that if we want to use binary search, we have to evaluate the function $H_a(x)$ or $H_d(x)$ on a large interval, which increases the cost of each individual computation. On the other hand, if we want simple computations, we need small intervals, but then there will be too many of them.

Let us clarify that in our model the initial points $z_0 = R_0$ lie on the real line. We impose the condition

$$R > |z - z_0| > 1,$$

since we want to use the evaluation of the function $H(x)$ for the binary search. For the first step of binary search between 1 and $\sqrt{t}$ we need to compute $H(2)$, $H\left(\lfloor\sqrt{t}\rfloor\right)$, $H\left(\frac{\lfloor\sqrt{t}\rfloor-2}{2}\right)$. Accordingly, $|z - z_0|$ must be sufficiently large so that the number of steps does not exceed a polynomial bound with respect to the length of t. We also assume, that t is semiprime, in other words, there is only one prime divisor between 1 and $\sqrt{t}$.

Take the above formulas for $m_{H_{a,R}}$ and $m_{H_{d,R}}$. We separate the fraction and we set

$$m_1 = \frac{\log\left(1-\frac{|z-z_0|}{R}\right)}{\log\left(\frac{R}{|z-z_0|}\right)}, \qquad m_2 = \frac{c_1\log R}{\log\left(\frac{R}{|z-z_0|}\right)},$$

with $c_1 = 2$ for $m_{H_{a,R}}$ and $c_1 = 1$ for $m_{H_{d,R}}$,

$$m_3 = \frac{n\log 2 \ + \ \log 2 \ + \ \log\left(\frac{C}{\sqrt{\pi}}\right)}{\log\left(\frac{R}{|z-z_0|}\right)}, \qquad m_4 = \frac{a\frac{5+e^{-\pi R}+e^{\pi R}+e^{2\pi t}}{4}}{\log\left(\frac{R}{|z-z_0|}\right)},$$

with $C = a$ for $m_{H_{d,R}}$ and $C = 3$ for $m_{H_{a,R}}$, and hence

$$m \geq -m_1 + m_2 + m_3 + m_4$$

For $R > |z - z_0| > 1$:

m_1 is always negative, and since it is subtracted in the remaining part of the formula for m, this only increases the resulting value of m.

m_2 is always positive.

m_3 is always positive.

m_4 can be estimated as

$$\frac{a}{4}\frac{e^{\pi R}}{\log R} \leq m_4 \leq e^{c_2 R} + e^{c_3 t}$$

$c_2 > 1$, since minimal $|z - z_0| = 1$ and we want to have maximal denominator for minimal expression.

Now we show that $R > c_0 t^{1/k}$. We can choose some initial point near $\sqrt{t}$, t, or 1. In order to guarantee that a root lies within the region under consideration during the binary search, we must either make R sufficiently large or cover it with small overlapping regions. On the other hand, the number of such regions must be at most polynomial with respect to the length of t.

$$d = \text{size}(t) \approx \log(t), \qquad 1 \leq \text{Number of steps} \leq k_1 (\log t)^{\ell}$$

so that the algorithm does not require a superpolynomial number of computations. In the first case, with large R, we obtain $R \geq k_2(t^{1/2})$; in the second case, we have do estimate **Average step size** as

$$\left(\left\lfloor \sqrt{t} \right\rfloor - 2\right) / (\text{Number of steps}) \approx c_4 \sqrt{t} / (\text{Number of steps})$$

even if we take different sized R_i, $i \in \mathbb{N}$ at least one of them must satisfy

$$R_{i_1} \geq \text{Average step size}$$

(pigeonhole principle), since the average step must be of size , that is

$$\text{Average step size} \geq \frac{c_4 t^{1/2}}{(\log t)^{\ell}},$$

we can bound this from below as

$$\text{Average step size} \geq \frac{c_4 t^{1/2}}{(\log t)^{\ell}} \geq c_0 t^{1/k}, \qquad 2 < k.$$

so that the number of steps in the binary search is not superpolynomial. Therefore, at least one of R has to be greater then $c_0 t^{1/k}$.

We get

$$\frac{a}{4}\frac{e^{\pi c_0 t^{1/k}}}{\log c_0 t^{1/k}} \leq m_4$$

what is superpolynomial growth with respect to the t, and since $t \approx 2^d$, the growth order with respect to d might be greater, but again greater than polynomial. Therefore m is also superpolynomial and since every step must be evaluated during Taylor series computation, the total number of steps in the algorithm superpolynomial.

7.3.6 Application of quadrature methods

Now we will consider the application of numerical quadrature, using the methods from Chapter 6.2 and the estimates obtained above. First of all, we must compress the function in order to place it into the interval $[-1, 1]$, to apply theorems from section 6.

$$\int_{-v}^{v} g\big(f(x)\big)\,dx = \int_{-1}^{1} g\big(avf(vx)\big)\,dx$$

Next, we must shift the beginning of this interval in order to avoid an essential singularity.

$$\int_{-1}^{1} g\Big(avf\big(v(x+u)\big)\Big)\,dx$$

We obtain for the functions h_a and h_d

$$H_{q,a} = \int_{-1}^{1} \operatorname{erfcn}\Big(av\big(\sin^2(\pi v(x+u)) + \sin^2\Big(\tfrac{\pi t}{v(x+u)}\Big)\big)\Big)\,dx$$

$$H_{q,d} = \int_{-1}^{1} \frac{a}{\sqrt{\pi}} \exp\Big(av\big(\sin^2(\pi v(x+u)) + \sin^2\Big(\tfrac{\pi t}{v(x+u)}\Big)\big)\Big)\,dx$$

Since v is essentially the boundary of the domain of integration, v must satisfy the same size conditions as R in order to satisfy the step-size requirement for the binary search. Therefore, we can split the integration interval of the function $h_v(x)$ into several segments $v_i \quad i \in \mathbb{N}$, and then at least one v_{i_0} is

$$v_{i_0} \geq c_0 t^{1/k}.$$

Now we have to estimate the constants M_{v,h_a} and M_{v,h_d} using previous estimations for $|\sin^2(\pi z)|$ and $|\sin^2(\frac{\pi t}{z})|$ considering (S) and taking into account that $|v(u+z)| > 1$ to follow the condition of step size for binary search, and more general following the conclusions of previous chapter about x_0 and R, where $u \in \mathbb{R}$ plays the role of x_0, and z is point on the ellipse instead of the radius R.

$$\big|\sin^2(\pi v(z+u))\big| \leq \left|\frac{2 + e^{-2\pi vz} + e^{2\pi vz}}{4}\right|$$

$$\left|\sin^2\left(\pi\left(\frac{t}{v(u+z)}\right)\right)\right| \leq \frac{2 + e^{-2\pi t} + e^{2\pi t}}{2} \leq \frac{3 + e^{2\pi t}}{4}$$

and then using estimation for both $h_a(z)$ and $h_d(z)$ we obtain

$$M_{mv,h_a} = \text{Upper bound}_{z\in\text{ellipse}}\left(\left|2\left(1 + \frac{2}{\sqrt{\pi}}\exp av\frac{5 + e^{-2\pi vz} + e^{2\pi vz} + e^{2\pi t}}{4}\right)\right|\right)$$

$$M_{mv,h_d} = \text{Upper bound}_{z\in\text{ellipse}}\left(\left|\left(\frac{a}{\sqrt{\pi}}\exp av\frac{5 + e^{-2\pi vz} + e^{2\pi vz} + e^{2\pi t}}{4}\right)\right|\right)$$

Here the number 2 is chosen in estimation for M_{mv,h_a} because earlier this factor was obtained as a result of estimating the function erfcn using the ML inequality. In this case, the total length of the integration interval is also 2. Furthermore, since we are estimating the maximum on the ellipse, we can obtain the following bound.

$$M_{v,h_a} = 2\left(1 + \frac{2}{\sqrt{\pi}} \exp av \frac{5 + e^{-2\pi v(\rho - \rho^{-1})/2} + e^{2\pi v(\rho+\rho^{-1})/2} + e^{2\pi t}}{4}\right)$$

$$M_{v,h_d} = \left(\frac{a}{\sqrt{\pi}} \exp av \frac{5 + e^{-2\pi v(\rho - \rho^{-1})/2} + e^{2\pi v(\rho+\rho^{-1})/2} + e^{2\pi t}}{4}\right)$$

and

$$M_{mv,h_a} \le M_{v,h_a}, \qquad M_{mv,h_d} \le M_{v,h_d}$$

Recall formula 6.3

$$n \ge \frac{\log_\rho(\epsilon^{-1} \cdot M_{\rho,f} \cdot b)}{c}.$$

using (7.7) again we can estimate this as

$$n_{v,h_a} \ge \frac{\log_\rho(\epsilon^{-1}) + \log_\rho(\frac{3}{\sqrt{\pi}}) + av\frac{5+e^{-2\pi v(\rho-\rho^{-1})/2}+e^{2\pi v(\rho+\rho^{-1})/2}+e^{2\pi t}}{4} + \log_\rho(e) + \log_\rho(b)}{c},$$

$$n_{v,h_d} \ge \frac{\log_\rho(\epsilon^{-1}) + \log_\rho(\frac{a}{\sqrt{\pi}}) + av\frac{5+e^{-2\pi v(\rho-\rho^{-1})/2}+e^{2\pi v(\rho+\rho^{-1})/2}+e^{2\pi t}}{4} + \log_\rho(e) + \log_\rho(b)}{c}.$$

or

$$n_{v,h_a} \ge \frac{\ln(\epsilon^{-1}) + \ln(\frac{3}{\sqrt{\pi}}) + av\frac{5+e^{-2\pi v(\rho-\rho^{-1})/2}+e^{2\pi v(\rho+\rho^{-1})/2}+e^{2\pi t}}{4} + \ln(e) + \ln(b)}{\ln(\rho)c},$$

$$n_{v,h_d} \ge \frac{\ln(\epsilon^{-1}) + \ln(\frac{a}{\sqrt{\pi}}) + av\frac{5+e^{-2\pi v(\rho-\rho^{-1})/2}+e^{2\pi v(\rho+\rho^{-1})/2}+e^{2\pi t}}{4} + \ln(e) + \ln(b)}{\ln(\rho)c}.$$

This is exponential with respect to the v in this model, and to satisfy the condition on v we must replace it with $c_0 t^{1/k}$ in the formulas above, which once again results in superpolynomial complexity, both with respect to t and with respect to d.

8 Discussion

Some methods from computer arithmetic, algebra, and numerical analysis were considered in this work. Some important methods and topics were omitted, such as the method of continued fractions, Padé approximations, Hypergeometric functions, asymptotic expansions, quotient-difference algorithm and the topic of guard digits. The considered methods turned out to be unsuitable for the unambiguous and fast computation of the factor function. Nevertheless, the method from Section 6.3 opens up some new perspectives. Firstly, it can be used on its own; secondly, in the context of the factor function and its computation, there are three reasons for optimism. Let us go through them one by one.

First of all, it opens up the possibility of searching for functional equations, either for functions that are analytic everywhere (as in the two-dimensional case), or everywhere except for a single point. Secondly, the factor function is holonomic. There is a remarkable conjecture that zeros of the holonomic functions might be holonomic numbers and elements of the $P-$sequences [**Huang and Kauers 2018, Chapter 5**].

For the third point, we need to delve briefly into the theory of differential equations. If a function is called holonomic or D-finite when it satisfies a linear differential equation with polynomial coefficients, then the class of DD-finite functions is a generalization: these are functions satisfying linear differential equations with holonomic coefficients. One can also construct further generalizations, namely, the so-called D^n-finite functions. Two properties are the most important for our purposes are the following.

The composition of two functions f and g, both of which are D-finite, is itself D-finite. Furthermore, the composition of a D^m-finite function with a D^n-finite function is D^{m+n}-finite[**Jiménez-Pastor, Pillwein, and Singer 2020, Theorem**] .

In addition, the class of D^n-finite functions is closed under integration [**Jiménez-Pastor, Pillwein, and Singer 2020**].

Although the estimated complexity of computing $H(x)$ is superpolynomial, there remain grounds for optimism. Indeed, $h(x)$, $g(f(x))$, and consequently $H(x)$ are D^n-finite; therefore, $H(x)$ may inherit significant structural properties and possibly allow for efficient computational methods.

References

[1] A.Okhotin. *Introduction to Theoretical Computer Science: Lecture 1*. Lecture notes available online. Accessed: December 10, 2024. 2019. URL.

[2] Milton Abramowitz and Irene A. Stegun, eds. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Vol. 55. Applied Mathematics Series. Tenth Printing, December 1972, with corrections. Washington, D.C.: National Bureau of Standards, 1964.

[3] Lars V. Ahlfors. *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable*. 3rd. International Series in Pure and Applied Mathematics. Originally published in 1953, 2nd edition in 1966. New York: McGraw-Hill, 1979. ISBN: 0070006571.

[4] José L. Balcázar and Joaquim Gabarro. "Nonuniform complexity classes specified by lower and upper bounds". In: *Informatique théorique et applications* 23.2 [**1989**]. © AFCET, 1989. Numdam digital library., pp. 177–194. URL.

[5] Gianfranco Bilardi and Lorenzo De Stefani. "The I/O Complexity of Toom–Cook Integer Multiplication". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, Jan. 2019, pp. 2034–2052. DOI: 10.1137/1.9781611975482.123.

[6] Markus Bläser and Chandan Saha. *Lecture 20, Computational Number Theory and Algebra*. https://www.csa.iisc.ac.in/~chandan/courses/CNT/notes/lec20.pdf. Scribe: Chandan Saha. June 2012.

[7] Jonathan M. Borwein and Peter B. Borwein. *Pi and the AGM: A Study in Analytic Number Theory and Computational Complexity*. Wiley-Interscience, July 1998, p. 432. ISBN: 978-0-471-31515-5. URL.

[8] R. P. Brent and P. Zimmermann. *Modern Computer Arithmetic*. Vol. 18. 2010. ISBN: 978-0-521-19475-5. URL.

[9] Richard P. Brent. "Multiple-Precision Zero-Finding Methods and the Complexity of Elementary Function Evaluation". In: *Analytic Computational Complexity*. Ed. by J. F. Traub. New York: Academic Press, 1975, pp. 151–176. URL.

[10] J. P. Buhler, H. W. Lenstra Jr., and Carl Pomerance. "The development of the number field sieve". In: *Factoring integers with the number field sieve*. Vol. 1554. Lecture Notes in Mathematics. Springer, 2006, pp. 50–94.

[11] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. 9th. Boston, MA: Brooks/Cole, Cengage Learning, 2010. ISBN: 978-0-538-73351-9.

[12] Haotian Chen. "On locating the zeros and poles of a meromorphic function". In: *Journal of Computational and Applied Mathematics* 402 [**Mar. 2022**]. DOI: 10.1016/j.cam.2021.113796. URL.

[13] David Chudnovsky and Gregory Chudnovsky. "Computer Algebra in the Service of Mathematical Physics and Number Theory". In: *Computers in Mathematics*. Vol. 125. Lecture Notes in Pure and Applied Mathematics. Based on the conference "Computers in Mathematics", Stanford, CA, 1986. New York: Dekker, 1990, pp. 109–232.

[14] David V. Chudnovsky and Gregory V. Chudnovsky. “Computer Assisted Number Theory with Applications”. In: *Number Theory: Proceedings of the First Conference of the Canadian Number Theory Association*. Vol. 1240. Lecture Notes in Mathematics. Based on invited lectures. Berlin, Heidelberg: Springer, 1987, pp. 1–72.

[15] Keith Conrad. *The Remainder in Taylor series*. https://kconrad.math.uconn.edu/blurbs/analysis/TaylorRemainder.pdf. Accessed: 2025-04-07. n.d.

[16] Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. 2nd. Springer Science & Business Media. New York: Springer, 2005. ISBN: 978-0-387-25282-7.

[17] L. M. Delves and J. N. Lyness. “A Numerical Method for Locating the Zeros of an Analytic Function”. In: *Mathematics of Computation* 21.100 [**1967**], pp. 543–560.

[18] L. Demanet and l. Ying. *Chebyshev Interpolation*. Accessed: 2024-12-18. 2010. URL.

[19] John D. Dixon. “Asymptotically Fast Factorization of Integers”. In: *Mathematics of Computation* 36.153 [**Jan. 1981**], pp. 255–260.

[20] Uri Elias. *Fundamentals of Ordinary Differential Equations*. Chapter “Solution of Differential Equations by Power Series” (Ch. 8). Cham, Switzerland: Birkhäuser, Springer Nature Switzerland AG, 2025. ISBN: 978-3-031-86531-2. DOI: 10.1007/978-3-031-86532-9. URL.

[21] Walter Gautschi. *Orthogonal Polynomials: Computation and Approximation*. Numerical Mathematics and Scientific Computation. Oxford: Oxford University Press, 2004. ISBN: 978-0-19-850672-0.

[22] David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. 2nd ed. Waltham, MA: Morgan Kaufmann, 2013. ISBN: 978-0123944245.

[23] D. Harvey and J. van der Hoeven. “Integer multiplication in time $O(n \log n)$”. In: *Annals of Mathematics* 193.2 [**2021**], pp. 563–617.

[24] J. van der Hoeven. “Fast evaluation of holonomic functions”. In: *TCS* 210 [**1999**], pp. 199–215.

[25] J. van der Hoeven. *Uniformly fast evaluation of holonomic functions*. Tech. rep. HAL, 2016.

[26] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd. Pearson, 2006.

[27] Hui Huang and Manuel Kauers. “D-finite Numbers”. In: *International Journal of Number Theory* 14.7 [**2018**], pp. 1827–1848. DOI: 10.1142/S1793042118501099. URL.

[28] Nikolaos I. Ioakimidis. “Quadrature Methods for the Determination of Zeros of Transcendental Functions - A Review”. In: *Numerical Integration: Recent Developments, Software and Applications*. Ed. by Patrick Keast and Graeme Fairweather. Dordrecht: Springer Netherlands, 1987, pp. 61–82. ISBN: 978-94-009-3889-2. DOI: 10.1007/978-94-009-3889-2_5. URL.

[29] Antonio Jiménez-Pastor, Veronika Pillwein, and Michael F. Singer. “Some structural results on D^n-finite functions”. In: *Advances in Applied Mathematics* 117 [**June 2020**], p. 102027. ISSN: 0196-8858. DOI: 10.1016/j.aam.2020.102027. URL.

[30] David Kahaner, Cleve Moler, and Stephen Nash. *Numerical Methods and Software*. Accessed: 2025-04-08. Englewood Cliffs, NJ: Prentice Hall, 1989. URL.

[31] N. S. Kambo. "Error of the Newton-Cotes and Gauss-Legendre Quadrature Formulas". In: *Mathematics of Computation* 24.110 [**Apr. 1970**], pp. 665–669.

[32] P. Mateus and V.R. Vieira. *Reducing the factorization of a semiprime integer to the integration of highly oscillatory functions*. Accessed: 2024-12-18. 2011. URL.

[33] Marc Mezzarobba. "Autour de l'évaluation numérique des fonctions D-finies". Thèse de doctorat. École polytechnique, Nov. 2011. URL.

[34] K. C. Ng. *Argument Reduction for Huge Arguments: Good to the Last Bit*. Technical Report. TEXed version by Derek O'Connor. SunPro, 1992. URL.

[35] Jeremy Orlof. *Topic 11 Notes — Complex Variables with Applications*. MIT OpenCourseWare, 18.04 Complex Variables with Applications, Spring 2018. 2018.

[36] Lucjan Piela. *Ideas of Quantum Chemistry*. 2nd ed. Elsevier, 2013. ISBN: 978-0-444-59436-5.

[37] C. Pomerance. "A Tale of Two Sieves". In: *Notices of the American Mathematical Society* 43.12 [**1996**], pp. 1473–1485. URL.

[38] Elaine Rich. *Automata, Computability and Complexity: Theory and Applications*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. ISBN: 978-0132288064.

[39] Edward B. Saff and Arthur David Snider. *Fundamentals of Complex Analysis with Applications to Engineering, Science, and Mathematics*. 3rd. Upper Saddle River, NJ: Prentice Hall, 2003. ISBN: 978-0139078743.

[40] Boris V. Shabat. *Introduction to Complex Analysis*. Russian. https://math324.narod.ru/olderfiles/1/SHabat_B.V_Vvedenie_v_kompleksnuei-90723.pdf. Moscow: Nauka, 1969.

[41] Michael Sipser. *Introduction to the Theory of Computation*. 3rd ed. Library of Congress Control Number: 2012938665. Boston: Cengage Learning, 2013. ISBN: 978-1133187790.

[42] L. N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2018.

[43] Luca Trevisan. *CS254: Computational Complexity, Handout 2*. Course handout. Mar. 2010.

[44] C. K. Yap. "Uniform Complexity of Approximating Hypergeometric Functions with Absolute Error". In: [**2009**]. Available on ResearchGate. URL.