Efficient Homomorphic Comparison Methods with Optimal Complexity

Jung Hee Cheon, Dongwoo Kim and Duhyeong Kim

Department of Mathematical Sciences, Seoul National University
`{jhcheon,dwkim606,doodoo1204}@snu.ac.kr`

Abstract. Comparison of two numbers is one of the most frequently used operations, but it has been a challenging task to efficiently compute the comparison function in homomorphic encryption (HE) which basically support addition and multiplication. Recently, Cheon et al. (Asiacrypt 2019) introduced a new approximate representation of the comparison function with a rational function, and showed that this rational function can be evaluated by an iterative algorithm. Due to this iterative feature, their method achieves a logarithmic computational complexity compared to previous polynomial approximation methods; however, the computational complexity is still not optimal, and the algorithm is quite slow for large-bit inputs in HE implementation.

In this work, we propose new comparison methods with *optimal* asymptotic complexity based on *composite polynomial* approximation. The main idea is to systematically design a constant-degree polynomial f by identifying the *core properties* to make a composite polynomial $f \circ f \circ \cdots \circ f$ get close to the sign function (equivalent to the comparison function) as the number of compositions increases. We additionally introduce an acceleration method applying a mixed polynomial composition $f \circ \cdots \circ f \circ g \circ \cdots \circ g$ for some other polynomial g with different properties instead of $f \circ f \circ \cdots \circ f$. Utilizing the devised polynomials f and g, our new comparison algorithms only require $\Theta(\log(1/\epsilon)) + \Theta(\log \alpha)$ computational complexity to obtain an approximate comparison result of $a, b \in [0, 1]$ satisfying $|a - b| \geq \epsilon$ within $2^{-\alpha}$ error.

The asymptotic optimality results in substantial performance enhancement: our comparison algorithm on encrypted 20-bit integers for $\alpha = 20$ takes 1.43 milliseconds in amortized running time, which is 30 times faster than the previous work.

1 Introduction

Homomorphic Encryption (HE) is a primitive of cryptographic computing, which allows computations over encrypted data without any decryption process. With HE, clients who sent encrypted data to an untrusted server are guaranteed data privacy, and the server can perform any operations over the encrypted data. In recent years, HE has gained worldwide interest from various fields related to data privacy issues including genomics [36, 37, 38] and finances [4, 30]. In particular, HE is emerging as one of the key tools to protect data privacy in machine learning tasks, which now became a necessary consideration due to public awareness of data breaches and privacy violation.

The comparison function $\mathrm{comp}(a, b)$, which outputs 1 if $a > b$, 0 if $a < b$ and $1/2$ if $a = b$, is one of the most prevalent operations along with addition and multiplication in various real-world applications. For example, many of the machine learning algorithms such as cluster analysis [18, 32], gradient boosting [24, 25], and support-vector machine [20, 39] require a number of comparison operations. Therefore, it is indispensable to find an efficient method to compute the comparison function in an encrypted state for HE applications.

Since HE schemes [8, 12, 23] basically support homomorphic addition and multiplication, to compute non-polynomial operations including the comparison function in an encrypted state, we need to exploit polynomial approximations on them. The usual polynomial approximation methods such as minimax find approximate polynomials with minimal degree on a target function for given a certain error bound. However, the computational complexity to evaluate these polynomials is so large that it is quite inefficient to obtain approximate results with high-precision by these methods. Recently, to resolve this problem, Cheon et al. [13] introduced a new identity $\mathrm{comp}(a, b) = \lim_{k\to\infty} a^k/(a^k + b^k)$, and showed that the identity can be computed by an iterative algorithm. Due to this iterative feature, their algorithm achieves a logarithmic computational complexity compared to usual polynomial approximation methods. However, the algorithm only achieves quasi-optimal computational complexity, and it is quite slow in HE implementation; more than 20 minutes is required to compute a single homomorphic comparison of 16-bit integers.

In this work, we propose new comparison methods using composite polynomial approximation on the sign function, which is equivalent to the comparison function. Starting from the analysis on the behavior of a composite polynomial $f^{(d)} := f \circ f \circ \cdots \circ f$, we identify the *core properties* of f that make $f^{(d)}$ get close to the sign function as d increases. We additionally introduce a novel acceleration method by applying a *mixed composition* of f and some other polynomial g with different properties instead of a simple composition of f. Applying these systematically devised polynomials f and g, we construct new comparison algorithms which firstly achieve the *optimal* computational complexity *among all polynomial evaluations* to obtain an approximate value of the comparison result within a certain error bound.

Our composite polynomial methods can be directly applied to evalute piecewise polynomials with two sub-polynomials including the absolute function: For example, the function p such that $p(x) = p_1(x)$ if $x \in [0, 1]$ and $p(x) = p_2(x)$ if $x \in [-1, 0)$ for polynomials p_1 and p_2 can be represented by $p_1(x)\cdot(1+\mathrm{sgn}(x))/2+p_2(x)\cdot(1-\mathrm{sgn}(x))/2$. Furthermore, our method is potentially applicable to more general piecewise polynomials including step functions (see Remark 1).

1.1 Our Idea and Technical Overview

Our key idea to identify several core properties of the basic function f essentially comes from a new interpretation of the previous work [13]. To be precise, [13] exploits the following identity to construct a comparison algorithm:

$$\lim_{k\to\infty} \frac{a^k}{a^k+b^k} = \left\{\begin{matrix} 1 & \text{if } a>b \\ 1/2 & \text{if } a=b \\ 0 & \text{if } a<b \end{matrix}\right\} = \text{comp}(a,b)$$

for positive numbers $a, b \in [1/2, 3/2]$. Since very large exponent $k = 2^d$ is required to obtain a comparison result within small error, they suggest to iteratively compute $a \leftarrow a^2/(a^2+b^2)$ and $b \leftarrow b^2/(a^2+b^2)$, which results in $a^{2^d}/(a^{2^d}+b^{2^d}) \simeq \text{comp}(a,b)$ after d iterations. The inverse operation $1/(a^2+b^2)$ in each iteration is computed by Goldschmidt's division algorithm [29].

The computational inefficiency of the comparison algorithm in [13] mainly comes from the inverse operation which should be done at least d times. Then, the natural question would be

"How can we construct an efficient comparison algorithm without inverse operation?"

To do this, we analyze the comparison algorithm in [13] with a new perspective. Let $f_0(x) = x^2/(x^2+(1-x)^2)$, then each iteration $a \leftarrow a^2/(a^2+b^2)$ and $b \leftarrow b^2/(a^2+b^2)$ can be interpreted as an evaluation of $f_0(a)$ and $f_0(b) = 1 - f_0(a)$ for $0 \le a, b \le 1$, respectively. Indeed, the d iterations correspond to the d-time composition of the basic function f_0 denoted by $f_0^{(d)} := f_0 \circ f_0 \circ \cdots \circ f_0$, and the comparison algorithm can be interpreted as approximating $(\text{sgn}(2x-1)+1)/2$ by a composite polynomial $f_0^{(d)}$.

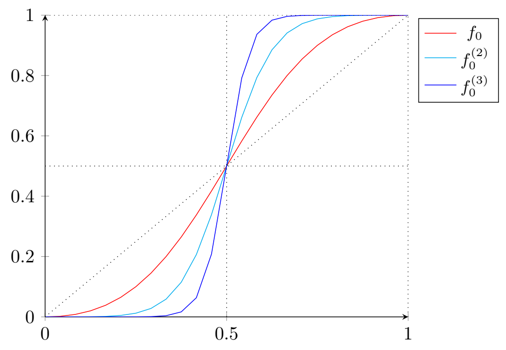


Fig. 1: Illustration of $f_0^{(d)}$ for $d = 1, 2, 3$

Our key observation on the basic function f_0 is that we actually do not need the exact formula of $f_0(x) = x^2/(x^2+(1-x)^2)$. Instead, it suffices to use other polynomials with *similar shape* to f_0: convex in $[0, 0.5]$, concave in $[0.5, 1]$, symmetric to the point $(0.5, 0.5)$, and have a value 1 at $x = 1$. For example, the composition $h_1^{(d)}$ of our devised polynomial $h_1(x) = -2x^3 + 3x^2$, which has similar shape to f_0, gets close to $(\mathrm{sgn}(2x-1)+1)/2$ as d increases. As a result, we can approximate the comparison function by a composite polynomial $f^{(d)}$ for some constant-degree polynomial f with several *core properties*, and identifying these core properties is the most important step in our algorithm construction.

Core Properties of f. Since the sign function is equivalent to the comparison function, via $\mathrm{sgn}(x) = 2 \cdot \mathrm{comp}(x, 0) - 1$ and $\mathrm{comp}(a, b) = (\mathrm{sgn}(a-b)+1)/2$, it is enough to find a polynomial f such that $f^{(d)}(x)$ *gets close to* $\mathrm{sgn}(x)$ over $[-1, 1]$ for some proper d. The core properties of f are as following:

Prop I. $f(-x) = -f(x)$

Prop II. $f(1) = 1$, $f(-1) = -1$

Prop III. $f'(x) = c(1-x)^n(1+x)^n$ for some constant $c > 0$

The first property is necessary from the origin symmetry of the sign function, and the second property is required to achieve $\lim_{d\to\infty} f^{(d)}(x) = 1$ for $0 < x \le 1$. The last property makes f to be concave in $[0, 1]$ and convex in $[-1, 0]$, and the multiplicity n of ± 1 in $f'(x)$ accelerates the convergence of $f^{(d)}$ to the sign function. Interestingly, for each $n \ge 1$, a polynomial f_n satisfying above three properties is uniquely determined as

$$f_n(x) = \sum_{i=0}^{n} \frac{1}{4^i} \cdot \binom{2i}{i} \cdot x(1-x^2)^i.$$

Since $\mathrm{sgn}(x)$ is a discontinuous function at $x = 0$, the closeness of a polynomial $f(x)$ to $\mathrm{sgn}(x)$ should be considered carefully. Namely, we do not consider a small neighborhood $(-\epsilon, \epsilon)$ of zero when measuring the difference between $f(x)$ and $\mathrm{sgn}(x)$. In Section 3.2, we prove that the infinite norm of $f_n^{(d)}(x) - \mathrm{sgn}(x)$ over $[-1, -\epsilon] \cup [\epsilon, 1]$ is smaller than $2^{-\alpha}$ if $d \ge d_n$ for some $d_n > 0$. Then, $(f_n^{(d_n)}(a-b)+1)/2$ outputs an approximate value of $\mathrm{comp}(a, b)$ within $2^{-\alpha}$ error for $a, b \in [0, 1]$ satisfying $|a - b| \ge \epsilon$.

Acceleration Method. Along with $\{f_n\}_{n\ge1}$, we provide another family of odd polynomials $\{g_n\}_{n\ge1}$ which reduces the required number of polynomial compositions d_n. At a high-level, we can interpret d_n as $d_n := d_\epsilon + d_\alpha$ where each of the terms d_ϵ and d_α has distinct aim as following: The first term d_ϵ is a required number of compositions to map the interval $[\epsilon, 1]$ into the interval $[1-\tau, 1]$ for some fixed constant $0 < \tau < 1$ (typically, $\tau = 1/4$), and the second term d_α is a required number of compositions to map $[1-\tau, 1]$ into $[1-2^{-\alpha}, 1]$, i.e.,

$$f_n^{(d_\epsilon)}([\epsilon, 1]) \subseteq [1-\tau, 1],$$
$$f_n^{(d_\alpha)}([1-\tau, 1]) \subseteq [1-2^{-\alpha}, 1].$$

In this perspective, our idea is to reduce d_ϵ by substituting $f_n^{(d_\epsilon+d_\alpha)}$ with $f_n^{(d_\alpha)} \circ g_n^{(d_\epsilon)}$ for some other $(2n+1)$-degree polynomial g_n with *weaker properties* than the core properties of f_n. Since the first d_ϵ compositions only needs to map $[\epsilon, 1]$ into $[1-\tau, 1]$, Prop II & III are *unnecessary* in this part. Instead, the following property along with Prop I is required:

Prop IV. $\exists\ 0 < \delta < 1$ s.t. $x < g_n(x) \le 1$ for $x \in (0, \delta]$ and $g_n([\delta, 1]) \subseteq [1-\tau, 1]$

For g_n satisfying Prop I & IV, the composition $g_n^{(d)}$ does not get close to the sign function as d increases; however, we can guarantee that $g_n^{(d_\epsilon)}([\epsilon, 1]) \subseteq [1-\tau, 1]$ for some $d_\epsilon > 0$ which is exactly the aim of first d_ϵ compositions. With some heuristic properties on g_n obtained by Algorithm 2, the required number of the first-part compositions d_ϵ is reduced by nearly half (see Section 3.5).

1.2 Our Results

New Comparison Methods with Optimal Complexity. We first propose a family of polynomials $\{f_n\}_{n\ge1}$ whose composition $f_n^{(d)}$ gets close to the sign function (in terms of (α, ϵ)-closeness) as d increases. Based on the approximation

$$\frac{f_n^{(d)}(a-b)+1}{2} \simeq \frac{\mathrm{sgn}(a-b)+1}{2} = \mathrm{comp}(a, b),$$

we construct a new comparison algorithm `NewComp`$(a, b; n, d)$ which achieves *optimal asymptotic complexity* among the polynomial evaluations obtaining an approximate value of comparison within a certain level of error. The following theorem is the first main result of our work:

Theorem 1. *If $d \ge \frac{2+o(1)}{\log n} \cdot \log(1/\epsilon) + \frac{1}{\log n} \cdot \log\alpha + O(1)$, the comparison algorithm `NewComp`$(a, b; n, d)$ outputs an approximate value of comp(a, b) within $2^{-\alpha}$ error for $a, b \in [0, 1]$ satisfying $|a-b| \ge \epsilon$.*

The theorem implies that one can obtain an approximate value of comp(a, b) within $2^{-\alpha}$ error for $a, b \in [0, 1]$ satisfying $|a-b| \ge \epsilon$ with $\Theta(\log(1/\epsilon)) + \Theta(\log\alpha) + O(1)$ complexity and depth with `NewComp`.

We also provide another family of polynomials $\{g_n\}_{n\ge1}$, which enables to reduce the number of polynomial compositions by substituting $f_n^{(d)}$ with $f_n^{(d_f)} \circ g_n^{(d_g)}$. From the mixed polynomial composition, we construct another comparison algorithm `NewCompG` with the following result:

Theorem 2 (Heuristic). *If $d_g \ge \frac{1+o(1)}{\log n} \cdot \log(1/\epsilon) + O(1)$ and $d_f \ge \frac{1}{\log n} \cdot \log\alpha + O(1)$, the comparison algorithm `NewCompG`$(a, b; n, d_f, d_g)$ outputs an approximate value of comp(a, b) within $2^{-\alpha}$ error for $a, b \in [0, 1]$ satisfying $|a-b| \ge \epsilon$.*

Since g_n and f_n have the same degree, the total depth and computational complexity of `NewCompG` are strictly smaller than those of `NewComp`.

The variety on choosing n in our comparison algorithms provides flexibility in complexity-depth tradeoff. For instance, one can choose $n = 4$ to achieve the minimal computational complexity (see Section 3.4). On the other hand, if one wants to obtain comparison results with larger complexity but smaller depth, one can choose n larger than 4. Assuming some heuristic properties of g_n, the total depth of $\texttt{NewCompG}(\cdot, \cdot; n, d_f, d_g)$ gets close to the theoretical minimal depth as n increases (see Section 3.5).

Improved Performance. For two 8-bit integers which are encrypted by an approximate HE scheme HEAAN [12], the comparison algorithm `NewComp` (for $\epsilon = 2^{-8}$ and $\alpha = 8$) takes 0.9 milliseconds in amortized running time, and the performance is twice accelerated by applying the other comparison algorithm `NewCompG`. The implementation result on `NewCompG` is about 8 times faster than that on the comparison algorithm of the previous work [13] based on HEAAN. Note that this performance gap grows up as the bit-length of input integers increases: For two encrypted 20-bit integers, our algorithm `NewCompG` is about 30 times faster than the previous work.

Application to Max. Since the max function is expressed by the sign function as $\max(a, b) = \frac{a+b}{2} + \frac{a-b}{2} \cdot \mathrm{sgn}(a-b)$, we can directly obtain max algorithms from the family of polynomials $\{f_n\}_{n\geq 1}$ (and hence $\{g_n\}_{n\geq 1}$). Our max algorithms `NewMax` and `NewMaxG` outperform the max algorithm in the previous work [13] in terms of both computational complexity and depth. To be precise, the max algorithm in [13] requires $4\alpha + O(1)$ depth and $6\alpha + O(1)$ complexity to obtain an approximate value of min/max of two numbers in $[0, 1]$ within $2^{-\alpha}$ error. In our case, the max algorithm `NewMax` applying f_4 only require $3.08\alpha + O(1)$ depth and complexity, and it can be even reduced to $1.54\alpha + 1.72 \log \alpha + O(1)$ by using the other max algorithm `NewMaxG`. In practice, for encrypted 20-bit integers our `NewMaxG` algorithm is 4.5 times faster than the max algorithm in [13].

Moreover, our max algorithms fundamentally solve a potential problem of the max algorithm in [13] when inputs are encrypted by HEAAN. When two input numbers are too close so that the difference is even smaller than approximate errors of HEAAN, then the max algorithm in [13] may output a totally wrong result; in contrast, our max algorithms works well for any inputs from $[0, 1]$.

1.3 Related Works

Numerical Analysis on the Sign Function. In the literature of numerical analysis, to the best of our knowledge, there exist two main approaches on the polynomial approximation of the sign function. One is to naively apply general polynomial approximation methods (Taylor, least squares, minimax, etc.), and the other is to apply Newton's root-finding algorithm on a function which has ± 1 as roots.

General polynomial approximation methods provide an approximate polynomial with *minimal degree* under a certain upper bound of the approximate error. However, the evaluation of such approximate polynomial requires at least $\Theta(\sqrt{degree})$ multiplications, which yields *super-large computational complexity* when we aim to obtain a high-precision approximation. For example, when we want to obtain an approximate polynomial of the sign function with α-bit precision over $[-1, -2^{-\alpha}] \cup [2^{-\alpha}, 1]$ via general polynomial approximation methods, the required computational complexity is at least $\Theta(\sqrt{\alpha}\cdot 2^{\alpha/2})$ which is exponential to α (see Section 2.2 for more details). There have been recent works [9, 31] applying Chebyshev polynomial approximation (on the sine function) instead of the minimax polynomial approximation for better efficiency. However, the Chebyshev polynomial approximation method still requires exponential computational complexity with respect to α when it is applied to the sign function.

Newton's root-finding algorithm outputs an approximate value of roots of a function $r(x)$ by iteratively computing $x_{n+1} = x_n - \frac{r(x_n)}{r'(x_n)}$ for an initial point x_0. That is, an iterative computation of the function $f(x) = x - \frac{r(x)}{r'(x)}$ gives an approximate value to one of the roots of r. The most simple choice of r to compute the sign function is $r(x) = 1 - x^2$ which derives $f(x) = \frac{1}{2}\cdot\left(x + \frac{1}{x}\right)$ so-called Newton's method [33, 35]. There have also been several attempts to improve the convergence rate of this iterative method to the sign function by changing f to $f(x) = \frac{3x+x^3}{1+3x^2}$ (Halley's method [41]), $f(x) = \frac{5x+10x^3+x^5}{1+10x^2+5x^4}$ [19], and $f(x) = \frac{10x+98x^3+126x^5+22x^7}{1+42x^2+140x^4+70x^6+3x^8}$ [45].[a] However, all these methods commonly require the inverse operation, and additional polynomial approximation on inverse is required to apply these methods in HE as the previous work [13]. Consequently, these methods are much less efficient than our methods for the evaluation of the sign function in HE due to a number of expensive inverse operations.

There has been proposed another choice of r that makes f a polynomial as in this paper, so-called Newton-Schulz method [33, 35]. When we take $r(x) = 1 - 1/x^2$, the function f is expressed as $f(x) = \frac{x}{2}\cdot(3 - x^2)$ and we can obtain an approximate value of the sign function by the iterative computation of f. Interestingly, this function is one of our devised polynomials f_1. However, we note that the design rationale of our methods, setting core properties of f that makes $f^{(d)}$ get close to the sign function as d increases, is totally different from that of the Newton's root-finding method. With Newton's method it is not clear at all how to generalize f_1 to f_n for $n > 1$ or how to obtain the intuition for devising other polynomials $\{g_n\}_{n\geq 1}$ for convergence acceleration. Our methods applying $\{f_n\}_{n>1}$ and $\{g_n\}_{n\geq 1}$ achieve much less computational complexity and depth than the previous numerical method (see Section 3.4 and Section 3.5).

HE-based Comparison Methods. There have been several works on comparison algorithms for HE schemes [8, 12, 23] basically supporting addition and multiplication. The most recent work was proposed by Cheon et al. [13] which exploits the identity $\mathrm{comp}(a, b) = \lim_{k\to\infty} \frac{a^k}{a^k+b^k}$ for $a, b > 0$ with an iterative inverse algorithm. Their comparison algorithm requires $\Theta(\alpha\log\alpha)$ complexity, which is quasi-optimal, to obtain an approximate value of $\mathrm{comp}(a, b)$ within $2^{-\alpha}$ error for $a, b \in [1/2, 3/2]$ satisfying $\max(a, b)/\min(a, b) \geq 1 + 2^{-\alpha}$.

[a] In fact, this line of work in numerical analysis aims to compute the matrix sign function [35] which is a more general object than the sign function in our context. An inverse operation is not much more costly than a multiplication in their (asymptotic) cost analysis and experiments, which is a crucial difference from HE which requires an additional costly polynomial approximation for inverse [13].

There have been several approaches to approximate the sign function by polynomials to obtain a comparison algorithm. In 2018, Boura et al. [6] proposed an analytic method to compute the sign function by approximating it via Fourier series over a target interval which has an advantage on numerical stability. In this method, one should additionally consider the error induced by the polynomial approximation on e^{ix}. Another approach is to approximate the sign function by $\tanh(kx) = \frac{e^{kx}-e^{-kx}}{e^{kx}+e^{-kx}}$ for sufficiently large $k > 0$ [15]. In order to efficiently compute $\tanh(kx)$, they repeatedly apply the double-angle formula $\tanh(2x) = \frac{2\tanh(x)}{1+\tanh^2(x)} \approx \frac{2x}{1+x^2}$ where the inverse operation is substituted by a low-degree minimax approximate polynomial. This procedure can be interpreted as a composition of polynomial f which is the low-degree minimax approximation polynomial of $\frac{2x}{1+x^2}$. However, their method does not catch core properties of the basic polynomial f (e.g., $f(1) = 1$), so the error between $f^{(d)}$ and $\mathrm{sgn}(x)$ cannot be reduced below a certain bound even if we increase d to ∞. As an independent work[b], Bajard et al. [5] recently proposed a new approach to approximately compute the sign function by applying the Newton's root-finding method on the function $r(x) = 1-1/x^2$, which corresponds to one of our devised polynomials f_1.

When each bit of message is encrypted separately [14, 17], one can perform a comparison operation of two α-bit integers with $O(\log\alpha)$ depth and $O(\alpha)$ complexity. The bit-by-bit encryption method was recently generalized to encrypt an integer a after decomposing it as $a = \sum a_i b^i$ for a power of small prime $b = p^r$ [46]. However, since these encryption methods are quite inefficient for addition and multiplication, they are not desirable when comparison operations are mixed with a number of polynomials such as cluster analysis and gradient tree boosting.

2 Preliminaries

2.1 Notations

All logarithms are of base 2 unless otherwise indicated, and e denotes the Euler's constant. $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$ denote the integer ring, the real number field and complex number field, respectively. For a finite set X, we denote the uniform distribution over X by $U(X)$. For a real-valued function f defined over $\mathbb{R}$ and a compact set $I \subset \mathbb{R}$, we denote the infinity norm of f over the domain I by $||f||_{\infty,I} := \max_{x\in I}|f(x)|$. The d-times composition of f is denoted by $f^{(d)} := f \circ f \circ \cdots \circ f$. We denote the sign function and the comparison function by

$$\mathrm{sgn}(x) := \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}, \quad \mathrm{comp}(a,b) := \begin{cases} 1 & \text{if } a > b \\ 1/2 & \text{if } a = b \\ 0 & \text{if } a < b \end{cases},$$

[b]Our paper was previously submitted to Eurocrypt 2020 prior to the publication of Bajard et al. [5] in 07 October 2019.

which are in fact equivalent to each other by $\mathrm{comp}(a, b) = (\mathrm{sgn}(a - b) + 1)/2$.

For $\alpha > 0$ and $0 < \epsilon < 1$, we say a polynomial f is (α, ϵ)-*close* to $\mathrm{sgn}(x)$ over $[-1, 1]$ if it satisfies

$$||f(x) - \mathrm{sgn}(x)||_{\infty,[-1,-\epsilon]\cup[\epsilon,1]} \leq 2^{-\alpha}.$$

For $a, b \in \mathbb{R}$, we denote the complexity $a \cdot \log(1/\epsilon) + b \cdot \log \alpha + O(1)$ by $L(a, b)$. The O notation in this paper regards to α and $1/\epsilon$. In the rest of this paper, we only consider the (non-scalar) multiplicative depth and (non-scalar) multiplicative computational complexity, i.e., we do not count the number of additions nor scalar multiplications in computational complexity.

2.2 Minimax Polynomial Approximation Method

In this paper, we measure the accuracy of polynomial approximation methods by the maximal error between the target function and an approximate polynomial over a predetermined domain. In this respect, the minimax approximation method provides the best approximate polynomials among general polynomial approximation methods. For a positive odd integer k, let us denote by $p_{k,\epsilon}$ the degree-k polynomial p which minimizes $||\mathrm{sgn}(x) - p(x)||_{\infty,[-1,-\epsilon]\cup[\epsilon,1]}$. For the sign function $\mathrm{sgn}(x)$, there exists a tight lower bound on the approximation error:

$$\lim_{k\to\infty} \sqrt{\frac{k-1}{2}} \cdot \left(\frac{1+\epsilon}{1-\epsilon}\right)^{\frac{k-1}{2}} \cdot ||\mathrm{sgn}(x) - p_{k,\epsilon}(x)||_{\infty,[-1,-\epsilon]\cup[\epsilon,1]} = \frac{1-\epsilon}{\sqrt{\pi\epsilon}}$$

for $0 < \epsilon < 1$, which was proved by Eremenko and Yuditskii [22]. More general works on minimax polynomial approximation of piecewise analytic function have been proposed [3, 43], but [22] provides more tight and accurate results on error analysis for the sign function.

Assume that k is large enough so that the left-hand side $\sqrt{\frac{k-1}{2}} \cdot \left(\frac{1+\epsilon}{1-\epsilon}\right)^{\frac{k-1}{2}} \cdot ||\mathrm{sgn}(x) - p_{k,\epsilon}(x)||_{\infty,[-1,-\epsilon]\cup[\epsilon,1]}$ is sufficiently close to the limit value. To bound the approximation error by $2^{-\alpha}$ for $\mathrm{sgn}(x)$ over $[-1, -\epsilon] \cup [\epsilon, 1]$, the degree k should be chosen to satisfy

$$\sqrt{\frac{k-1}{2}} \cdot \left(\frac{1+\epsilon}{1-\epsilon}\right)^{\frac{k-1}{2}} \cdot \frac{\sqrt{\pi\epsilon}}{1-\epsilon} > 2^{\alpha},$$

which implies that k should be at least $\Theta(\alpha/\epsilon)$ from the fact $\log\left(\frac{1+\epsilon}{1-\epsilon}\right) \approx \frac{\epsilon}{2}$ for small ϵ. Then, the evaluation of the polynomial $p_{k,\epsilon}$ requires at least $\log \alpha + \log(1/\epsilon) + O(1)$ depth and $\Theta\left(\sqrt{\alpha/\epsilon}\right)$ complexity applying the Paterson-Stockmeyer method [42] which is asymptotically optimal.

There exists a well-known theorem called the equioscillation theorem attributed to Chebychev, which specifies the *shape* of the minimax approximate polynomial $p_{k,\epsilon}$.

Lemma 1 (Equioscillation Theorem for sign function [22]). *Let $sgn(x)$ be the sign function (Section 2.1). For $k \geq 1$ and $0 < \epsilon < 1$, an odd polynomial $p_{k,\epsilon}$ of degree $(2k+1)$ minimizes the infinity norm $||sgn - p_{k,\epsilon}||_{\infty,[-1,-\epsilon]\cup[\epsilon,1]}$ if and only if there are $k+2$ points $\epsilon = x_0 < x_1 < \cdots < x_{k+1} = 1$ such that $sgn(x_i) - p_{k,\epsilon}(x_i) = (-1)^i ||sgn - p_{k,\epsilon}||_\infty$. Here, x_1, x_2,..., x_k are critical points.*

Note that the if-and-only-if statement of the above lemma also implies the *uniqueness* of the minimax polynomial approximation of $\mathrm{sgn}(x)$ on $[-1, -\epsilon] \cup [\epsilon, 1]$ for given ϵ and degree $2k+1$. In the rest of paper, we will use the fact that $p_{k,\epsilon}$ is concave and increasing in the interval $[0, x_0]$ (in fact it holds for $[0, x_1]$).

2.3 Homomorphic Encryption

HE is a cryptographic primitive which allows arithmetic operations including addition and multiplication over encrypted data without decryption process. HE is regarded as a promising solution which prevents leakage of private information during analyses on sensitive data (e.g., genomic data, financial data). A number of HE schemes [7, 8, 12, 16, 21, 23, 27] have been suggested following Gentry's blueprint [26], and achieving successes in various applications [6, 10, 28, 36].

In this paper, we mainly focus on word-wise HE schemes, i.e., the HE schemes whose basic operations are addition and multiplication of encrypted message vectors over $\mathbb{Z}/p\mathbb{Z}$ for $p \geq 2$ [8, 23, 27] or the complex number field $\mathbb{C}$ [12]. An HE scheme consists of the following algorithms:

- $\underline{\texttt{KeyGen}(\mathsf{params})}$. For parameters params determined by a level parameter L and a security parameter λ, output a public key pk, a secret key sk, and an evaluation key evk.
- $\underline{\texttt{Enc}_{\mathsf{pk}}(m)}$. For a message m, output the ciphertext ct of m.
- $\underline{\texttt{Dec}_{\mathsf{sk}}(\mathsf{ct})}$. For a ciphertext ct of m, output the message m.
- $\underline{\texttt{Add}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For ciphertexts ct_1 and ct_2 of m_1 and m_2, output the ciphertext $\mathsf{ct}_{\mathtt{add}}$ of $m_1 + m_2$.
- $\underline{\texttt{Mult}_{\mathsf{evk}}(\mathsf{ct}_1, \mathsf{ct}_2)}$. For ciphertexts ct_1 and ct_2 of m_1 and m_2, output the ciphertext $\mathsf{ct}_{\mathtt{mult}}$ of $m_1 \cdot m_2$.

Though any arithmetic circuit can be computed by HE theoretically, the number of multiplications and multiplicative depth of the circuit are major factors affecting the practical performance and feasibility in real-world applications.

3 Our New Comparison Method

Since the comparison function and the sign function are equivalent, it suffices to find a nice approximate polynomial (with one variable) of the sign function instead of the comparison function (with two variables). In this section, we will introduce new polynomial approximation methods for the sign function which we call *composite polynomial approximation*, and analyze their computational efficiency. As in [13], we assume that the input numbers are contained in the bounded interval $[0,1]$, since $x \in [c_1, c_2]$ for known constants $c_1 < c_2$ can be scaled down into $[0,1]$ via mapping $x \mapsto (x-c_1)/(c_2-c_1)$. Therefore, the domain of $\text{sgn}(x)$ we consider in this paper is $[-1,1]$.

3.1 Composite Polynomial Approximation of Sign Function

As described in [13], approximating a non-polynomial function by composite polynomials has an advantage in computational complexity: A composite function F of a constant-degree polynomial f, i.e., $F := f \circ f \circ \cdots \circ f$, can be computed within $O(\log(\deg F))$ complexity, while the evaluation of an arbitrary polynomial G requires at least $\Theta(\sqrt{\deg G})$ [42]. However, even if this methodology achieves a log-degree computational complexity, it would be meaningless if the total degree of F is extremely large (e.g., $\deg F = 2^{\deg G}$). Therefore, it is very important to *well-design* a constant polynomial f so that it requires small d to make $f^{(d)}$ sufficiently close to $\text{sgn}(x)$ over $[-1,1]$. Since $\text{sgn}(x)$ is discontinuous at $x = 0$, we are not able to obtain a nice polynomial approximation of $\text{sgn}(x)$ over $(-\epsilon, \epsilon)$ for any $0 < \epsilon < 1$. As a result, we set our goal to find f whose composition $f^{(d)}$ is (α, ϵ)-*close* to the sign function for $\alpha > 0$ and $0 < \epsilon < 1$ with small d.

The key observation for designing such polynomial f is as follows: For $x_0 \in [-1,1]$, let x_i be the i-time composition value $f^{(i)}(x_0)$. Then, the behavior of x_i's can be easily estimated with the graph of f. For example, given x_0 on the x-coordinate, x_1 can be identified by the x-coordinate of the intersection point of the graph $y = x$ and the horizontal line $y = f(x_0)$. Note that we can iteratively estimate x_{i+1} with the previous point x_i (see Figure 2).

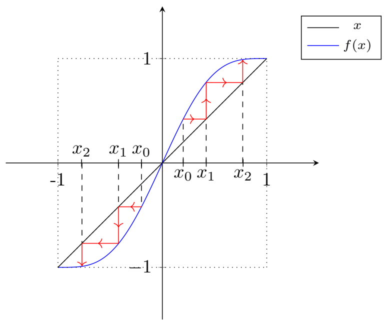


Fig. 2: Behavior of $x_i = f^{(i)}(x_0)$ for $f(x) = -\frac{5}{16}x^7 + \frac{21}{16}x^5 - \frac{35}{16}x^3 + \frac{35}{16}x$

In this perspective, the basic polynomial f should be constructed so that x_i gets close to 1 if $x_0 \in (0,1]$ and -1 if $x_0 \in [-1,0)$ as i increases. We can formally identify three properties of f as follows: Firstly, since the sign function is an odd function, we also set f to be an odd function. Secondly, we set $f(1) = 1$ and $f(-1) = -1$ to make $f^{(d)}(x)$ point-wise converge to $\text{sgn}(x)$ whose value is ± 1 for $x \neq 0$. More precisely, if $f^{(d)}(x)$ for some $x \in [-1,1]$ converges to y as d

increases, it must hold that $f(y) = f\left(\lim_{d\to\infty} f^{(d)}(x)\right) = \lim_{d\to\infty} f^{(d)}(x) = y$. Lastly, f should be considered as a *better* polynomial if it is *more concave* over $[0,1]$ (hence *more convex* over $[-1,0]$), which will accelerate the convergence of $f^{(d)}$ to the sign function. In order to increase convexity, we set the derivative function f' of f to have maximal multiple roots at 1 and -1. These properties are summarized as following.

Core Properties of f:

Prop I.	$f(-x) = -f(x)$	(Origin Symmetry)
Prop II.	$f(1) = 1,\ f(-1) = -1$	(Convergence to ± 1)
Prop III.	$f'(x) = c(1-x)^n(1+x)^n$ for some $c > 0$	(Fast convergence)

For a fixed $n \geq 1$, a polynomial f of the degree $(2n+1)$ satisfying those three properties is uniquely determined, and we denote this polynomial by f_n (and the uniquely determined constant c by c_n): From Prop I and III, we get $f_n(x) = c_n \int_0^x (1-t^2)^n dt$, and the constant c_n is determined by Prop II. By applying the following identity

$$\int_0^x \cos^m t\ dt = \frac{1}{m} \cdot \cos^{m-1} x \cdot \sin x + \frac{m-1}{m} \cdot \int_0^x \cos^{m-2} t\ dt$$

which holds for any $m \geq 1$, we obtain

$$f_n(x) = \sum_{i=0}^{n} \frac{1}{4^i} \cdot \binom{2i}{i} \cdot x(1-x^2)^i.$$

Hence, we can easily compute f_n as following:

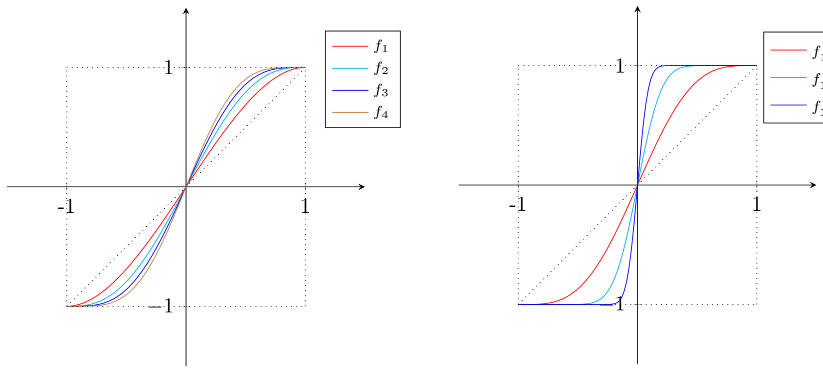
- $f_1(x) = -\frac{1}{2}x^3 + \frac{3}{2}x$
- $f_2(x) = \frac{3}{8}x^5 - \frac{10}{8}x^3 + \frac{15}{8}x$
- $f_3(x) = -\frac{5}{16}x^7 + \frac{21}{16}x^5 - \frac{35}{16}x^3 + \frac{35}{16}x$
- $f_4(x) = \frac{35}{128}x^9 - \frac{180}{128}x^7 + \frac{378}{128}x^5 - \frac{420}{128}x^3 + \frac{315}{128}x$

Since $\binom{2i}{i} = 2 \cdot \binom{2i-1}{i-1}$ is divisible by 2 for $i \geq 1$, every coefficient of f_n can be represented as $m/2^{2n-1}$ for $m \in \mathbb{Z}$.

Size of the Constant c_n. The constant c_n takes an important role on the convergence of $f_n^{(d)}$ (on d) to the sign function. Informally, since the coefficient of x term is exactly c_n, we can regard f_n as $f_n(x) \simeq c_n \cdot x$ for small $x > 0$, and then it holds that $1 - f_n(x) \simeq 1 - c_n \cdot x \simeq (1-x)^{c_n}$. In the next subsection, we will present a rigorous proof of the inequality $1 - f_n(x) \leq (1-x)^{c_n}$ for $0 < x < 1$. (see Section 3.2). From a simple computation, we obtain c_n as a linear summation of binomial coefficients

$$c_n = \sum_{i=0}^{n} \frac{1}{4^i}\binom{2i}{i},$$

which is simplified by the following lemma.



(a) f_n for $n = 1, 2, 3, 4$ (b) $f_1^{(d)}$ for $d = 2, 4, 6$

Fig. 3: Illustration of $f_n^{(d)}$

Lemma 2. *It holds that* $c_n = \sum_{i=0}^{n} \frac{1}{4^i}\binom{2i}{i} = \frac{2n+1}{4^n}\binom{2n}{n}$.

Proof. We prove the statement by induction. It is easy to check for $n = 1$. Assume that $c_n = \frac{2n+1}{4^n}\binom{2n}{n}$ for some $n \geq 1$. Then, it holds that

$$c_{n+1} = c_n + \frac{1}{4^{n+1}}\binom{2n+2}{n+1} = \frac{1}{4^{n+1}} \cdot \left(\frac{2 \cdot (2n+2)!}{(n+1)!n!} + \frac{(2n+2)!}{(n+1)!(n+1)!}\right)$$
$$= \frac{2n+3}{4^{n+1}}\binom{2n+2}{n+1}.$$

Therefore, the lemma is proved by induction. □

To measure the size of c_n, we apply Wallis's formula [34] which gives us very tight lower and upper bound:

$$\frac{1}{\sqrt{\pi}} \cdot \frac{2n+1}{\sqrt{n+\frac{1}{2}}} < \frac{2n+1}{4^n}\binom{2n}{n} < \frac{1}{\sqrt{\pi}} \cdot \frac{2n+1}{\sqrt{n}}.$$

From the inequality, we can check that $c_n = \Theta(\sqrt{n})$, which diverges as $n \to \infty$.

Remark 1. Our method can be naturally generalized to the composite polynomial approximation on *step functions*. For example, if we substitute Prop III by $f'(x) = cx^{2m}(1-x^2)^n$ for $m, n \geq 1$, then $f^{(d)}$ would get close to a step function F (as d increases) such that $F(x) = -1$ if $x \in [-1, -t)$, $F(x) = 0$ if $x \in [-t, t]$ and $F(x) = 1$ if $x \in (t, 1]$, for some $0 < t < 1$ as d increases.

3.2 Analysis on the Convergence of $f_n^{(d)}$

In this subsection, we analyze the convergence of $f_n^{(d)}$ to the sign function as d increases. To be precise, we give a lower bound of d which makes $f_n^{(d)}$ to be

(α, ϵ)-close to the sign function. The following lemma gives a nice upper bound on $1 - f_n(x)$, which is even tighter than the Bernoulli's inequality [40]: This well-known inequality implies $1 - c_n x \leq (1-x)^{c_n}$, but since $1 - c_n x \leq 1 - f_n(x)$ we cannot directly obtain the upper bound of $1 - f_n(x)$ from this inequality.

Lemma 3. *It holds that* $0 \leq 1 - f_n(x) \leq (1-x)^{c_n}$ *for* $x \in [0,1]$.

Proof. It is trivial that $f_n(x) \leq f_n(1) = 1$ for $x \in [0,1]$. We will prove $G(x) := (1-x)^{c_n} - (1 - f_n(x)) \geq 0$ for $x \in [0,1]$ by showing

1. $G(0) = G(1) = 0$,
2. there exists $x_0 \in (0,1)$ s.t. $G(x_0) > 0$,
3. there exists a unique $y_0 \in (0,1)$ s.t. $G'(y_0) = 0$.

We first check why these three conditions derive the result $G(x) \geq 0$. Assume that there exists $x_1 \in (0,1)$ such that $G(x_1) < 0$. Since G is continuous, there exists a root x_2 of G between x_0 and x_1. Then by the mean value theorem, there exist $y_1 \in (0, x_2)$ and $y_2 \in (x_2, 1)$ satisfying $G'(y_1) = G'(y_2) = 0$, which contradicts to the third condition.

Now we prove the three conditions. The first condition is trivial. To show the second condition, we observe $G(0) = 0$, $G'(0) = 0$ and $G''(0) > 0$ which can be easily checked. Since G'' is continuous, $G'(0) = 0$ and $G''(0) > 0$ imply that $G'(x) > 0$ for $x \in (0, \delta)$ for some $\delta > 0$. Combining with $G(0) = 0$, we obtain $G(x) > 0$ for $x \in (0, \delta)$ which implies the second condition.

To show the uniqueness, let $G'(x) = c_n(1-x^2)^n - c_n(1-x)^{c_n - 1} = 0$. Then it holds that $(1-x)^{n - c_n + 1} \cdot (1+x)^n = 1$ for $x \in (0,1)$. Taking a logarithm, the equation is equivalent to

$$\frac{\log(1+x)}{\log(1-x)} = -\frac{n - c_n + 1}{n}.$$

Since $\log(1+x)/\log(1-x)$ is a strictly increasing function, there should exist a unique $y_0 \in (0,1)$ satisfying the equation which implies $G'(y_0) = 0$. ◻

We give another inequality on $1 - f_n(x)$ which is tighter than the inequality in the previous lemma when x is close to 1.

Lemma 4. *It holds that* $0 \leq 1 - f_n(x) \leq 2^n \cdot (1-x)^{n+1}$ *for* $x \in [0,1]$.

Proof. Let $y = 1 - x$, and set

$$H(y) = \frac{c_n \cdot 2^n}{n+1} \cdot y^{n+1} - (1 - f_n(1-y)).$$

Then $H'(y) = c_n \cdot 2^n \cdot y^n - f_n'(1-y) = c_n \cdot 2^n \cdot y^n - c_n \cdot y^n (2-y)^n \geq 0$ for $y \in [0,1]$. Since $H(0) = 0$, it holds that $H(y) \geq 0$. Therefore, we obtain

$$1 - f_n(x) \leq \frac{c_n \cdot 2^n}{n+1} \cdot (1-x)^{n+1} \leq 2^n \cdot (1-x)^{n+1}$$

for $x \in [0,1]$, where the second inequality comes from $c_n < n+1$. ◻

Now we obtain the theorem on the convergence of $f_n^{(d)}$ to the sign function.

Theorem 3 (Convergence of $f_n^{(d)}$). *If $d \geq \frac{1}{\log c_n} \cdot \log(1/\epsilon) + \frac{1}{\log(n+1)} \cdot \log(\alpha - 1) + O(1)$, then $f_n^{(d)}(x)$ is an (α, ϵ)-close polynomial to $sgn(x)$ over $[-1, 1]$.*

Proof. Since f_n is an odd function, it suffices to consider the case that the input x is non-negative. We analyze the lower bound of d for the convergence of $f_n^{(d)}$ by applying Lemma 3 and Lemma 4. Note that Lemma 3 is tighter than Lemma 4 if x is close to 0 while the reverse holds if x is close to 1. To this end, to obtain a tight lower bound of d, our analysis is divided into two steps each of which applies Lemma 3 and Lemma 4, respectively.

Step 1. Since f_n is an odd function, it suffices to consider the case $x \in [\epsilon, 1]$ instead of $[-1, -\epsilon] \cup [\epsilon, 1]$. Let $d_\epsilon = \left\lceil \frac{1}{\log(c_n)} \cdot \log\left(\log\left(\frac{1}{\tau}\right)/\epsilon\right) \right\rceil$ for some constant $0 < \tau < 1$. Then applying Lemma 3, we obtain following inequality for $x \in [\epsilon, 1]$.

$$\begin{aligned} 1 - f_n^{(d_\epsilon)}(x) &\leq (1-x)^{c_n^{d_\epsilon}} \\ &\leq (1-\epsilon)^{\log\left(\frac{1}{\tau}\right)/\epsilon} < \left(\frac{1}{e}\right)^{\log\left(\frac{1}{\tau}\right)} < \tau. \end{aligned}$$

Step 2. Now let $d_\alpha = \left\lceil \frac{1}{\log(n+1)} \cdot \log\left((\alpha - 1)/\log\left(\frac{1}{2\tau}\right)\right) \right\rceil$. Applying previous result and Lemma 4, we obtain following inequality for $x \in [\epsilon, 1]$.

$$\begin{aligned} 2 \cdot \left(1 - f_n^{(d_\epsilon + d_\alpha)}(x)\right) &\leq \left(2 \cdot \left(1 - f_n^{(d_\epsilon)}(x)\right)\right)^{(n+1)^{d_\alpha}} \\ &\leq (2\tau)^{(n+1)^{d_\alpha}} \leq (2\tau)^{(\alpha-1)/\log\left(\frac{1}{2\tau}\right)} = 2^{-\alpha+1}. \end{aligned}$$

Therefore, if $d \geq d_\epsilon + d_\alpha$, we obtain $1 - f_n^{(d)}(x) \leq 2^{-\alpha}$ for $x \in [\epsilon, 1]$.

Note that the choice of the constant τ is independent to ϵ and α. When $\tau = 1/4$, then we get $d_\epsilon + d_\alpha = \frac{1}{\log(c_n)} \cdot \log(1/\epsilon) + \frac{1}{\log(n+1)} \cdot \log(\alpha - 1) + \frac{1}{\log(c_n)} + O(1)$. Since $\frac{1}{\log(c_n)} \leq 2$, the theorem is finally proved. □

3.3 New Comparison Algorithm `NewComp`

Now we introduce our new comparison algorithm based on the previous composite function approximation (Theorem 3) of the sign function. From the identity $\text{comp}(a, b) = (\text{sgn}(a - b) + 1)/2$ and approximation $f_n^{(d)}(x) \simeq \text{sgn}(x)$, we get

$$\text{comp}(a, b) \simeq \frac{f_n^{(d)}(a - b) + 1}{2},$$

which results in our new comparison algorithm denoted by `NewComp` described in Algorithm 1.

It is quite natural that the larger d gives more accurate result. Since the comparison algorithm $\texttt{NewComp}(\cdot, \cdot; n, d)$ is obtained from the evaluation of $f_n^{(d)}$, Theorem 3 is directly transformed into the context of `NewComp` as Corollary 1, which informs us how large d is sufficient to get the result in certain accuracy.

Algorithm 1 `NewComp`$(a, b; n, d)$

```
Input: $a, b \in [0, 1]$, $n, d \in \mathbb{N}$
Output: An approximate value of 1 if $a > b$, 0 if $a < b$ and 1/2 otherwise
1: $x \leftarrow a - b$
2: for $i \leftarrow 1$ to $d$ do
3:     $x \leftarrow f_n(x)$                                  // compute $f_n^{(d)}(a - b)$
4: end for
5: return $(x + 1)/2$
```

Corollary 1. *If $d \geq \frac{1}{\log c_n} \cdot \log(1/\epsilon) + \frac{1}{\log(n+1)} \cdot \log(\alpha - 2) + O(1)$, then the error of the output of* `NewComp`$(a, b; n, d)$ *compared to the true value is bounded by $2^{-\alpha}$ for any $a, b \in [0, 1]$ satisfying $|a - b| \geq \epsilon$.*

Remark 2. One can substitute scalar multiplications in the evaluation of f_n with additions by linearly transforming f_n to an integer-coefficient polynomial h_n as

$$\begin{aligned} h_n(x) := \frac{f_n(2x-1)+1}{2} &= \sum_{i=0}^{n} \frac{1}{4^i} \cdot \binom{2i}{i} \cdot (2x-1) \cdot (4x - 4x^2)^i \\ &= \sum_{i=0}^{n} \binom{2i}{i} \cdot (2x-1) \cdot (x - x^2)^i, \end{aligned}$$

since multiplying m can be interpret as m additions for any positive integer m. Note that it is easily proved that $h_n^{(d)}(x) = \frac{f^{(d)}(2x-1)+1}{2}$ by induction, so we can express the comparison functions as

$$\mathrm{comp}(a, b) \simeq \frac{f_n^{(d)}(a-b)+1}{2} = h_n^{(d)}\left(\frac{(a-b)+1}{2}\right).$$

Therefore, Algorithm 1 can be naturally converted into the context of h_n which does not require scalar multiplications for the evaluation.

3.4 Computational Complexity of `NewComp` and its Asymptotic Optimality

In this subsection, we analyze the computational complexity of our new comparison method, and compare the result with the previous methods. Note that the (multiplicative) computational complexity of `NewComp`$(\cdot, \cdot; n, d)$ equals to that of evaluating $f_n^{(d)}$, so it suffices to focus on this composite polynomial.

For each $n \geq 1$, let C_n be the required number of multiplications (hence the computational complexity) of f_n using some polynomial evaluation algorithm, and denote the lower bound of d in Theorem 3 by $d_n := \frac{1}{\log c_n} \cdot \log(1/\epsilon) + \frac{1}{\log(n+1)} \cdot \log(\alpha - 1) + O(1)$. Then the total computational complexity of $f_n^{(d_n)}$ is $TC_n := d_n \cdot C_n$ which varies on the choice of n. When n becomes larger, then

n	D_n	C_n	d_n	TD_n	TC_n
1	2	2	$L(1.71, 1)$	$L(3.42, 2)$	$L(3.42, 2)$
2	3	3	$L(1.1, 0.63)$	$L(3.3, 1.89)$	$L(3.3, 1.89)$
3	3	4	$L(0.89, 0.5)$	$L(2.67, 1.5)$	$L(3.56, 2)$
4	4	4	$L(0.77, 0.43)$	$L(3.08, 1.72)$	$\boldsymbol{L(3.08, 1.72)}$
5	4	5	$L(0.7, 0.39)$	$L(2.8, 1.56)$	$L(3.5, 2.45)$
6	4	6	$L(0.64, 0.36)$	$L(2.56, 1.44)$	$L(3.84, 2.16)$
7	4	7	$L(0.61, 0.33)$	$L(2.44, 1.32)$	$L(4.27, 2.31)$

Table 1: Depth / Computational Complexity of f_n and $f_n^{(d_n)}$

d_n becomes smaller but C_n becomes larger. Namely, there is a trade-off between d_n and C_n, so we need to find the best choice of n which minimizes the total computational complexity TC_n of $f_n^{(d_n)}$.

We assume that each polynomial f_n is computed by the Paterson-Stockmeyer method [42] which achieves an optimal computational complexity upto constant. Then, the depth is $D_n := \log(\deg f_n) + O(1) = \log n + O(1)$, and the computational complexity is $C_n := \Theta(\sqrt{\deg f_n}) = \Theta(\sqrt{n})$[c]. The total depth of $f_n^{(d_n)}$ is $TD_n := d_n \cdot D_n = L\left(\frac{\log n + O(1)}{\log c_n}, \frac{\log n + O(1)}{\log(n+1)}\right)$ (see Section 2.1 for L notation). Since $c_n = \Theta(\sqrt{n})$ by Lemma 2, the total depth TD_n gets close to $L(2, 1)$ as n increases[d]. On the other hand, the total computational complexity of $f_n^{(d_n)}$ is

$$TC_n := d_n \cdot C_n = L\left(\frac{1}{\log c_n} \cdot \Theta(\sqrt{n}), \frac{1}{\log(n+1)} \cdot \Theta(\sqrt{n})\right),$$

which diverges as n increases, contrary to the total depth TD_n. Therefore, the optimal choice of n which minimize the total complexity TC_n exists. The exact number of multiplications C_n of f_n and the exact value of TC_n for small n's are described in Table 1. From simple computations, we can check that $n = 4$ gives the minimal computational complexity TC_4.

Asymptotic Optimality. As described in Section 2.2, the minimal degree of an (α, ϵ)-close approximate polynomial of the sign function over $[-1, 1]$ is $\Theta(\alpha/\epsilon)$. Since the sign function and the comparison function are equivalent, this implies that any comparison algorithm on inputs $a, b \in [0, 1]$ whose output is within $2^{-\alpha}$ error when $|a - b| \geq \epsilon$ requires at least $\Theta(\log \alpha) + \Theta(\log(1/\epsilon))$ complexity. As described above, the computational complexity of $\texttt{NewComp}(\cdot, \cdot; n, d_n)$

[c]The complexity notations in D_n and C_n only depend on n, not α and ϵ.

[d]It does *not* mean the "convergence" to $L(2, 1)$ as $n \to \infty$, since the equation $TD_n = L\left(\frac{\log n + O(1)}{\log c_n}, \frac{\log n + O(1)}{\log(n+1)}\right)$ only holds when $n = O(1)$ with respect to α and $1/\epsilon$.

is $\Theta(\log\alpha) + \Theta(\log(1/\epsilon))$ for each n. Therefore, our new comparison method achieves an *optimality in asymptotic computational complexity* upto constant, while the previous method [13] only achieves quasi-optimality with an additional $\log\alpha$ factor.

For several settings of α and ϵ, we compare the computational complexity of our method to the minimax approximation and the method in [13] as Table 2.

Parameters	Minimax Approx.	[13] Method	**Our Method**
$\log(1/\epsilon) = \Theta(1)$	$\Theta(\sqrt{\alpha})$	$\Theta(\log^2\alpha)$	$\boldsymbol{\Theta(\log\alpha)}$
$\log(1/\epsilon) = \Theta(\alpha)$	$\Theta(\sqrt{\alpha}\cdot 2^{\alpha/2})$	$\Theta(\alpha\cdot\log\alpha)$	$\boldsymbol{\Theta(\alpha)}$
$\log(1/\epsilon) = 2^\alpha$	$\Theta\left(\sqrt{\alpha}\cdot 2^{2^{\alpha-1}}\right)$	$\Theta(\alpha\cdot 2^\alpha)$	$\boldsymbol{\Theta\left(2^\alpha\right)}$

Table 2: Asymptotic Computational Complexity for each Comparison Method

3.5 Heuristic Methodology of Convergence Acceleration

In the previous subsection, we dealt with the asymptotic optimality in computational complexity of our comparison algorithm `NewComp`. In this subsection, we introduce a heuristic methodology to reduce the *constants* a and b in $L(a,b)$ of the computational complexity TC_n, which accelerates `NewComp` in practice.

The intuition of our acceleration method can be found in the proof of Theorem 3. The proof is divided into two steps: Step 1 is to make $f_n^{(d_\epsilon)}([\epsilon,1]) \subseteq [1-\tau,1]$ for some constant $0<\tau<1$ (applying Lemma 3), and Step 2 is to make $f_n^{(d_\alpha)}([1-\tau,1]) \subseteq [1-2^{-\alpha},1]$ (applying Lemma 4). Our key observation is that we can accelerate Step 1 by using another function g rather than f_n. The convergence of $f_n^{(d)}$ $(1\le d\le d_\epsilon)$ in Step 1 mainly depends on the constant c_n, the derivative of f_n at zero. Therefore, we may expect that the required number of polynomial compositions d_ϵ in Step 1 can be reduced if we substitute f_n by some other odd polynomial g which satisfies $g'(0) > f_n'(0)$.

However, we cannot take any g with large derivative at 0, since the range of $g^{(d)}$ over the domain $[\epsilon,1]$ must be contained in $[1-\tau,1]$ when d is large enough. In particular, the polynomial g must satisfy following properties (compare it with the Core Properties of f in Section 3.1):

Prop I. $g(-x) = -g(x)$ (Origin Symmetry)

Prop IV. $\exists\ 0<\delta<1$ s.t. $x < g(x) \le 1$ for all $x\in(0,\delta]$, (Toward $[1-\tau,1]$)

and $g([\delta,1]) \subseteq [1-\tau,1]$ (Keep in $[1-\tau,1]$)

For each g, we denote the minimal $0<\delta<1$ in Prop IV by δ_0 in the rest of paper.

Note that Prop IV is necessary to make $g^{(d)}(x) \in [1-\tau, 1]$ for $x \in [\epsilon, 1]$ when $d \geq d_0$ for some sufficiently large $d_0 > 0$. Intuitively, among all g of the same degree satisfying above properties, a smaller d is required for $g^{(d)}([\epsilon, 1]) \subseteq [1-\tau, 1]$ if g satisfies Prop IV with smaller δ_0 and has bigger value on the interval $(0, \delta_0)$ (hence $g'(0)$ is bigger).

We introduce a novel algorithm (Algorithm 2) which outputs a degree-$(2n+1)$ polynomial denoted by $g_{n,\tau}$ having *minimal* δ_0 of Prop IV among all degree-$(2n+1)$ polynomials satisfying Prop I & IV. In a certain condition, we can additionally show that $g_{n,\tau}(x) > g(x)$ on $x \in (0, \delta)$ (hence larger derivative at zero) for any other polynomials g satisfying Prop I & IV (see Theorem 4 and Corollary 2). It implies that $g_{n,\tau}$ is the best polynomial among all same-degree polynomials achieving our goal, i.e., $g_{n,\tau}^{(d)}([\epsilon, 1]) \subseteq [1-\tau, 1]$ with minimal d.

Algorithm 2 `FindG`(n, τ)

Input: $n \geq 1$, $0 < \tau < 1$

Output: A degree-$(2n+1)$ polynomial $g_{n,\tau}$ satisfying Prop I & IV with minimal δ of Prop IV.

```
1: $g_{n,\tau} \leftarrow x$                                  // Initialize $g_{n,\tau}(x) = x$
2: repeat
3:     $\delta_0 \leftarrow$ minimal $\delta$ s.t. $g_{n,\tau}([\delta, 1]) \subseteq [1-\tau, 1]$     // Initial $\delta_0$ is $1-\tau$
4:     $g_{min} \leftarrow$ degree-$(2n+1)$ minimax approx. poly. of $(1-\frac{\tau}{2}) \cdot \mathrm{sgn}(x)$ over
       $[-1, -\delta_0] \cup [\delta_0, 1]$
5:     $g_{n,\tau} \leftarrow g_{min}$
6:     $S \leftarrow ||g_{n,\tau} - (1-\frac{\tau}{2})||_{\infty,[\delta_0,1]}$
7: until $S == \frac{\tau}{2}$
8: return $g_{n,\tau}$
```

In Algorithm 2, the equality check $S == \frac{\tau}{2}$ on line 7 is done with a certain precision in practice (e.g., 2^{-10} or 2^{-53}). Note that S converges (increases) to $\frac{\tau}{2}$, δ_0 converges (decreases) to some $\delta_{conv} > 0$, and hence $g_{n,\tau}$ converges to some polynomial $g_{n,\tau}^{conv}$ (see Appendix A). From this, we obtain two facts: First, Algorithm 2 terminates in finite iterations given a finite precision for the equality check. Second, the output of the algorithm satisfies Prop I & IV[e].

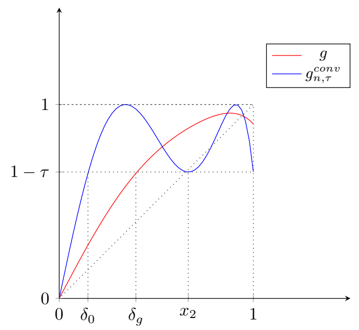
We provide a theoretical analysis on $g_{n,\tau}^{conv}$ to which $g_{n,\tau}$ converges, which we call *the ideal output polynomial* of Algorithm 2. Note that the ideal ouput polynomial $g_{n,\tau}^{conv}$ satisfies $||g_{n,\tau}^{conv} - (1-\frac{\tau}{2})||_{\infty,[\delta_0,1]} = \frac{\tau}{2}$. The following theorem shows the optimality of $g_{n,\tau}^{conv}$, which implies that the real output of Algorithm 2 with a certain precision is nearly optimal.

[e]In every iteration of Algorithm 2, the minimax approximate polynomial g_{min} of $(1-\frac{\tau}{2}) \cdot \mathrm{sgn}(x)$ over $[-1, \delta_0] \cup [\delta_0, 1]$ satisfies Prop I & IV. Prop I is trivial, and $g_{min}([\delta_0, 1]) \subset [1-\tau, 1]$ by Lemma 1. Since $g_{min}(\delta_0) > 1-\tau \geq \delta_0$ and g_{min} is concave & increasing in $[0, \delta_0]$, it holds that $x < g_{min}(x) < 1$ for $x \in (0, \delta_0]$.

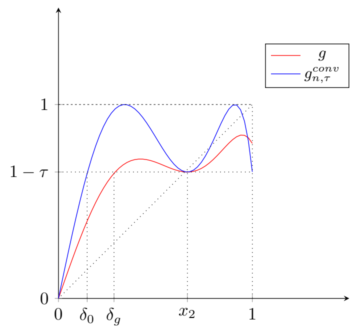
Theorem 4 (Optimality of $g_{n,\tau}^{conv}$). *The ideal output polynomial $g_{n,\tau}^{conv}$ of Algorithm 2 satisfies Prop I & IV with minimal δ_0 among all degree-$(2n+1)$ polynomials satisfying Prop I & IV. Let $x_2 > 0$ be the smallest positive x-coordinate of local minimum points of $g_{n,\tau}$ following the notation in Lemma 1 (If local minimum does not exist, set $x_2 = 1$). If $x_2 \geq 1-\tau$, then $g_{n,\tau}(x) > g(x)$ for $x \in (0, \delta_0)$ for any other degree-$(2n+1)$ polynomial g satisfying Prop I & IV.*

Proof. Let δ_{conv} be the minimal δ such that $g_{n,\tau}^{conv}([\delta, 1]) \subseteq [1-\tau, 1]$. Assume that there exists a degree-$(2n+1)$ polynomial g satisfying Prop I & IV with $\delta \leq \delta_{conv}$. By Prop IV, we get $||g - (1-\frac{\tau}{2})||_{\infty,[\delta,1]} \leq \frac{\tau}{2}$, and then it trivially holds that $||g - (1-\frac{\tau}{2})||_{\infty,[\delta_{conv},1]} \leq \frac{\tau}{2} = ||g_{n,\tau}^{conv} - (1-\frac{\tau}{2})||_{\infty,[\delta_{conv},1]}$. Therefore, $g = g_{n,\tau}^{conv}$ by Lemma 1 which implies the minimality of δ_{conv}.

Now we prove the second statement. Let g be a degree-$(2n+1)$ polynomial satisfying Prop I & IV which is distinct from $g_{n,\tau}^{conv}$, and δ_g be the minimal δ such that $g([\delta, 1]) \subseteq [1-\tau, 1]$. From the minimality of δ_{conv} and Prop IV, it holds that $\delta_{conv} < \delta_g \leq 1-\tau \leq x_2$. By Lemma 1, $g_{n,\tau}^{conv}$ oscillates on $[\delta_{conv}, 1]$ with 1 and $1-\tau$ as maximum and minimum, respectively, and it has n critical points in $(\delta_{conv}, 1)$. Since $g([\delta_g, 1]) \subseteq [1-\tau, 1]$ and $\delta_g \leq x_2$, the polynomial g intersects with $g_{n,\tau}^{conv}$ on at least n points in $[\delta_g, 1]$: when $g(x) = g_{n,\tau}^{conv}(x)$ and $g'(x) = g_{n,\tau}^{conv\prime}(x)$, then x is counted as two points (see Figure 4). Now our second argument is proved as following: If $g(x) \geq g_{n,\tau}^{conv}(x)$[f] on some $x \in (0, \delta_{conv}) \subset (0, \delta_g)$, then g and $g_{n,\tau}^{conv}$ intersect on at least one point in $(0, \delta_g)$ by intermediate value theorem since there exists $y \in (\delta_{conv}, \delta_g)$ such that $g(y) < 1-\tau \leq g_{n,\tau}^{conv}(y)$ by the definition of δ_g. This leads to a contradiction since g and $g_{n,\tau}^{conv}$ intersect on $2(n+1)+1 = 2n+3$ points (the factor 2 comes from the fact that both are odd polynomials) including the origin while the degree of both g and $g_{n,\tau}^{conv}$ is $2n+1 < 2n+3$. Therefore, $g_{n,\tau}^{conv}(x) > g(x)$ for all $x \in (0, \delta_{conv})$. □



(a) Intersections without multiplicity



(b) Intersection with multiplicity at x_2

Fig. 4: Example description of intersections of g and $g_{n,\tau}^{conv}$ for $n = 3$

[f] If $g(x) = g_{n,\tau}^{conv}(x)$ on some $x \in (0, \delta_0)$, it is the point of intersection in $(0, \delta_g)$, and proof continues.

Corollary 2. *Let $g_{n,\tau}^{conv}$ be the ideal output polynomial of Algorithm 2, and δ_0 be the corresponding minimal δ satisfying Prop IV. If $n = 1$, $(n, \tau) = (2, 0.25)$, or $(n, \tau) = (3, 0.35)$, then $\delta_0 < \delta_g$ and $g_{n,\tau}^{conv}(x) > g(x)$ on $x \in (0, \delta_0)$ for any other degree-$(2n+1)$ polynomial g satisfying Prop I & IV.*

Though $g_{n,\tau}$ is hard to be expressed in closed form contrary to f_n, we can find it with a certain precision (e.g., 2^{-10}) by running Algorithm 2 in MATLAB. For example, we provide explicit descriptions of the polynomials $g_{n,\tau}$ for $n = 1, 2, 3, 4$ and $\tau = \frac{1}{4}$. In this case, the equality check in Algorithm 2 was done with 10^{-4} precision. We omit the subscript τ of $g_{n,\tau}$ for $\tau = \frac{1}{4}$ for convenience.

- $g_1(x) = -\frac{1359}{2^{10}} \cdot x^3 + \frac{2126}{2^{10}} \cdot x$
- $g_2(x) = \frac{3796}{2^{10}} \cdot x^5 - \frac{6108}{2^{10}} \cdot x^3 + \frac{3334}{2^{10}} \cdot x$
- $g_3(x) = -\frac{12860}{2^{10}} \cdot x^7 + \frac{25614}{2^{10}} \cdot x^5 - \frac{16577}{2^{10}} \cdot x^3 + \frac{4589}{2^{10}} \cdot x$
- $g_4(x) = \frac{46623}{2^{10}} \cdot x^9 - \frac{113492}{2^{10}} \cdot x^7 + \frac{97015}{2^{10}} \cdot x^5 - \frac{34974}{2^{10}} \cdot x^3 + \frac{5850}{2^{10}} \cdot x$

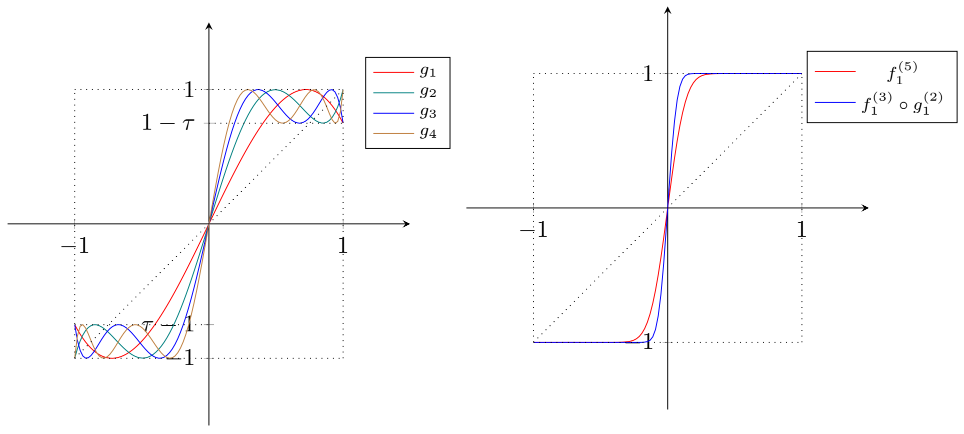


Fig. 5: Illustration of g_n and the comparison of $f_1^{(d_f+d_g)}$ and $f_1^{(d_f)} \circ g_1^{(d_g)}$

We can empirically check that g_n also satisfies the following two heuristic properties. The first property shows how large $g_n'(0)$ is when it compared to $f_n'(0)$, and the second property shows how fast $g_n(x)$ gets close to ± 1, i.e., the g_n-version of Lemma 3.

Heuristic Properties of g_n:

1. $g_n'(0) \simeq 0.98 \cdot f_n'(0)^2$ (Hence, $\log g_n'(0) \simeq 2 \cdot \log c_n$)
2. $1 - g_n(x) \leq (1-x)^{g_n'(0)}$ for $x \in [0, \delta_0]$ where δ_0 is the minimal δ in Prop IV

Experimental results supporting above heuristic properties are described in Appendix B. Applying these g_n polynomials, we can provide a new comparison algorithm (Algorithm 3), which is a modified version of Algorithm 1 and offers the same functionality with the reduced computational complexity and depth. We can also estimate the number of compositions d_f and d_g required for this modified algorithm to achieve a certain accuracy as Corollary 3.

Algorithm 3 `NewCompG`$(a, b; n, d_f, d_g)$

```
Input: $a, b \in [0, 1]$, $n, d_f, d_g \in \mathbb{N}$
Output: An approximate value of 1 if $a > b$, 0 if $a < b$ and 1/2 otherwise
1: $x \leftarrow a - b$
2: for $i \leftarrow 1$ to $d_g$ do
3:     $x \leftarrow g_n(x)$                        // compute $g_n^{(d_g)}(a - b)$
4: end for
5: for $i \leftarrow 1$ to $d_f$ do
6:     $x \leftarrow f_n(x)$                        // compute $f_n^{(d_f)} \circ g_n^{(d_g)}(a - b)$
7: end for
8: return $(x + 1)/2$
```

Corollary 3. *(With Heuristic Properties) If* $d_g \geq \frac{1}{\log g_n'(0)} \cdot \log(1/\epsilon) + O(1) = \frac{1/2+o(1)}{\log c_n} \cdot \log(1/\epsilon) + O(1)$ *and* $d_f \geq \frac{1}{\log n} \cdot \log(\alpha - 2) + O(1)$*, then the error of the output of* `NewCompG`$(a, b; n, d_f, d_g)$ *compared to the true value is bounded by* $2^{-\alpha}$ *for any* $a, b \in [0, 1]$ *satisfying* $|a - b| \geq \epsilon$.

Proof. Following the proof of Theorem 3, it suffices to show that $1 - g_n^{(d_g)}(x) \leq \tau$ for $x \in [\epsilon, 1]$ where $\tau = 1/4$. Let $e_n := g_n'(0)$. By the second heuristic property of g_n, we obtain two inequalities: $1 - g_n^{(d)}(x) \leq (1 - x)^{e_n^d}$ for d satisfying $g_n^{(d-1)}(x) \leq \delta_0$, and $1 - g_n^{(d)}(x) \leq \tau$ for $g_n^{(d-1)}(x) > \delta_0$. Therefore, it holds that

$$1 - g_n^{(d)}(x) \leq \max\left((1 - x)^{e_n^d}, \tau\right)$$

for any $d > 0$. Applying $d = d_g := \left\lceil \frac{1}{\log e_n} \cdot \log\left(\log\left(\frac{1}{\tau}\right)/\epsilon\right)\right\rceil$, we finally obtain $1 - g_n^{(d_g)}(x) \leq \tau$ since $(1 - x)^{e_n^{d_g}} \leq (1 - \epsilon)^{\log\left(\frac{1}{\tau}\right)/\epsilon} < \tau$. $\square$

The important point is that d_g is reduced as approximately *half* (applying the first heuristic property of g_n) compared to the previous case that only uses f_n to approximate the sign function. Since g_n and f_n requires same number of non-scalar multiplications, we can conclude that the computational complexity of $f_n^{(d_f)} \circ g_n^{(d_g)}$ is $L\left(\frac{a_n}{2}, b_n\right)$ where a_n and b_n are defined from $TC_n = L(a_n, b_n)$.

The total depth of $f_n^{(d_f)} \circ g_n^{(d_g)}$ is $L\left(\frac{log n + O(1)}{2 \cdot \log c_n}, \frac{\log n + O(1)}{\log(n+1)}\right)$ which gets close to $L(1,1)$ as n increases[g]. Note that $L(1,1)$ is theoretically the minimal depth obtained by minimax polynomial approximation (see Section 2.2).

4 Application to Min/Max

As described in [13], min/max functions correspond to the absolute function as

$$\min(a,b) = \frac{a+b}{2} - \frac{|a-b|}{2} \quad \text{and} \quad \max(a,b) = \frac{a+b}{2} + \frac{|a-b|}{2}.$$

Therefore, an approximate polynomial of $|x|$ directly gives us the approximate polynomial of min/max functions. Since $|x| = x \cdot \mathrm{sgn}(x)$, we can consider the convergence of $x \cdot f_n^{(d)}(x)$ to $|x|$ as an analogue. As $\min(a,b)$ is directly computed from $\max(a,b)$, we only describe an algorithm of max for convenience.

Contrary to $\mathrm{sgn}(x)$, the absolute function $|x|$ is continuous so that the parameter ϵ is unnecessary. The following theorem provides the convergence rate of $x \cdot f_n^{(d)}(x)$ to $|x|$.

Theorem 5 (Convergence of $x \cdot f_n^{(d)}$). *If $d \geq \frac{1}{\log c_n} \cdot (\alpha - 1)$, then the error of $x \cdot f_n^{(d)}(x)$ compared to $|x|$ is bounded by $2^{-\alpha}$ for any $x \in [-1,1]$.*

Proof. Since $|x| = x \cdot \mathrm{sgn}(x)$, the error is upper bounded as

$$\left| x \cdot f_n^{(d)}(x) - |x| \right| = |x| \cdot \left| f_n^{(d)}(x) - \mathrm{sgn}(x) \right| \leq |x| \cdot |1 - |x||^{c_n^d}.$$

Let $y = |x| \in [0,1]$ and $k = c_n^d$, then the error upper bound is expressed as $E(y) = y \cdot (1-y)^k$. By a simple computation, one can check that $E(y)$ has the maximal value at $y = 1/(k+1)$. Therefore, k should satisfy

$$E\left(\frac{1}{k+1}\right) = \frac{k^k}{(k+1)^{k+1}} \leq 2^{-\alpha}.$$

Since $2 \leq (1 + 1/k)^k \leq e$ for $k \geq 1$, setting $k \geq 2^{\alpha-1}$ implies $d \geq \frac{1}{\log c_n} \cdot (\alpha - 1)$. □

We denote an algorithm which evaluates $\frac{a+b}{2} + \frac{a-b}{2} \cdot f_n^{(d)}(a-b)$ by `NewMax` (see Algorithm 4), and Theorem 5 is naturally transformed into the context of min/max as Corollary 4.

Corollary 4. *If $d \geq \frac{1}{\log c_n} \cdot (\alpha - 2)$, then the error of the output of `NewMax`$(a, b; n, d)$ compared to the true value is bounded by $2^{-\alpha}$ for any $a, b \in [0,1]$.*

[g]It does *not* mean the "convergence" to $L(1,1)$ as $n \to \infty$, since n should be $O(1)$ with respect to α and $1/\epsilon$.

Algorithm 4 `NewMax`$(a, b; n, d)$

```
Input: $a, b \in [0, 1]$, $n, d \in \mathbb{N}$
Output: An approximate value of $\max(a, b)$
1: $x \leftarrow a - b$, $y \leftarrow \frac{a+b}{2}$
2: for $i \leftarrow 1$ to $d$ do
3:     $x \leftarrow f_n(x)$                                   // compute $f_n^{(d)}(a - b)$
4: end for
5: $y \leftarrow y + \frac{a-b}{2} \cdot x$
6: return $y$
```

Our Max *v.s.* Previous Max. In [13], Cheon et al. introduced a max algorithm exploiting the same identity $\max(a, b) = \frac{a+b}{2} + \frac{|a-b|}{2}$, but they interpret the absolute function as $|x| = \sqrt{x^2}$ which is different with our our interpretation $|x| = x \cdot \text{sgn}(x)$. To compute $\sqrt{(a-b)^2}$, they exploit Wilkes's algorithm [47] denoted by `Sqrt`$(y; d)$ which approximately computes $\sqrt{y}$ for $y \in [0, 1]$: Let $a_0 = y$ and $b_0 = y - 1$, and iteratively compute $a_{n+1} = a_n \left(1 - \frac{b_n}{2}\right)$ and $b_{n+1} = b_n^2 \left(\frac{b_n - 3}{4}\right)$ for $0 \le n \le d - 1$, where the final output is a_d.

We note that the output of `Sqrt`$(x^2; d)$ equals to $x \cdot f_1^{(d)}(x)$, which means our max algorithm `NewMax`$(a, b; 1, d)$ (in the case of $n = 1$) gives the same output to the max algorithm in [13]. However, there are several significant advantages to use our max algorithm instead of the max algorithm in [13].

- `Sqrt`$(x^2; d)$ requires 3 multiplications including 1 square multiplication for each iteration, while $f_1(x)$ can be computed by only 2 multiplications. Therefore, `NewMax`$(\cdot, \cdot; 1, d_1)$ is faster than the max algorithm in [13].
- We can further optimize our max algorithm by substituting $f_1(x)$ with $f_n(x)$ for some $n > 1$. As an analogue of Section 3.4, we can select an optimal n which minimizes $d \cdot C_n$ where $d = \frac{1}{\log c_n} \cdot (\alpha - 2)$, where $n = 4$ is optimal.
- Applying the approximate HE scheme HEAAN [11, 12], the max algorithm in [13] is unstable when two inputs a and b are too close. To be precise, if the input $(a - b)^2$ is close to zero and even smaller than an error accompanied by HEAAN, then the input attached with the error can be a negative value. However, the output of `Sqrt`$(y; d)$ for $y < 0$ diverges as d increases. In contrary, $f_n^{(d)}$ is stable over the interval $[-1, 1]$, so our max algorithm still works well even if two inputs are very close.

Applying $\{g_n\}_{n\ge1}$ to Max. As a construction of `NewCompG`, we can also apply the family of polynomials $\{g_n\}_{n\ge1}$ with heuristic properties to accelerate our `NewMax` algorithm. We denote an algorithm which evaluates $\frac{a+b}{2} + \frac{a-b}{2} \cdot f^{(d_f)} \circ g^{(d_g)}(a - b)$ by `NewMaxG`$(a, b; n, d_f, d_g)$. Applying $\epsilon = 2^{-\alpha}$ to Corollary 3, one can easily obtain the following result on `NewMaxG`.

Corollary 5. *If $d_g \ge \frac{1}{\log g_n'(0)} \cdot \alpha + O(1)$ and $d_f \ge \frac{1}{\log n} \cdot \log(\alpha - 2) + O(1)$, then the error of the output of* `NewMaxG`$(a, b; n, d_f, d_g)$ *compared to the true value is bounded by* $2^{-\alpha}$.

5 Experimental Results

We measured the performance of our algorithms with comparison to `Comp` or `Max` of [13]. The experiments are divided into two categories: 1. Running algorithms on plain inputs, 2. Running algorithms on encrypted inputs. All experiments were conducted on Linux with Intel Xeon CPU at 2.10GHz processor with 8 threads. For experiments in an encrypted state, we used HEAAN library [12, 44].

5.1 Approximate HE Scheme HEAAN

Cheon et al. [12] proposed an HE scheme HEAAN which supports approximate computations of real/complex numbers. Let N be a power-of-two integer and L be the bit-length of initial ciphertext modulus, and define $q_\ell = 2^\ell$ for $1 \leq \ell \leq L$. For $R = \mathbb{Z}[X]/(X^N + 1)$ and $R_q := R/qR$, let $\chi_{\texttt{key}}$, $\chi_{\texttt{err}}$ and $\chi_{\texttt{enc}}$ be distributions over R. A (field) isomorphism $\tau : \mathbb{R}[X]/(X^N+1) \to \mathbb{C}^{N/2}$ is applied for encoding/decoding of plaintexts.

- $\underline{\texttt{KeyGen}(N, L, D)}$.
  - Sample $s \leftarrow \chi_{\texttt{key}}$. Set the secret key as $\mathsf{sk} \leftarrow (1, s)$.
  - Sample $a \leftarrow U(R_{q_L})$ and $e \leftarrow \chi_{\texttt{err}}$. Set $\mathsf{pk} \leftarrow (-a \cdot s + e, a) \in R_{q_L}^2$.
  - Sample $a' \leftarrow U(R_{q_L^2})$ and $e' \leftarrow \chi_{\texttt{err}}$, and set $\mathsf{evk} \leftarrow (b' = -a' \cdot s + e' + q_L \cdot s^2, a') \in R_{q_L^2}^2$.
- $\underline{\texttt{Enc}_{\mathsf{pk}}(\boldsymbol{m}; \Delta)}$.
  - For a plaintext $\boldsymbol{m} = (m_0, ..., m_{N/2-1})$ in $\mathbb{C}^{N/2}$ and a scaling factor $\Delta = 2^p > 0$, compute a polynomial $\mathfrak{m} \leftarrow \lfloor \Delta \cdot \tau^{-1}(\boldsymbol{m}) \rceil \in R$
  - Sample $v \leftarrow \chi_{\texttt{enc}}$ and $e_0, e_1 \leftarrow \chi_{\texttt{err}}$. Output $\mathsf{ct} = [v \cdot \mathsf{pk} + (\mathfrak{m} + e_0, e_1)]_{q_L}$.
- $\underline{\texttt{Dec}_{\mathsf{sk}}(\mathsf{ct}; \Delta)}$.
  - For a ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{q_\ell}^2$, compute $\mathfrak{m}' = [c_0 + c_1 \cdot s]_{q_\ell}$.
  - Output a plaintext vector $\boldsymbol{m}' = \Delta^{-1} \cdot \tau(\mathfrak{m}') \in \mathbb{C}^{N/2}$.
- $\underline{\texttt{Add}(\mathsf{ct}, \mathsf{ct}')}$. For $\mathsf{ct}, \mathsf{ct}' \in R_{q_\ell}^2$, output $\mathsf{ct}_{\texttt{add}} \leftarrow [\mathsf{ct} + \mathsf{ct}']_{q_\ell}$.
- $\underline{\texttt{Mult}_{\mathsf{evk}}(\mathsf{ct}, \mathsf{ct}')}$. For $\mathsf{ct} = (c_0, c_1), \mathsf{ct}' = (c_0', c_1') \in \mathcal{R}_{q_\ell}^2$, let $(d_0, d_1, d_2) = (c_0c_0', c_0c_1' + c_1c_0', c_1c_1')$. Compute $\mathsf{ct}'_{\texttt{mult}} \leftarrow [(d_0, d_1) + \lfloor q_L^{-1} \cdot d_2 \cdot \mathsf{evk} \rceil]_{q_\ell}$, and output $\mathsf{ct}_{\texttt{mult}} \leftarrow [\lfloor \Delta^{-1} \cdot \mathsf{ct}'_{\texttt{mult}} \rceil]_{q_{\ell-p}}$.

The secret key distribution $\chi_{\texttt{key}}$ is set to be $\mathcal{HWT}_N(256)$, which uniformly samples an element with ternary coefficients in R that has 256 non-zero coefficients.

5.2 Parameter Selection

We have two parameters α and ϵ which measure the quality of our comparison algorithms. In our experiments, we set $\epsilon = 2^{-\alpha}$, which is the case expecting that input and output of algorithms have the same precision bits.

HEAAN Parameters. We fix the dimension $N = 2^{17}$, then we can set the initial ciphertext modulus q_L upto 2^{2250} to achieve 128-bit security estimated by

Albrecht's LWE estimator [1, 2]. In each experiment, we set the initial modulus such that the modulus bit after each algorithm is $\log \Delta + 10$. For example, on our comparison algorithm `NewComp`$(\cdot, \cdot; n, d)$, we set the initial modulus bit as

$$\log q_L = (\log \Delta \cdot \lceil \log(2n+1) \rceil + 2n - 1) \cdot d + \log \Delta + 10.$$

Note that each coefficient of f_n is of the form $m/2^{2n-1}$ for $m \in \mathbb{Z}$ (Section 3.1). We progress the scalar multiplication of $m/2^{2n-1}$ in an encrypted state by m additions and $(2n-1)$-bit scaling down which results in the factor $(2n-1)$ in the above equation. In the case of `NewCompG`$(\cdot, \cdot; n, d_f, d_g)$, we similarly set

$$\log q_L = \log \Delta \cdot \lceil \log(2n+1) \rceil \cdot (d_f + d_g) + (2n-1) \cdot d_f + 10 \cdot d_g + \log \Delta + 10.$$

The bit-length of the scaling factor Δ is set to be around 40 as in [13].

Note that one can evaluate $N/2$ comparison functions simultaneously in a single homomorphic comparison. In this sense, an amortized running time of our algorithm is obtained by dividing the total running time by $N/2 = 2^{16}$.

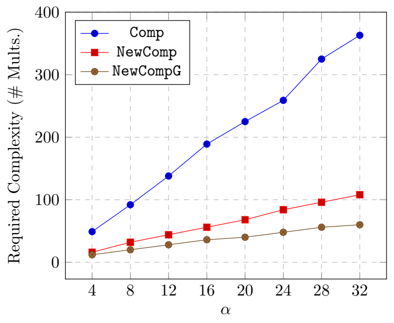
Choice of n in $\{f_n\}_{n \geq 1}$ and $\{g_n\}_{n \geq 1}$. One should consider a different cost model other than TC_n in the case of experiments in an encrypted state. When running our algorithms with HEAAN, not only the complexity TC_n but also the depth TD_n is an important factor affecting the running time, since the computational cost of a homomorphic multiplication is different for each level. Instead of TC_n, we take another cost model $TD_n \cdot TC_n$ considering that a multiplication in R_q takes (quasi-)linear time with respect to $\log q$. Under the setting $\epsilon = 2^{-\alpha}$, one can check by simple computation that $n = 4$ also minimizes $TD_n \cdot TC_n$ as well as TC_n, and we used f_n and g_n with $n = 4$ for the experiments.

5.3 Performance of `NewComp` and `NewCompG`

We compare the performance of our new comparison algorithms `NewComp` and `NewCompG` with the previous comparison algorithm `Comp` proposed in [13]. The following experimental results show that `NewComp` is much faster than `Comp` in practice, and applying g_n polynomials (`NewCompG`) substantially improves the performance of `NewComp`.

Plain State Experiment. For "plain inputs" $a, b \in [0, 1]$ satisfying $|a - b| \geq \epsilon = 2^{-\alpha}$, we measured the required computational complexity and depth of each comparison algorithm to obtain an approximate value of $\text{comp}(a, b)$ within $2^{-\alpha}$ error. The parameters d, d_f and d_g are chosen as the lower bounds described in Corollary 1 and Corollary 3, and we checked that these theoretical lower bounds are indeed very close to those obtained experimentally.

From Figure 6, we can see that `NewComp` requires much less depth and complexity than `Comp`, and those of `NewCompG` are even smaller. Note that the gap between these algorithms in terms of both depth and complexity grows up as α increases. For example, when $\alpha = 8$, the required complexity is $\times$3–4 less in `NewComp` and `NewCompG`; when $\alpha = 32$, it is over $\times 7$ less in `NewCompG`.



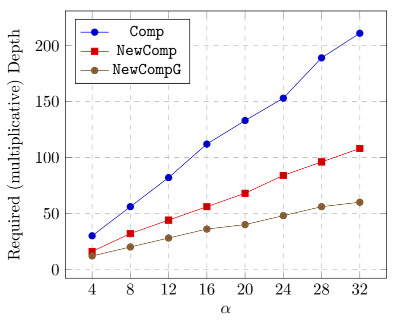


Fig. 6: Comp, NewComp and NewCompG on various α with $\epsilon = 2^{-\alpha}$ in a plain state

Encrypted State Experiment. We also measured the performance of our algorithms which ouput an approximate value of $\mathrm{comp}(a, b)$ within $2^{-\alpha}$ error for "encrypted inputs" $a, b \in [0, 1]$ satisfying $|a - b| \geq \epsilon$. Note that parameters d, d_f and d_g are chosen as the lower bounds in Corollary 1 and 3. We checked through 100 experiments that our algorithms with chosen parameters give accurate results in spite of errors accompanied by HEAAN.

In Table 3, we can see the running time (and amortized running time) of our algorithms NewComp, NewCompG, and that of Comp ([13]) for various α. Note that our new algorithms NewComp and NewCompG provide outstanding performance in terms of amortized running time: NewComp takes 0.9 milliseconds for 8-bit comparison, and NewCompG only takes about 1 millisecond to compare up to 20-bit inputs. It is a significant improvement over the previous algorithm Comp. For example, NewCompG is about $\times 8$ faster than Comp when $\alpha = 8$, about $\times 18$ faster when $\alpha = 16$, and the ratio increases as α increases.

Note that the required depth of Comp is much larger than that of our algorithms as described in Figure 6. Consequently, to run Comp for $\alpha \geq 10$ in an encrypted state with 128-bit security, one must increase the HEAAN parameter $N = 2^{17}$ to $N = 2^{18}$, or use bootstrapping techniques [11], both of which yields more than twice performance degradation, especially in total running time.

α	Comp	NewComp	NewCompG
8	238 s (3.63 ms)	**59 s (0.90 ms)**	**31 s (0.47 ms)**
12	572 s (8.73 ms)*	**93 s (1.42 ms)**	**47 s (0.72 ms)**
16	1429 s (21.8 ms)*	**151 s (2.30 ms)**	**80 s (1.22 ms)**
20	2790 s (42.6 ms)*	**285 s (4.35 ms)***	**94 s (1.43 ms)**

Table 3: Running time (amortized running time) of Comp, NewComp and NewCompG on HEAAN for various α and $\epsilon = 2^{-\alpha}$; an asterisk (*) means that the parameter for HEAAN does not achieve 128-bit security due to large $\log q_L \geq 2250$.

5.4 Performance of NewMax and NewMaxG

We also compared the performance of NewMax and NewMaxG in an encrypted state to that of the max algorithm Max in the previous work [13]. The parameters d, d_f and d_g were chosen from the theoretical lower bounds described in Corollary 4 and Corollary 5, and were confirmed that they are very close to those obtained experimentally. In Figure 7, we can see the running time of our new algorithms NewMax, NewMaxG, and that of Max in [13]. Our algorithms improve the Max considerably in running time (and depth), and the gap increases for larger α: when $\alpha = 8$, our NewMax and NewMaxG algorithms are $\times 1.6$ and $\times 2$ faster than Max, respectively; when $\alpha = 20$, our NewMaxG algorithm is $\times 4.5$ faster than Max.

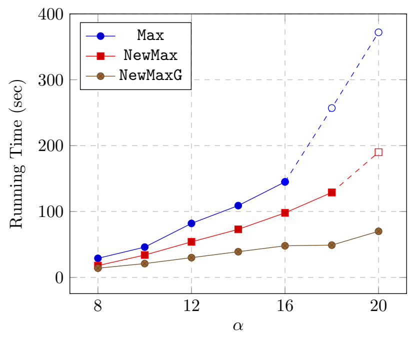


Fig. 7: Running Time of Max, NewMax and NewMaxG on HEAAN for various α. Hollow marker implies that the parameter for HEAAN does not achieve 128-bit security due to large $\log q_L \geq 2250$, which can be achieved by setting $N = 2^{18}$

References

1. M. R. Albrecht. A Sage Module for estimating the concrete security of Learning with Errors instances., 2017. https://bitbucket.org/malb/lwe-estimator.
2. M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 2015.
3. V. Andrievskii. Polynomial approximation of piecewise analytic functions on a compact subset of the real line. *Journal of Approximation Theory*, 2009.
4. F. Armknecht, C. Boyd, C. Carr, K. Gjøsteen, A. Jäschke, C. A. Reuter, and M. Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Report 2015/1192, 2015.
5. J.-C. Bajard, P. Martins, L. Sousa, and V. Zucca. Improving the efficiency of svm classification with fhe. *IEEE Transactions on Information Forensics and Security*, 2019.
6. C. Boura, N. Gama, and M. Georgieva. Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning. Cryptology ePrint Archive, Report 2018/758, 2018.
7. Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

8. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *Proc. of ITCS*, pages 309–325. ACM, 2012.
9. H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 34–54. Springer, 2019.
10. J. H. Cheon, K. Han, S. M. Hong, H. J. Kim, J. Kim, S. Kim, H. Seo, H. Shim, and Y. Song. Toward a secure drone system: Flying with real-time homomorphic authenticated encryption. *IEEE Access*, 2018.
11. J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. Bootstrapping for approximate homomorphic encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 360–384. Springer, 2018.
12. J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.
13. J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 415–445. Springer, 2019.
14. J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In *International Conference on Financial Cryptography and Data Security*, pages 142–159. Springer, 2015.
15. D. Chialva and A. Dooms. Conditionals in homomorphic encryption and machine learning applications. Cryptology ePrint Archive, Report 2018/1032, 2018.
16. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–33. Springer, 2016.
17. I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 377–408. Springer, 2017.
18. D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2002.
19. A. Cordero, F. Soleymani, J. R. Torregrosa, and M. Z. Ullah. Numerically stable improved chebyshev–halley type schemes for matrix sign function. *Journal of Computational and Applied Mathematics*, 318:189–198, 2017.
20. C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 1995.
21. L. Ducas and D. Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.
22. A. Eremenko and P. Yuditskii. Uniform approximation of sgn x by polynomials and entire functions. *Journal d'Analyse Mathématique*, 101(1):313–324, 2007.
23. J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.
24. J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
25. J. H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.
26. C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. http://crypto.stanford.edu/craig.

27. C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.
28. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 2016.
29. R. E. Goldschmidt. *Applications of division by convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.
30. K. Han, S. Hong, J. H. Cheon, and D. Park. Logistic regression on homomorphic encrypted data at scale. The AAAI Conference on Innovative Applications of Artificial Intelligence, 2019.
31. K. Han and D. Ki. Better bootstrapping for approximate homomorphic encryption. Cryptology ePrint Archive, Report 2019/688, 2019. To Appear in CT-RSA 2020.
32. J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1979.
33. N. J. Higham. *Functions of matrices: theory and computation*. Siam, 2008.
34. D. K. Kazarinoff. On wallis' formula. *Edinburgh Mathematical Notes*, 1956.
35. C. S. Kenney and A. J. Laub. The matrix sign function. *IEEE Transactions on Automatic Control*, 40(8):1330–1348, 1995.
36. A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC Medical Genomics*, 11(4):83, Oct 2018.
37. D. Kim, Y. Son, D. Kim, A. Kim, S. Hong, and J. H. Cheon. Privacy-preserving approximate gwas computation based on homomorphic encryption. Cryptology ePrint Archive, Report 2019/152, 2019.
38. M. Kim, Y. Song, B. Li, and D. Micciancio. Semi-parallel logistic regression for gwas on encrypted data. Cryptology ePrint Archive, Report 2019/294, 2019.
39. Y. Lin. A note on margin-based loss functions in classification. *Statistics & probability letters*, 68(1):73–82, 2004.
40. D. S. Mitrinović, J. E. Pečarić, and A. Fink. Bernoulli's inequality. In *Classical and New Inequalities in Analysis*, pages 65–81. 1993.
41. Y. Nakatsukasa, Z. Bai, and F. Gygi. Optimizing halley's iteration for computing the matrix polar decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(5):2700–2720, 2010.
42. M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.
43. E. Saff and V. Totik. Polynomial approximation of piecewise analytic functions. *Journal of the London Mathematical Society*, 2(3):487–498, 1989.
44. snucrypto. HEAAN. `https://github.com/snucrypto/HEAAN`, 2017.
45. A. R. Soheili, F. Toutounian, and F. Soleymani. A fast convergent numerical method for matrix sign function with application in sdes. *Journal of Computational and Applied Mathematics*, 282:167–178, 2015.
46. B. H. M. Tan, H. T. Lee, H. Wang, S. Q. Ren, and K. M. M. Aung. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. Cryptology ePrint Archive, Report 2019/332, 2019.
47. M. V. Wilkes. *The Preparation of Programs for an Electronic Digital Computer: With special reference to the EDSAC and the Use of a Library of Subroutines*. Addison-Wesley Press, 1951.

A Convergence of δ_0, S and $g_{n,\tau}$

It is trivial that $S \leq \frac{\tau}{2}$. Let us denote S, δ_0 and $g_{n,\tau}$ updated in the i-th iteration by S_i, $\delta_{0,i}$ and $g_{n,\tau,i}$ respectively. Assume that $S_i < \frac{\tau}{2}$ for some $i \geq 1$. Then it holds that $g_{n,\tau,i}(x) \geq (1 - \frac{\tau}{2}) - S_i > 1 - \tau$ for $x \in [\delta_{0,i}, 1]$. Therefore, $\delta_{0,i+1}$ should be smaller than $\delta_{0,i}$, and hence S_{i+1} is larger than S_i. Since $\delta_{0,i}$ has a lower bound 0, $\delta_{0,i}$ converges to some constant $\delta_{conv} > 0$ as i increases. Hence, $g_{n,\tau,i}$ converges to some $g_{n,\tau}^{conv}$, and S_i converges to some $S_{conv} \leq \frac{\tau}{2}$.

Now, assume that $S_{conv} < \frac{\tau}{2}$ and let $\rho = \frac{\tau}{2} - S_{conv} > 0$. Since $\delta_{0,i}$ converges (and decreases) to δ_{conv}, there exists some $i \geq 1$ such that $\delta_{0,i} < \frac{1-\tau+\rho}{1-\tau} \cdot \delta_{conv}$. Note that $g_{n,\tau,i}$ is concave in $[0, \delta_{0,i}]$ as noted in Section 2.2. Therefore, it holds that $\frac{g_{n,\tau,i}(\delta_{0,i})-(1-\tau)}{\delta_{0,i}-\delta_{0,i+1}} < \frac{g_{n,\tau,i}(\delta_{0,i})}{\delta_{0,i}}$ where $g_{n,\tau,i}(\delta_{0,i+1}) = 1-\tau$. Since $g_{n,\tau,i}(\delta_{0,i}) - (1-\tau) \geq \rho$, we obtain

$$\begin{aligned}\delta_{0,i} - \delta_{0,i+1} > \frac{g_{n,\tau,i}(\delta_{0,i}) - (1-\tau)}{g_{n,\tau,i}(\delta_{0,i})}\delta_{0,i} &= \delta_{0,i} - \frac{1-\tau}{g_{n,\tau,i}(\delta_{0,i})}\delta_{0,i} \\ &\geq \delta_{0,i} - \frac{1-\tau}{1-\tau+\rho}\delta_{0,i} = \frac{\rho}{1-\tau+\rho}\delta_{0,i}.\end{aligned}$$

Hence, we get $\delta_{0,i} > \frac{1-\tau+\rho}{1-\tau} \cdot \delta_{0,i+1} \geq \frac{1-\tau+\rho}{1-\tau} \cdot \delta_{conv}$, which is a contradiction.

B Heuristic Properties on g_n

We provide experimental results validating the heuristic properties in Section 3.5:

1. $g_n'(0) \simeq 0.98 \cdot f_n'(0)^2$ (Hence, $\log g_n'(0) \simeq 2 \cdot \log c_n$)
2. $1 - g_n(x) \leq (1-x)^{g_n'(0)}$ for $x \in [0, \delta_0]$ where δ_0 is the minimal δ in Prop IV

On the First Heuristic. Using MATLAB, we computed $g_n'(0)$ and compared it with $f_n'^2(0)$ derived from Lemma 2. See Figure 8 for $1 \leq n \leq 20$.

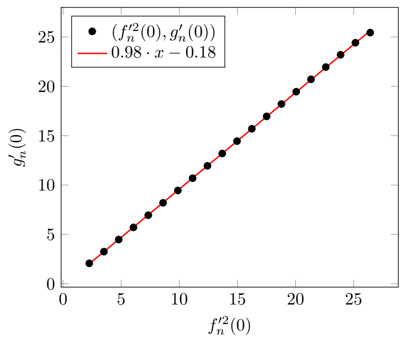
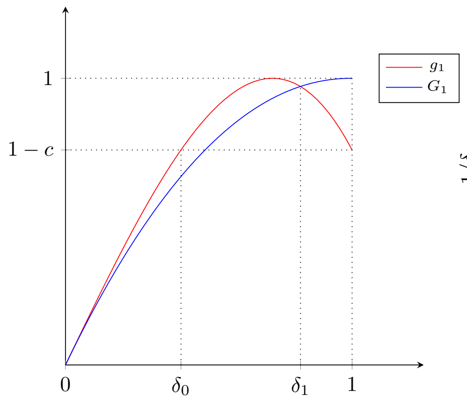
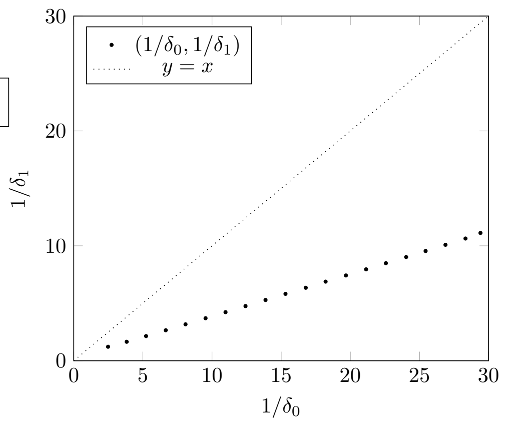


Fig. 8: $f_n'^2(0)$ and $g_n'(0)$ ($R^2 = 0.9999$); $n = 1, 2, ..., 20$ from the left to the right

On the Second Heuristic. Let $G_n(x) := 1 - (1-x)^{g_n'(0)}$, then we can experimentally check that $G_n(x) \leq g_n(x)$ when $x \in (0, \delta_0]$, which is equivalent to $1 - g_n(x) \leq (1-x)^{g_n'(0)}$. Let δ_1 be the largest δ such that $G_n(x) \leq g_n(x)$ for all $x \in [0, \delta]$ (see Figure 9a). The experiment results show that $1/\delta_0 > 1/\delta_1$ which is equivalent to $\delta_0 < \delta_1$ (see Figure 9b for $1 \leq n \leq 20$).



(a) Description of δ_0, δ_1, G_1, and g_1



(b) $1/\delta_0$ and $1/\delta_1$; $n = 1, 2, ..., 20$ from the left to the right

Fig. 9: Experimental evidence on $1 - g_n(x) \leq (1-x)^{g_n'(0)}$ when $x \in (0, \delta_0]$