Polynomial Approximation of Inverse sqrt Function for FHE

Samanvaya Panda

International Institute of Information Technology, Hyderabad, India
samanvaya.panda@research.iiit.ac.in

Abstract. Inverse sqrt and sqrt function have numerous applications in linear algebra and machine learning such as vector normalisation, eigenvalue computation, dimensionality reduction, clustering, etc. This paper presents a method to approximate and securely perform the inverse sqrt function using CKKS homomorphic encryption scheme. Since the CKKS homomorphic scheme allows only computation of polynomial functions, we propose a method to approximate the inverse sqrt function polynomially. In the end, we provide an implementation of our method for the inverse sqrt function.

Keywords: Polynomial Approximation, Inverse sqrt, Homomorphic encryption, CKKS

1 Introduction

Privacy-preserving computation has been used to mitigate personal and sensitive information leakage problems. There are many ways to compute functions on data securely, keeping the data private. One such technique is homomorphic encryption. Homomorphic encryption allows us to evaluate any function on encrypted data without knowing the actual data. In recent times, there have been numerous advancements in homomorphic encryption schemes. The CKKS homomorphic encryption scheme proposed recently by Cheon et al. in [3] is one such example. It supports arithmetic on floating-point numbers which is why it has been extensively used for different machine learning tasks [1, 5, 6].

The problem with CKKS homomorphic scheme is that it only supports polynomial operations. So, we can't implement many non-polynomial functions such as logarithm, sqrt, sine, etc. directly. These functions need to be polynomially approximated. There have been several methods proposed to approximate non-polynomial functions, such as using Chebyshev's polynomial basis [13], minimax polynomials [12], Fourier series, etc. in [2–4, 14]. But in most FHE-based algorithms using schemes similar to CKKS, the approximation of inverse functions is skipped and such computations are performed on plaintext. Few attempts have been made in this direction in [14] and [9]. In [14], authors suggested approximating division operation using Newton's method and Goldschmidt's algorithm. They used an initial guess $y_0 = 2^{1-k}$ for all values. In [9] too, the author used

Newton's method and Goldschmidt's algorithm to approximate the inverse sqrt and sqrt function simultaneously. They took inspiration from the fast inverse sqrt algorithm [7] and tried a similar approach in a homomorphic setting. Instead of fixing the initial guess, they used constrained linear regression to approximate the inverse sqrt function. The line served as a good initial guess for larger values of x. But for smaller values, it requires more iterations for convergence.

1.1 Contributions

We also draw inspiration from the fast inverse sqrt algorithm [7]. We can have some curve approximating the $\frac{1}{\sqrt{x}}$ as a good initial guess and then use Newton's iteration to improve upon that guess. This paper proposes a new method to find a better initial guess for the inverse sqrt function and apply Newton's iterations. The advantage of using Newton's iterations for inverse functions is that they are polynomial and can be evaluated homomorphically. Upon observing the shape of the inverse sqrt function, we notice that the value of the function keeps decreasing slowly to the right of $x = 1$ and increases drastically to the left of $x = 1$. Over a considerable interval, we can see that the shape of the function looks like the 'L' alphabet.

This gave us the intuition to approximate the curve $\frac{1}{\sqrt{x}}$ using two lines. One of the lines approximates the curve over a large interval capturing the *slow* decreasing trend of the values. The other line captures the *rapid* increasing trend in the values of the curve. The intersection of the two lines is called the pivot point. Approximation of the curve $\frac{1}{\sqrt{x}}$ can be written as a convex combination of the two lines about the pivot point using a sign function as described in [4]. In sections ahead, we will define how to find those two lines and what properties they must satisfy. Our method provides sufficient accuracy with multiplicative depth[1] comparable to the approximation in [9] and also reduces the number of iterations almost by half. Finally, we provide experimental results on the homomorphic implementation of our method.

2 Preliminaries

2.1 CKKS Homomorphic Scheme

The CKKS(Cheon-Kim-Kim-Song) scheme [3] is a leveled homomorphic encryption scheme. Unlike other HE schemes, CKKS supports approximate arithmetic on real and complex numbers with predefined precision. The main idea behind the CKKS scheme is that it treats noise generated upon decryption as an error in computation for real numbers. This makes it ideal for performing machine learning tasks where most of the calculations are approximate. The CKKS scheme becomes an FHE(fully homomorphic encryption) scheme with the bootstrapping technique.

[1] We consider non-scalar multiplicative depth i.e ciphertext-ciphertext multiplication

Let $N = \phi(M)$ be the degree of the M-*th* cyclotomic polynomial $\Phi_M(X)$. If N is chosen as a power of 2 then $M = 2N$ and the M-*th* cyclotomic polynomial $\Phi_M(X) = X^N + 1$. Let $\mathcal{R} = \mathbb{Z}[X]/\Phi_M(X) = \mathbb{Z}[X]/(X^N + 1)$ be the ring of polynomials defined for the plaintext space. Let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[X]/(X^N + 1)$ be the residue ring defined for the ciphertext space. Let $\mathbb{H}$ be a subspace of $\mathbb{C}^N$ which is isomorphic to $\mathbb{C}^{N/2}$. Let $\sigma : \mathcal{R} \to \sigma(\mathcal{R}) \subseteq \mathbb{H}$ be a canonical embedding. Let $\pi : \mathbb{H} \to \mathbb{C}^{N/2}$ be a map that projects a vector from a subspace of $\mathbb{C}^N$ to $\mathbb{C}^{N/2}$.

The CKKS scheme provides the following operations:-

- **KeyGen**(N) :- Generates secret polynomial $s(X)$, public polynomial $p(X)$.
- **Encode**(z) :- Encodes a message vector $z \in \mathbb{C}^{N/2}$ to a message polynomial $m(X) \in \mathcal{R}$ where $m(X) = \sigma^{-1}(\lfloor \Delta \cdot \pi^{-1}(z) \rceil) \in \mathcal{R}$.
- **Decode**$(m(X))$:- Decodes a message polynomial $m(X) \in \mathcal{R}$ back to a message vector $z \in \mathbb{C}^{N/2}$.
- **Encrypt**$(m(X), p(X))$:- Encrypts the message polynomial $m(X) \in \mathcal{R}$ to get ciphertext polynomial $c(X) = (c_0(X), c_1(X)) = (m(X), 0) + p(X) = (m(X) - a(X) \cdot s(X) + e(X), a(X)) \in (\mathbb{Z}_q[X]/(X^N + 1))^2$.
- **Decrypt**$(c(X), s(X))$:- Decrypts the ciphertext polynomial $c(X)$ to the corresponding message polynomial $m(X)$.

Apart from the above operations, it provides an evaluator function that can perform specialised ciphertext operations. This include:-

- **Add**$(c(X), c'(X))$:- to add 2 ciphertext polynomials.
- **Multiply**$(c(X), c'(X))$:- to multiply 2 ciphertext polynomials.
- **Rotate**$(c(X), i)$:- to rotate the ciphertext polynomial by i positions left.

2.2 Polynomial Approximation of Sign Function

The sign function is non-polynomial and can't be used directly in the CKKS scheme. So, we use a polynomial approximation of the sign function instead. In general, we approximate the sign function in the domain $x \in [-1, 1]$ and the sign of any other value can be found by scaling it inside the domain. In this paper, we will use the approximation proposed by Cheon et al. in [4]. We approximate the sign function as a composite polynomial $f^{(d)}$ where f is a polynomial with *similar shape* to the sign function in the interval $[-1, 1]$. The properties satisfied by f are:-

- $f(-x) = -f(x)$ (Origin symmetry)
- $f(1) = 1, f(-1) = -1$ (Range of sign function)
- $f'(x) = c(1-x)^n(1+x)^n$ for some $c > 0$ (Faster convergence)

Evaluating the polynomial f for different values of n we obtain :-

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

Theorem 1. *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* $[-1, 1]$ *implies* $|f_n^{(d)}(x) - sgn(x)| \leq 2^{-\alpha}$ *where* $x \in [-1, -\epsilon] \cup [\epsilon, 1]$.

Proof of theorem 1 can be found in [4]. To speedup up the convergence further, we use a polynomial g instead of f that has a larger derivative at $0(c_n)$. The polynomial g would be a minimax polynomial satisfying the following properties :-

- $g(-x) = -g(x)$ (Origin Symmetry)
- $\exists\, 0 < \delta < 1$ s.t. $x < g(x) < 1\ \forall x \in (0, \delta]$ and $g([\delta, 1]) \subseteq [1 - \tau, 1]$

Its hard to represent g in closed form and is evaluated using algorithm 2 in [4]. In this paper, we use $n = 3$ to approximate the sign function. The approximate sign function is computed as a composition $f_3^{d_f}(x) \circ g_3^{d_g}(x)$ where $d_g = \frac{1}{2log(c_n)} \cdot log(1/\epsilon) = 0.445 \cdot log(1/\epsilon)$ and $d_f = \frac{1}{log(n+1)} \cdot log(\alpha - 1) = 0.5 \cdot log(\alpha - 1)$. The polynomial $f_3(x)$ and $g_3(x)$ are:-

$$f_3(x) = \frac{1}{2^4}(35x - 35x^3 + 21x^5 - 5x^7)$$
$$g_3(x) = \frac{1}{2^{10}}(4589x - 16577x^3 + 25614x^5 - 12860x^7)$$

2.3 Inverse sqrt approximation

In [9], the author mentioned a method to polynomially approximate $\frac{1}{\sqrt{x}}$ that could be used in FHE schemes. It first uses a line as an initial guess and then uses Newton's and Goldschmidt's algorithms to improve upon their guess. Goldschmidt's algorithm is used to compute $\sqrt{x}$ alongside with $\frac{1}{\sqrt{x}}$ which is required for their application. For the initial guess, they perform a linear approximation of $\frac{1}{\sqrt{x}}$ by formulating a constrained linear regression. It is formulated as the following minimization problem :-

$$\begin{aligned} \min_{w} \quad & \frac{1}{n}\sum_{i=1}^{n}(y_i - w^T x_i)^2 \\ \text{subject to} \quad & w^T x_i \geq 0 \quad \forall\, i = \{1, 2, \cdots n\} \end{aligned} \tag{1}$$

3 Approximation of $\frac{1}{\sqrt{x}}$

As mentioned before, we use Newton's method to approximate the value of $y = \frac{1}{\sqrt{x}}$. It is because in each iteration of Newton's method, the update equation is polynomial in nature. Let $f(x) = y^{-2} - x$. Then the update equation for $f(x)$ is :- $y_{i+1} = \frac{y_i}{2}(-xy_i^2 + 3)$. The only thing now to consider is the initial guess for each value.

3.1 A good initial guess

A good initial guess for Newton's update equation would mean faster convergence[2]. So, a good initial guess would be a good approximation of the $\frac{1}{\sqrt{x}}$ function. Another thing to remember is that the initial guess must guarantee convergence. The range of values for which y_i guarantees convergence would be $\frac{y_i}{2}(-xy_i^2+3) > 0 \implies 0 < y_i < \sqrt{\frac{3}{x}}$.

Any value of y_i between 0 and $\sqrt{\frac{3}{x}}$ will eventually converge because the term $-xy_i^2+3$ is always greater than 0 pushing it towards the value $\frac{1}{\sqrt{x}}$. But, we wish that our algorithm would converge in a fixed number of iterations rather than converging eventually. So, we observe that for any x, the number of iterations needed for the initial guess $y_0 \in [\frac{1}{\sqrt{x}}, \sqrt{\frac{3}{x}}]$ to converge increases as we move away from $\frac{1}{\sqrt{x}}$ to $\sqrt{\frac{3}{x}}$. Same is the case for the interval $[0, \frac{1}{\sqrt{x}}]$. For a given number of iterations, we could use binary search on both of the intervals to find the new reduced range for the initial guess that guarantees convergence. To keep things simple and uniform, we assume that the reduced range of initial guess for each x would also be a function of $\frac{1}{\sqrt{x}}$ and the new range would be $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ where $0 < k_1 < 1$ and $1 < k_2 < \sqrt{3}$. Using lemma 1, we show that for constants k_1 and k_2, we can guarantee convergence for all values of x with a certain error.

Lemma 1. *Let d be the given number of Newton's iterations. Let the absolute error at the point $x = 1$ for the initial guess $y_0 = k_1$ or $y_0 = k_2$ after d iterations be $\leq \mathcal{E}$ where $0 < k_1 < 1$ and $1 < k_2 < \sqrt{3}$. Then the absolute error at any point x after d iterations for any initial guess in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ would be $\mathcal{E}_x \leq \frac{\mathcal{E}}{\sqrt{x}}$.*

Proof. Let us first consider the lower bound k_1. At point $x = 1$, we have $y_0 = k_1$. Then, $y_1 = \frac{k_1}{2} \cdot (-k_1^2 + 3)$. Let's say $K_0 = k_1$ and $K_i = \frac{K_{i-1}}{2} \cdot (-K_{i-1}^2 + 3)$. After d iterations we would have $|1 - K_d| \leq \mathcal{E}$. Now for any x, $y_0 = \frac{k_1}{\sqrt{x}}$, $y_1 = \frac{y_0}{2} \cdot (-xy_0^2 + 3) = \frac{k_1}{2\sqrt{x}} \cdot (-k_1^2 + 3) = \frac{K_1}{\sqrt{x}}$. So, after d iterations we get $\mathcal{E}_x = |\frac{1}{\sqrt{x}} - \frac{K_d}{\sqrt{x}}| \leq \frac{\mathcal{E}}{\sqrt{x}}$. Since values of k_1 and k_2 are evaluated for fixed $\mathcal{E}$, we can similarly argue for the upper bound k_2 that it will guarantee convergence for any x with an absolute error $\frac{\mathcal{E}}{\sqrt{x}}$.

Corollary 1. *The mean absolute error over all x in the interval $[a, b]$ after d iterations would be $\bar{\mathcal{E}} = \frac{2\mathcal{E}}{\sqrt{a}+\sqrt{b}}$*

Corollary 2. *If we consider the two intervals $x \in [a, 1] \cup [1, b]$, where a, b are constants and $a < 1 < b$ then the mean absolute error over all x after d iterations would be $\bar{\mathcal{E}}' = \bar{\mathcal{E}}_1 + \bar{\mathcal{E}}_2 = \frac{\mathcal{E}}{1+\sqrt{a}} + \frac{\mathcal{E}}{1+\sqrt{b}}$.*

The corollaries 1 and 2 can be easily verified using Mean value theorem i.e. $\int_a^b f(x)dx = f(c)(b-a)$. The value of k_1 and k_2 is found using the algorithm 1 at $x = 1$ for a fixed number of Newton's iterations d and absolute error $\mathcal{E}$.

[2] By convergence we mean that the difference between the actual and predicted value is bounded by some predefined error

Algorithm 1 Finding constants for initial guess range of $\frac{1}{\sqrt{x}}$

```
Input: $d, \mathcal{E}$: No.of iterations and error.
Output: $k_1$, $k_2$: Upper and lower bound of initial guess.
 1: $l \leftarrow 2\delta - 1$, $r \leftarrow 1$
 2: while $r - l \geq \delta$ do
 3:     $mid \leftarrow (r + l)/2$
 4:     $val \leftarrow mid$
 5:     for $i = 1$ to $d$ do
 6:         $val \leftarrow \frac{val}{2}(-val^2 + 3)$
 7:     end for
 8:     $diff \leftarrow |val - 1|$
 9:     if $diff \leq \mathcal{E}$ then
10:         $r \leftarrow mid$
11:     else
12:         $l \leftarrow mid + \delta$
13:     end if
14: end while
15: $k_1 \leftarrow l$
16: $l \leftarrow 0$, $r \leftarrow 2\sqrt{3} - 1$
17: while $r - l \geq \delta$ do
18:     $mid \leftarrow (r + l)/2$
19:     $val \leftarrow mid$
20:     for $i = 1$ to $d$ do
21:         $val \leftarrow \frac{val}{2}(-val^2 + 3)$
22:     end for
23:     $diff \leftarrow |val - 1|$
24:     if $diff \leq \mathcal{E}$ then
25:         $l \leftarrow mid$
26:     else
27:         $r \leftarrow mid - \delta$
28:     end if
29: end while
30: $k_2 \leftarrow l$
31: return $k_1, k_2$
```

3.2 2-line approximation

Now that we have the range for initial guess, we need to find an approximation of $\frac{1}{\sqrt{x}}$ that lies within this initial guess for all values of x. As mentioned earlier, we approximate the function $\frac{1}{\sqrt{x}}$ using two intersecting lines(L_1, L_2) by limiting the domain of x to $[a, b]$. The intersection point of the lines is called the *pivot* point(denoted as P). Let L_2 approximate $\frac{1}{\sqrt{x}}$ on larger values of x i.e $x \in [P, b]$ and L_1 approximate $\frac{1}{\sqrt{x}}$ on the smaller values of x i.e $x \in [a, P]$. The overall approximation of $\frac{1}{\sqrt{x}}$ can be written as convex combination in terms of L_1 and L_2 as:-

$$h(x) = (1 - \beta(x)) \cdot L_1(x) + \beta(x) \cdot L_2(x) \tag{2}$$

where $\beta(x) = comp(\frac{P}{b-a}, \frac{x}{b-a})$ and $comp(x, y) = \frac{1+sgn(x-y)}{2}$. We can evaluate the sign function polynomially as mentioned in [4]. So far, we have mentioned how to compute the approximate value of $\frac{1}{\sqrt{x}}$ using L_1 and L_2. Now we are going to discuss how to find these lines. Remember that approximate value of $\frac{1}{\sqrt{x}}$ for all x must be in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$ to guarantee convergence. To ensure this, the maximum value for any point x on the line cannot exceed $\frac{k_2}{\sqrt{x}}$. That means both the lines are tangent to the curve $\frac{k_2}{\sqrt{x}}$. Similarly, the minimum value at any point x on the line cannot go below $\frac{k_1}{\sqrt{x}}$. This implies that the extreme points of both the lines must either lie on the curve $\frac{k_1}{\sqrt{x}}$ or above it.

Let x_1, x_2 be the points were lines L_1, L_2 are tangent to the curve $\frac{k_2}{\sqrt{x}}$ respectively. The slope of any tangent to the curve $\frac{k_2}{\sqrt{x}}$ at point γ is $-\frac{1}{2}k_2 \cdot \gamma^{-3/2}$. So, the equation of L_1 and L_2 will be:-

$$L_1 : y = -\frac{1}{2}k_2 x_1^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_1}} \tag{3}$$

$$L_2 : y = -\frac{1}{2}k_2 x_2^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_2}} \tag{4}$$

The last step in figuring out the lines L_1, L_2 are the points x_1, x_2 respectively. They are the points where the lines L_1 and L_2 touch the curve $\frac{k_2}{\sqrt{x}}$. We consider that these lines must pass through the extreme points of the domain of x i.e L_2 must pass through the point $x = b$ and L_1 must pass through $x = a$ on the curve $\frac{k_1}{\sqrt{x}}$. Each of the lines L_1, L_2 also intersect the curve $\frac{k_1}{\sqrt{x}}$ at points different than $x = a$ and $x = b$ respectively. Let the line L_2 pass through a point x on the curve $\frac{k_1}{\sqrt{x}}$. Then the equation of L_2 becomes:-

$$\frac{k_1}{\sqrt{x}} = -\frac{1}{2}k_2 x_2^{-3/2} x + \frac{3}{2}\frac{k_2}{\sqrt{x_2}}$$

$$\implies k_2^2 x^3 - 6k_2^2 x^2 x_2 + 9k_2^2 x x_2^2 - 4k_1^2 x_2^3 = 0 \tag{5}$$

Solving the equation 5 with $x = b$, we would obtain the point x_2 and ultimately L_2. Similarly, we can evaluate x_1 at $x = a$ and get L_1.

3.3 Finding pivot point

In the previous section, we presented a method to obtain the lines L_1, L_2 so that they lie in the range $[\frac{k_1}{\sqrt{x}}, \frac{k_2}{\sqrt{x}}]$. But we aim to approximate the original function $\frac{1}{\sqrt{x}}$ using two intersecting lines in the given range. The above approach for finding the lines doesn't guarantee the intersection of lines L_1 and L_2 inside the desired range i.e P may or may not lie above or on the curve $\frac{k1}{\sqrt{x}}$. One way to ensure that the pivot point lies within the range of convergence is to increase the number of newton's iterations. Increasing the iterations would adjust the values of k_1 and k_2 accordingly, allowing L_1 and L_2 to intersect in the inside region. Another way to ensure that is by tweaking the process of finding the lines L_1, L_2 a little bit. Instead of increasing the number of iterations to adjust the values of k_1 and k_2, we fix the pivot point. We follow the following steps:-

- **Step 1:** Find x_2 using equation 5 at $x = b$ to evaluate L_2.
- **Step 2:** Find the other point of intersection of line L_2 with the curve $\frac{k_1}{\sqrt{x}}$ by solving for x in equation 5. Now this point becomes the pivot point P.
- **Step 3:** Find x_1 using the pivot point. Substitute $x_2 = x_1$ and $x = P$ in equation 5 and solve for x_1.

Note that if we follow the above steps to find lines L_1 and L_2, then they will always intersect at point P. But, the line L_1 no longer intersects the curve

$\frac{k_1}{\sqrt{x}}$ at $x = a$ but rather at $x = a'$ where $a < a'$. This is a trade-off between accuracy and the number of iterations. With practical results, we can argue that a sufficient level of accuracy can be achieved with a few iterations using the above method. Another fact to consider is that we choose to fix the pivot point as the point of intersection of the line L_2 to the curve $\frac{k_1}{\sqrt{x}}$ instead of L_1. This is because the curve $\frac{k_1}{\sqrt{x}}$ is closer to 0 on larger values of x. If we take the pivot point as the other point of intersection of line L_1 then line L_2 would intersect the curve $\frac{k_1}{\sqrt{x}}$ at $x = b'$ where $b' < b$. So, for some values of $x > b'$, the value of L_2 would be negative. When we apply Newton's iterations on these points, there values would converge to $-\frac{1}{\sqrt{x}}$ instead of $\frac{1}{\sqrt{x}}$.

4 Implementation details

We implemented the secure inverse sqrt approximation using the SEAL library [11] for CKKS homomorphic scheme. The implementation can be found at [10]. For the approximate sign function, we fix the value of $d_f = 2$ which would give us the value of $\alpha = 17$. We know that the value of $d_g = 0.445 \cdot log(1/\epsilon)$ where the value of ϵ determines the precision of values for the *comp*() function. For any $z_1, z_2 \in [0, 1], |z_1 - z_2| \geq \epsilon$. The lower the value of ϵ, larger the precision, better accuracy of the approximation and larger the multiplicative depth. For our experiments mentioned in table 1, we fixed $d_g = 7$ making $\epsilon \approx 2 \times 10^{-5}$. The multiplicative depth for both f_3 and g_3 is 3. So, the multiplicative depth required to compute the initial guess would be $3(d_g + d_f) + 1$. The maximum multiplicative depth required in a single Newton's iteration is 2. Hence, the maximum multiplicative depth required to compute the inverse operation would be $3(d_g + d_f) + 2d + 1$. While evaluating the function on encrypted data, we observed that the output of *comp*() was slightly > 1 due to additive noise. So, the points near the tangent exceeded the upper bound. To mitigate this, we introduced a constant error i.e $err = 8.5 \times 10^{-7}$ that was subtracted from the value of the *comp*() so that the final value remained < 1. So on encrypted data, the convex combination of lines becomes - $h(x) = (1 + err - \beta(x)) \cdot L_1(x) + (\beta(x) - err) \cdot L_2(x)$.

To obtain the parameters of pivot-tangent method, we first fix the interval $[a, b]$ and then fix the values of number of iterations d and absolute error $\mathcal{E}$. The smaller the value of $\mathcal{E}$ the smaller the interval for initial guess $[\frac{k1}{\sqrt{x}}, \frac{k2}{\sqrt{x}}]$ would be. To increase the initial guess interval, we have to increase the number of iterations d. So, smaller $\mathcal{E}$ requires larger number of iterations d. We should keep this in mind while fixing d and $\mathcal{E}$. Also, notice that equation 5 is a cubic equation. So, while solving for x_2, we consider the largest root as x_2. Similarly while computing x_1, we consider the smallest root as x_1. For the pivot point P, we consider the root closest to 1 as P. The combined method for the polynomial approximation of $\frac{1}{\sqrt{x}}$ using pivot-tangent method is given in the algorithm 2.

Algorithm 2 Finding approximate value of $\frac{1}{\sqrt{x}}$

```
Input: $[a,b], \mathcal{E}, d, d_g, d_f, x, err$.
Output: $y_d$: Approximate value of $\frac{1}{\sqrt{x}}$.
1: Find $k_1, k_2$ using Algorithm 1.
2: Find $x_2, P, x_1$ using pivot-tangent method.
3: Compute $\beta(P,x)$ and $y_0 = h(x)$.
4: Compute $d$ Newton's iterations to obtain $y_d$.
5: return $y_d$
```

5 Results and Comparison

Table 1 summarizes the value of various parameters computed using the pivot-tangent method with $[a,b] = [10^{-4}, 10^3]$ and $\mathcal{E} = 0.007$. The points are equally divided in two intervals i.e. $[10^{-4}, 1]$ and $[1, 10^3]$. Using corollary 2, we get the theoretical upper bound on error $= \frac{0.007}{1+10^{-2}} + \frac{0.007}{1+\sqrt{1000}} = 0.00714$. We observe that the mean absolute error obtained after the experiments for fixed parameters is lower than theoretical upper bound. Figure 1 shows different components of the pivot tangent method.

d	k_1	k_2	x_1	x_2	P	Mean Abs. Error (without encryption)	Mean Abs. Error (with encryption)	Depth
7	0.128	1.6645	0.3111	343.6645	0.9053	0.00265	0.0055	42
8	0.0855	1.6876	0.1322	340.035	0.3887	0.00083	0.00112	44
9	0.0702	1.6958	0.0876	338.781	0.2587	0.000091	0.0001	46

Table 1: Values of different parameters for given no.of Newton's iterations

Now, to compare our method with the technique of finding $\frac{1}{\sqrt{x}}$ mentioned in [9], we conducted some experiments by fixing the interval for $x \in [a,b]$. For the method in [9], we take d_1, d_2 as the number of Newton's and Goldschmidt's iterations respectively. For the method in [9], the values of (slope,intercept)

$[a,b]$	Iteration		Depth		$\mathcal{E}$	err	Error	
	(d_1,d_2)	(d,d_g)	$2d_1+3d_2$	$2d+3d_g+6$			[9]	Ours
$[10^{-3}, 750]$	(12, 3)	(7, 5)	33	35	7e-3	1.2e-6	8.587e-5	2.23e-4
	(10, 5)	(7, 5)	35	35	1e-3	1.2e-6	8.695e-5	1.01e-4
	(12, 4)	(8, 6)	36	40	1e-4	1.2e-6	8.571e-6	2.23e-5
$[10^{-4}, 10^3]$	(12, 5)	(7, 6)	39	38	7e-3	1.2e-6	1.421e-3	5.65e-3
	(14, 4)	(8, 6)	41	40	7e-3	1.2e-6	2.323e-4	2.98e-3
	(15, 5)	(9, 7)	45	45	1e-4	8.5e-7	5.824e-6	1.87e-5

Table 2: Comparison of our method with that of in [9].

used for the initial guess line in the intervals $[10^{-3}, 750]$ and $[10^{-4}, 10^{3}]$ are $(-0.00019703, 0.14777278)$ and $(-1.29054537e-04, 1.29054537e-01)$ respectively. From table 2, we can see that for similar multiplicative depth, method mentioned in [9] has comparatively lower mean absolute error. Note that our method significantly reduces(almost half) the number of iterations required for convergence(by comparing $d_1 + d_2$ and d in table 2). The biggest bottleneck for our method is the sign function approximation. It takes up more depth than the Newton's iterations i.e $2d < 3(d_g + d_f)$. While the overall pivot-tangent method reduces the number of iterations required for convergence, it wastes most of its multiplicative depth in initially computing the sign approximation. Thus, when we try to compare it with method in [9] in terms of similar multiplicative depth, the method in [9] would get double the number of iterations compared to our method and hence has slightly better results. It is also important to remember that while method in [9] has no convergence guarantees(it may diverge for some values or converge to -$\frac{1}{\sqrt{x}}$ for larger x), our method provides guaranteed convergence with a certain error.

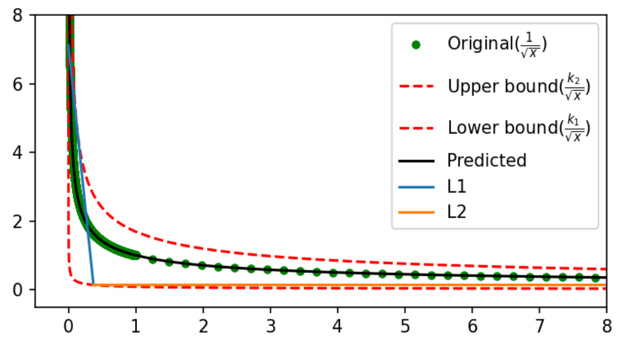

Fig. 1: Inverse sqrt approximation using pivot-tangent method

6 Conclusion and Future Work

In this paper, we presented the pivot-tangent method and approximated $\frac{1}{\sqrt{x}}$ function. Our approximation provides guaranteed convergence with sufficient accuracy compared to the previous method for the same multiplicative depth. Our method reduces the number of iterations required for convergence almost by half. However, the biggest bottleneck for our method is the computation of the approximate sign function. So, a new way to polynomially represent piece-wise lines without relying on sign function would significantly reduce our method's multiplicative depth. It is worth mentioning that in recent times there have been approaches such as Chimera [2] and Pegasus [8] that provide a bridge between different FHEs. This enables us to non-polynomial functions on CKKS ciphertexts. But these transformations are very costly in terms of memory. It would be better to have an inverse sqrt approximation in the CKKS scheme that provides sufficient accuracy and precision over a large interval. We also plan to extend our pivot-tangent method to approximate other inverse functions as a generalized approach. Now that we have a good approximation method of an inverse sqrt function, we also plan to apply this algorithm to different linear algebraic and machine learning algorithms in the future.

References

1. Boemer, F., Costache, A., Cammarota, R., Wierzynski, C.: Ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In: Proceedings of the 7th ACM Workshop on Encrypted Computing and Applied Homomorphic Cryptography. p. 45–56. WAHC'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338469.3358944, https://doi.org/10.1145/3338469.3358944
2. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Cryptology ePrint Archive, Report 2018/758 (2018), https://eprint.iacr.org/2018/758
3. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Report 2016/421 (2016), https://eprint.iacr.org/2016/421
4. Cheon, J.H., Kim, D., Kim, D.: Efficient homomorphic comparison methods with optimal complexity. Cryptology ePrint Archive, Report 2019/1234 (2019), https://ia.cr/2019/1234
5. Han, K., Hong, S., Cheon, J.H., Park, D.: Efficient logistic regression on large encrypted data. Cryptology ePrint Archive, Report 2018/662 (2018), https://eprint.iacr.org/2018/662
6. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., No, J.S.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. Cryptology ePrint Archive, Report 2021/783 (2021), https://ia.cr/2021/783
7. Lomont, C.: Fast inverse square root. Tech. rep., Purdue University (2003), http://www.matrix67.com/data/InvSqrt.pdf
8. Lu, W., Huang, Z., Hong, C., Ma, Y., Qu, H.: Pegasus: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 2021 IEEE Symposium on Security and Privacy (SP). pp. 1057–1073. IEEE Computer Society, Los Alamitos, CA, USA (may 2021). https://doi.org/10.1109/SP40001.2021.00043
9. Panda, S.: Principal component analysis using ckks homomorphic encryption scheme. Cyber Security Cryptography and Machine Learning, 5th International Symposium, CSCML 2021 (2021), https://eprint.iacr.org/2021/914
10. Panda, S.: Pivot-tangent method. https://github.com/pandasamanvaya/Pivot-tangent (2022)
11. Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL (Sep 2021), microsoft Research, Redmond, WA.
12. TASISSA, A.: Function approximation and the remez algorithm (2019)
13. Trefethen, L.N.: Approximation Theory and Approximation Practice, Extended Edition. SIAM (2019)
14. Çetin, G.S., Doröz, Y., Sunar, B., Martin, W.J.: Arithmetic using word-wise homomorphic encryption (2016)