A Number theory approach to square root computation

Anshuman Singh

27/01/2025

Abstract

The computation of square roots remains a cornerstone operation in various fields, with applications ranging from numerical analysis to cryptography. Traditional methods such as Newton's iteration and the bisection method offer robust solutions, but there is significant room for improvement in terms of mathematical efficiency and convergence rates. This paper introduces a novel, mathematically focused algorithm for computation of integer value of square root of integers, built entirely on theoretical principles and devoid of reliance on hardware-specific optimizations. The proposed method aims to reduce iteration complexity while ensuring precision and stability, paving the way for advancements in computational mathematics.

1 Introduction

The calculation of square roots is a fundamental mathematical operation with diverse applications in engineering, physics, and computational sciences. Classic algorithms like Newton-Raphson and binary search are known for their effectiveness, yet they often require numerous iterations to converge, particularly for high-precision computations.

This paper presents a purely mathematical algorithm designed to achieve faster convergence without resorting to hardware-level optimizations. By leveraging advanced mathematical constructs and theoretical insights, the proposed method seeks to find integer value of square roots of integers of very large magnitudes.

The key contributions of this work include:

- the perception of squares as summation of a sequence of odd integers.
- an algorithm to compute the integer value of square roots of all integers numbers.

2 Related Work

The computation of square roots has been extensively studied, with algorithms ranging from classical iterative methods to modern numerical techniques. Newton's method, known for its quadratic convergence, is widely regarded as the benchmark for square root computation. The digit-by-digit algorithm, though less efficient, remains a cornerstone in educational contexts for its intuitive appeal.

3 methodologies

The algorithm is built upon several mathematical insights which are as follows

3.1 result 1

The square of a positive integer N for all N belonging to the set of all positive integers can be expressed as the sum of the first N consecutive odd integers starting from 1. Formally,

$$sum_{i=1}^{N} 2i - 1 \tag{1}$$

where i belongs to the set of all natural numbers.

3.1.1 proof

For a given positive integer N,

$$N^2 = (1 + (N - 1))^2 \tag{2}$$

$$N^2 = 1^2 + (N - 1)^2 + 2 * (N - 1) \tag{3}$$

$$N^2 = 1 + 2 * N - 2 + (N - 1)^2 \tag{4}$$

$$N^2 = (2 * N - 1) + (N - 1)^2 \tag{5}$$

The equation 5 proves that for any integer N belonging to the set of all positive integers, N^2 is equal to the sum of Nth odd integer starting from 1 and $(N - 1)^2$.
The equation 5 can be generalized as,

$$(N - k)^2 = (2 * (N - k) - 1) + (N - (k - 1))^2 \tag{6}$$

Where k belongs to the set of all whole numbers less than N.

From equation 6,the square of N can be written as,

$$(N - 0)^2 = 2 * N - 1 + (N - 1)^2 \tag{7}$$

$$(N - 0)^2 = 2 * N - 1 + 2 * (N - 1) - 1 + (N - 2)^2 \tag{8}$$

$$(N - 0)^2 = 2 * N - 1 + 2 * (N - 1) - 1 + 2 * (N - 2) - 1 + (N - 3)^2 \tag{9}$$

On recursive application of the formula from the equation 6 the equation 9 can be expanded as.

$$N^2 = 2 * N - 1 + 2 * (N - 1) - 1 + ... + 1 \tag{10}$$

From equation 10 the consistency of result 1 is verified and N^2 is expressed as the sum of first N consecutive odd integers starting from 1.

3.2 result 2

For a set of consecutive odd integers starting from 1 and the length of the set be of the form 3^k for k belonging to the set of all natural numbers, the sum of all the elements of the set is equal to 9^k. Formally, for the length of the set to be given by X and,

$$X = 3^k \tag{11}$$

Where k belongs to the set of all natural numbers then this specifies that the sum of all the elements of the set given by S is,

$$S = 9^k \tag{12}$$

3.2.1 proof

Let S be the sum of first 3^k consecutive odd integers starting from 1.

$$S = 1 + 3 + ...(1 + (3^k - 1) * 2) \tag{13}$$

$$S = sum_{n=1}^{3(k-1)}(6n - 1 + 6n - 3 + 6n - 5) \tag{14}$$

$$\tag{15}$$

where n is a natural number.

$$S = sum_{n=1}^{3(k-1)} 9(2n - 1) \tag{16}$$

$$S = sum_{n=1}^{3(k-2)} 9^2(2n - 1) \tag{17}$$

$$...S = sum_i n = 1^{3^0} 9^k(2n - 1) \tag{18}$$

$$\tag{19}$$

As in the equation 18 $3(k - k)$ or 1 is the maximum value of n and 1 is also the lowest value of n in the summation, the equation 18 reduces to,

$$S = 9^k(2 * 1 - 1) \tag{20}$$

$$S = 9^k \tag{21}$$

The equation 21 verifies the consistency of result 2.

4 The algorithm

4.1 step 1

Store the input integer in a variable N and create a flag variable $negative = false$. If $N < 0$, reassign $N = -1 * N$ and $negative = true$.

4.2 step 2

In a temporary variable $temp$ and initialize a variable $tripletSize$ valued 1.

4.3 step 3

Iteratively divide the value of $temp$ by 9 until it becomes 0 and for each iteration multiply the value of $tripletSize$ by 3.

4.4 step 4

Initialize the variables $unitSize$, $squareRoot$, $startTerm$ valued $tripletSize/3$, 0, 1 respectively. Reassign $temp$ equal to N.

4.5 step 5

Initialize variables $sum1$ and $sum2$. The variables $sum1$ stores the arithmetic sum of the first $unitSize$ terms starting from $startTerm$ with a common difference of 2, given by the formula

$$sum1 = (unitSize * (2 * startTerm + (unitSize - 1) * 2))/2 \tag{22}$$

The variable $sum2$ stores the arithmetic sum of the first $2 * unitsize$ terms starting from $startTerm$ with a common difference of 2, given by the formula

$$sum2 = ((unitSize * 2) * (2 * startTerm + ((unitsize * 2) - 1) * 2)/2 \tag{23}$$

4.6 step 6

Compare the value of $temp$ and $sum2$, the following cases arise:

4.6.1 $sum2 <= temp$

In this case, increment the value of $squareRoot$ by $2 * unitSize$ and decrement the value of $temp$ by $sum2$. Then proceed to step 7.

4.6.2 $sum2 > temp$

In this case proceed to step 6.

4.7 step 7

Compare the value of $temp$ and $sum1$,the following cases arise.

4.7.1 $sum1 <= temp$

In this case increment the value of $squareRoot$ by $unitSize$ and decrement the value of $temp$ by $sum1$.

4.7.2 $sum1 > temp$

In this case proceed to step 7.

4.8 step 8

Reassign the value of $unitSize$ to $unitSize/3$. Also change the value of $startTerm$ to $1 + squareRoot * 2$.

4.9 step 9

Iteratively repeat each step from step 4 to step 8 until the value of either $temp$ or $unitSize$ falls below 1.

4.10 step 10

The value of the *squareRoot* variable holds the magnitude of integer value of the square root of the input integer N. Check the value of *negative* if it is *true* then return $-1 * squareRoot$ else return $squareRoot$.

4.11 C++ Implementation

Repository for C++ implementation

5 Discussion

The proposed algorithm for finding integer value of square roots of all integers is based upon subtle mathematical insights.

6 Conclusion

This paper introduces a novel, mathematically grounded algorithm for square root computation. By focusing on theoretical efficiency and precision, the proposed method offers a significant advancement over traditional approaches for computation of integer value of square roots of integers of very large magnitude. Future work will aim to refine the algorithm further and explore its integration into broader mathematical frameworks.