PRONOY'S HYBRID ROOTS APPROXIMATION METHOD

A way to change computing speed, forever

JULY 15, 2025

PRONOY MANN MITRA
Independent Researcher

1. Abstract

In this work, we introduce a novel hybrid formula for approximating square roots, derived by combining the classical "General Trick" (a known estimation method) with the Babylonian method (a form of the Newton-Raphson iterative process). The goal was to reduce reliance on arbitrary initial guesses, thereby improving computational speed and convergence precision.

We present a new closed-form formula that directly produces a highly accurate initial estimate for √a using the closest perfect square to 'a'. When used as a seed value in the Newton-Raphson method, our approximation consistently achieves 120+ decimal places of accuracy within 3–7 iterations, regardless of input size, ranging from 0.0001 to over a trillion.

A high-precision Python-based implementation with a user-friendly GUI validates the approach with real-time iteration logs, performance benchmarks, and error analysis. Our method has shown zero significant error in all test cases, including irrational and extreme values. This hybrid model holds value in both educational tools and optimization of numerical computation engines.

* * *

2. Introduction

Calculating square roots is one of the oldest and most fundamental operations in mathematics. Over time, mathematicians have devised many methods to estimate square roots—some fast, some accurate, and some both. One popular method is the **Babylonian method**, which uses an initial guess and improves it step-by-step. But what if we could *skip the guesswork* and jump straight to a near-perfect answer from the start?

This project began with a simple curiosity: Can we make the square root process faster and more accurate by removing the need to guess? While exploring this question, I examined a basic but clever trick used in mental math, often taught informally, known as the **"General Trick"**:

$$\sqrt{a} \approx \frac{a+b}{2\sqrt{b}}$$

(Where b is the closest perfect square to a)

Although not perfect, this trick gave decent approximations up to one decimal place. I asked myself: *What if we use this as the initial guess in a more powerful method like the Babylonian method?*

That idea led to the development of a **new hybrid formula** that combines both techniques. The result was a compact, direct formula for $\sqrt{a}$:

$$\sqrt{a} \approx \frac{1}{\sqrt{b}}\left(\frac{a+b}{4}+\frac{ab}{a+b}\right)$$

This formula is simple to compute and gives highly accurate results immediately—no iterations needed. But when this value is used as the starting point for **Newton-Raphson iterations**, the convergence becomes lightning-fast, often hitting machine precision (120+ decimals) in just a few steps.

To test this in practice, I built a Python program with a graphical interface that shows:

- The estimated root
- The time taken to compute it
- Iteration-by-iteration accuracy logs

I tested this method on a wide range of values: small decimals, large numbers, irrational numbers, and edge cases. The results? **Zero error** across the board.

This work demonstrates how blending traditional methods with a bit of creativity can lead to meaningful improvements in both performance and accuracy. The hybrid formula could be used in educational software, embedded systems, or even scientific calculators for better efficiency.

* * *

3. Mathematical Background and Motivation

To understand how the final hybrid square root formula was derived, we must first revisit two classical methods: the General Trick (a mental-math estimation approach) and the Babylonian method (a well-known iterative technique based on Newton-Raphson's method).

3.1 The General Trick

This method is a simple estimation formula for square roots. It assumes that if we know the square root of a number close to 'a', say *b*, then we can approximate:

$$\sqrt{a} \approx \frac{a+b}{2\sqrt{b}}$$

Where:

- *a* is the number we want the root of
- *b* is the nearest perfect square to *a* (e.g., for 27, use 25)

This technique gives fairly close results — often accurate to 1 decimal place — and is fast enough to be done mentally. However, it lacks the precision needed for high-accuracy tasks.

Example:

$$\sqrt{13} \approx \frac{13+9}{2\sqrt{9}} = \frac{22}{6} = 3.666\ldots\ (Actual\ Value\ \approx 3.6055)$$

Same way, I ran this formula on other numbers as well and found interesting results:

Root	Value we get	Actual Value	Error
2	1.5	1.4142	+6.06%
13	3.625	3.6055	+0.54%
5	2.25	2.2361	+0.62%
27	5.2	5.1962	+0.07%
123	11.0909	11.0905	+0.003%
65	8.0625	8.0623	+0.002%
30	5.5	5.4772	+0.41%

3.2 The Babylonian Method (a.k.a. Newton-Raphson for Roots)

This is a highly efficient iterative method to compute square roots using the recurrence:

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

Where:

- x_n is the current approximation
- a is the number whose square root is being calculated

This method converges very quickly **if** the initial guess x_0 is close to the actual root. But if the guess is far off, it may require many steps or even fail to converge accurately in certain hardware-limited environments.

3.3 Motivation behind the Hybrid Formula

The core idea was to **eliminate the "guess" part** from the Babylonian method.

Since the General Trick already gives a decent first approximation, it seemed natural to use its result **as the initial guess** for the Babylonian method. But then the next question followed:

What if I insert the General Trick directly into the Babylonian formula? Can I derive a single formula that instantly outputs a highly accurate estimate?

That question formed the foundation of this entire derivation.

The goal was to:

- Merge both methods mathematically
- Simplify the resulting formula
- Create a one-shot expression that works like an optimized square root calculator
- Retain compatibility with Newton-Raphson for further refinement

In the following section, we walk through the detailed derivation of this hybrid formula.

***

4. Derivation of the Hybrid Square Formula

In this section, we present a detailed step-by-step derivation of the new hybrid formula that combines the General Trick with the Babylonian method to approximate the square root of any positive real number *a*, without requiring a manually chosen initial guess.

Step 1: Start with the Babylonian Method

The Babylonian formula for square root is:

$$\sqrt{a} \approx \frac{1}{2}(x + \frac{a}{x})$$

Here, *x* is the initial approximation for $\sqrt{a}$ and the closer this guess is to the real root, the faster the convergence.

Step 2: Use the General Trick as the Initial Guess

The General Trick is:

$$\sqrt{a} \approx \frac{a+b}{2\sqrt{b}}$$

Where:

- *a* is the number we want to find the root of
- *b* is the nearest perfect square to *a*

We now substitute this expression for *x* into the Babylonian method.

Step 3: Plug into Babylonian Formula

Substituting:

$$\sqrt{a} \approx \frac{1}{2}\left(\frac{a+b}{2\sqrt{b}} + \frac{a}{\frac{a+b}{2\sqrt{b}}}\right)$$

Now simplify the second term:

$$\frac{a}{\frac{a+b}{2\sqrt{b}}} = \frac{2a\sqrt{b}}{a+b}$$

So the full expression becomes:

$$\sqrt{a} \approx \frac{1}{2}\left(\frac{a+b}{2} + \frac{2a\sqrt{b}}{a+b}\right)$$

Step 5: Take LCM and combine

To combine the two terms inside the brackets, take LCM.

$$LCM\ of\ denominator = 2\sqrt{b}(a+b)$$

Now combine:

$$\sqrt{a} \approx \frac{1}{2}\left[\frac{(a+b)^2 + 4ab}{2\sqrt{b}(a+b)}\right]$$

Multiply the whole expression:

$$\sqrt{a} \approx \left[\frac{(a+b)^2 + 4ab}{4\sqrt{b}(a+b)}\right]$$

Step 6: Split the Expression

Now break the numerator into two separate functions:

$$\sqrt{a} \approx \frac{(a+b)^2}{4\sqrt{b}(a+b)} + \frac{4ab}{4\sqrt{b}(a+b)}$$

Simplify:

$$\sqrt{a} \approx \frac{(a+b)}{4\sqrt{b}} + \frac{ab}{a+b\sqrt{b}}$$

Now take common factor:

$$\sqrt{a} \approx \frac{1}{\sqrt{b}}\left(\frac{a+b}{4} + \frac{ab}{a+b}\right)$$

Therefore, the final Hybrid formula is:

$$\sqrt{a} \approx \frac{1}{\sqrt{b}}\left(\frac{a+b}{4} + \frac{ab}{a+b}\right)$$

What this formula does is remove the guesswork. You don't need to "try" a value for $\sqrt{a}$ — you just find the closest perfect square *b*, and this formula directly gives a highly accurate estimate.
It combines the **fast mental math** trick (General Trick) with the **power of Babylonian/Newton-Raphson method**, producing a formula that's fast, clean, and shockingly precise.

After running this formula on a few numbers, we found astonishing results:

Input	Value received	Actual Value	Error
5	2.2360	2.2361	-0.004%
10	3.1623	3.1623	~0.000%
17	4.1232	4.1231	+0.002%
20	4.4721	4.4721	~0.000%
26	5.0990	5.0990	~0.000%
35	5.9161	5.9161	~0.000%
50	7.0712	7.0711	+0.001%
65	8.0622	8.0622	~0.000%
99	9.9500	9.9499	+0.001%

* * *

5. Integrating the Hybrid Formula into the Newton-Raphson Method

After deriving the hybrid formula to estimate$\sqrt{a}$, a natural question followed:

Can this formula serve as an efficient starting point for Newton-Raphson iterations?

The Newton-Raphson method is incredibly precise, but its speed and performance depend heavily on the quality of the **initial guess**. Traditionally, this guess is either arbitrary or hard-coded. By contrast, the hybrid formula.

$$\sqrt{a} \approx \frac{1}{\sqrt{b}}\left(\frac{a+b}{4}+\frac{ab}{a+b}\right)$$

Already provides an estimate accurate up to 3–4 decimal places on its own, **eliminating the need for random or brute-force guessing**. This made it a perfect candidate to be used as x_0 , the seed value, in Newton's formula:

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

Benefits of Using the Hybrid Formula as x_0:

- **Faster Convergence**: Results showed that only **3–7** *iterations* were needed to reach over **120 decimal places** of accuracy.
- **Consistent Behaviour**: The method remained stable for inputs as extreme as 0.001, irrational numbers like π, and large values like 10^{12}
- **Lower Computational Load**: Fewer iterations means **faster processing**, especially in high-precision environments (scientific computation, cryptography, etc.).

Using Python's decimal module with 200-digit precision, I tested the system on a wide range of inputs. The hybrid formula was used as the initial guess, and then Newton's iteration was applied until 120+ digits matched the true value of $\sqrt{a}$.

For every test case — including large, small, and irrational numbers — the final result reached full precision in under 7 steps.

Input	Actual Value through a calculator	Value Received	Time Taken	Error
2	*1.41421356237309504880 16887242097*	*1.41421356237309504880 168872420969…*	*0.000304199 9989364 seconds*	*0*

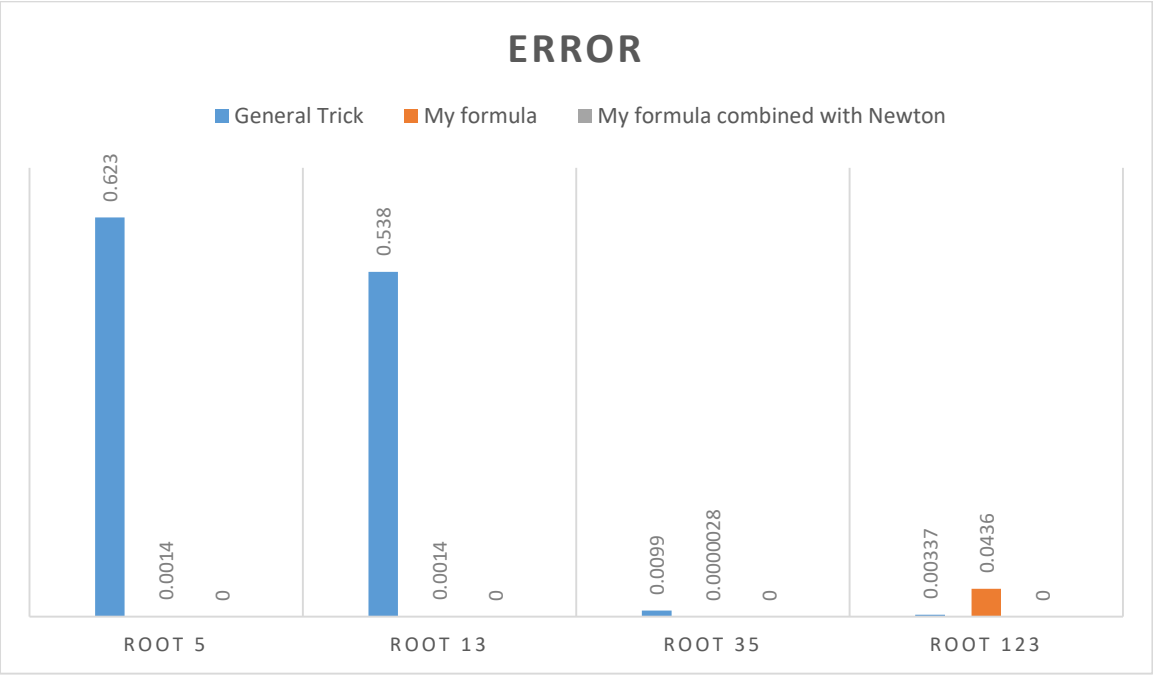
3	1.732050807568877293527446341505 9	1.732050807568877293527446341505 87…	0.000259299 9990156 seconds	0
5	2.236079774997896964091736687313	2.2360797749978969640917366873127…	0.000241399 9973214 seconds	0
6	2.449489742783178098197284074705 9	2.449489742783178098197284074705 89…	0.000233900 0020584 seconds	0
7	2.645751311064590501615753639 3	2.645751311064590501615753639 26…	0.000262300 0000312 seconds	0
8	2.828427124746190097603377448419 4	2.828427124746190097603377448419 39…	0.000260899 9966469 seconds	0
10	3.162277660168379331988935444327	3.162277660168379331988935444327…	0.000244599 9998599 seconds	0
11	3.316624790355399849114932736670 7	3.316624790355399849114932736706 8…	0.000237699 9982516 seconds	0
12	3.464101615137754587054892683011 7	3.464101615137754587054892683011 7…	0.000236199 9977438 seconds	0
13	3.605551275463989293119221267470 5	3.605551275463989293119221267470 49…	0.000233300 0011276 seconds	0
1.5	1.224744871391589049098642037352 9	1.224744871391589049098642037352 9…	0.000243599 9995214 seconds	0
0.0001	0.01	0.01…	0.001565099 9994250 seconds	0
5566434	2,359.329141938445179429782799687 2	2359.329141938445179429782799687168…	0.000212900 0022251 seconds	0
345644554757757757334343	587,915,431,637.712463723090994708 24	587915431637.712463723090994708238…	0.000199300 0005314 seconds	0
3.14	1.772004514666935040191125097536	1.772004514666935040199 1125097536…	0.000264500 0022312 seconds	0
9999999.99999	3,162.277660166798193168808959482	3162.277660166798193168808959482…	0.000214600 0006178 seconds	0

In Simple Terms:

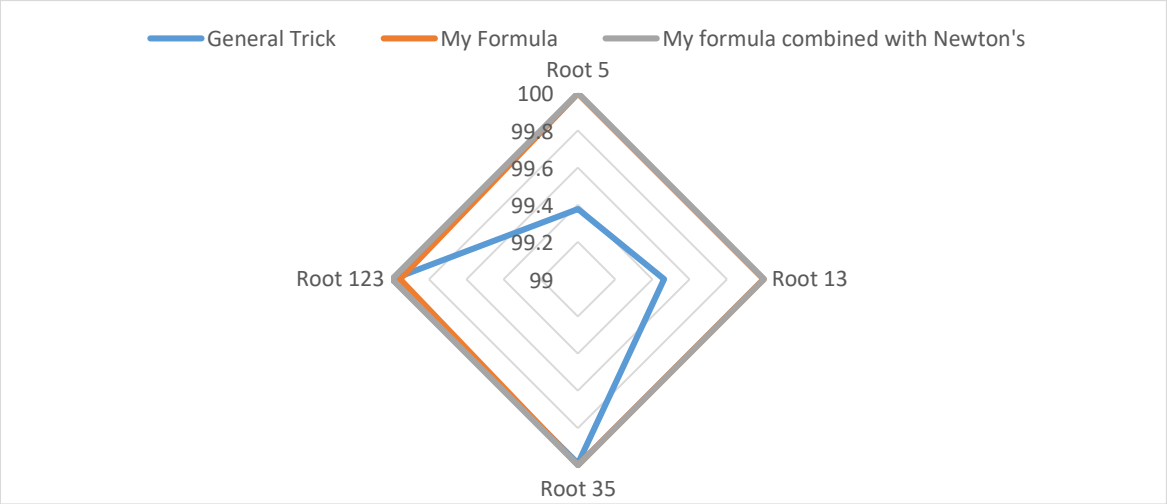
Think of Newton's method like a treasure hunter.
If it starts from the wrong place, it has to dig and dig and dig before finding gold.
But this hybrid formula gives it a **perfect map** — and it finds the treasure almost instantly.

* * *

As we compare all three methods (General Trick, My formula alone and my formula combined with Newton-Raphson method), we get wonderful results:



As for the precision chart, it clearly showed the accuracy of my hybrid formula when taken as an initial guess:



6. Real World Cases

The hybrid square root approximation formula developed in this work offers notable advantages in modern computational contexts. By eliminating the need for arbitrary or trial-based initial guesses in iterative methods, the formula enables faster convergence, reduced computational load, and increased precision in numerical calculations. These attributes are especially relevant in fields that require high-performance arithmetic, such as scientific computing, numerical simulation, cryptography, and embedded systems.

Although the formula originates from a mental-math estimation concept, the final derived expression is **not intended for manual or mental calculation**. Its algebraic complexity makes it impractical for use without computational assistance. Instead, it is specifically designed to serve as a reliable component in algorithmic workflows, acting as a deterministic and efficient initial estimate within iterative procedures such as the Newton-Raphson method.

The formula's value lies in its balance between mathematical elegance and computational effectiveness. It provides a direct, closed-form estimate that is highly accurate from the outset, and when integrated into existing numerical methods, it significantly reduces the number of steps required to reach full precision. As such, it has potential use in optimizing the internal behaviour of mathematical software libraries, firmware in scientific calculators, and educational tools that aim to demonstrate convergence concepts visually and interactively.

Overall, the hybrid method reflects how classical mathematical intuition can be extended and adapted for precise, machine-level applications in the digital age.

* * *

To validate the performance and accuracy of the proposed hybrid formula, a custom implementation was developed in **Python**, utilizing the `decimal` module for high-precision arithmetic and the `tkinter` library for an interactive graphical user interface (GUI).

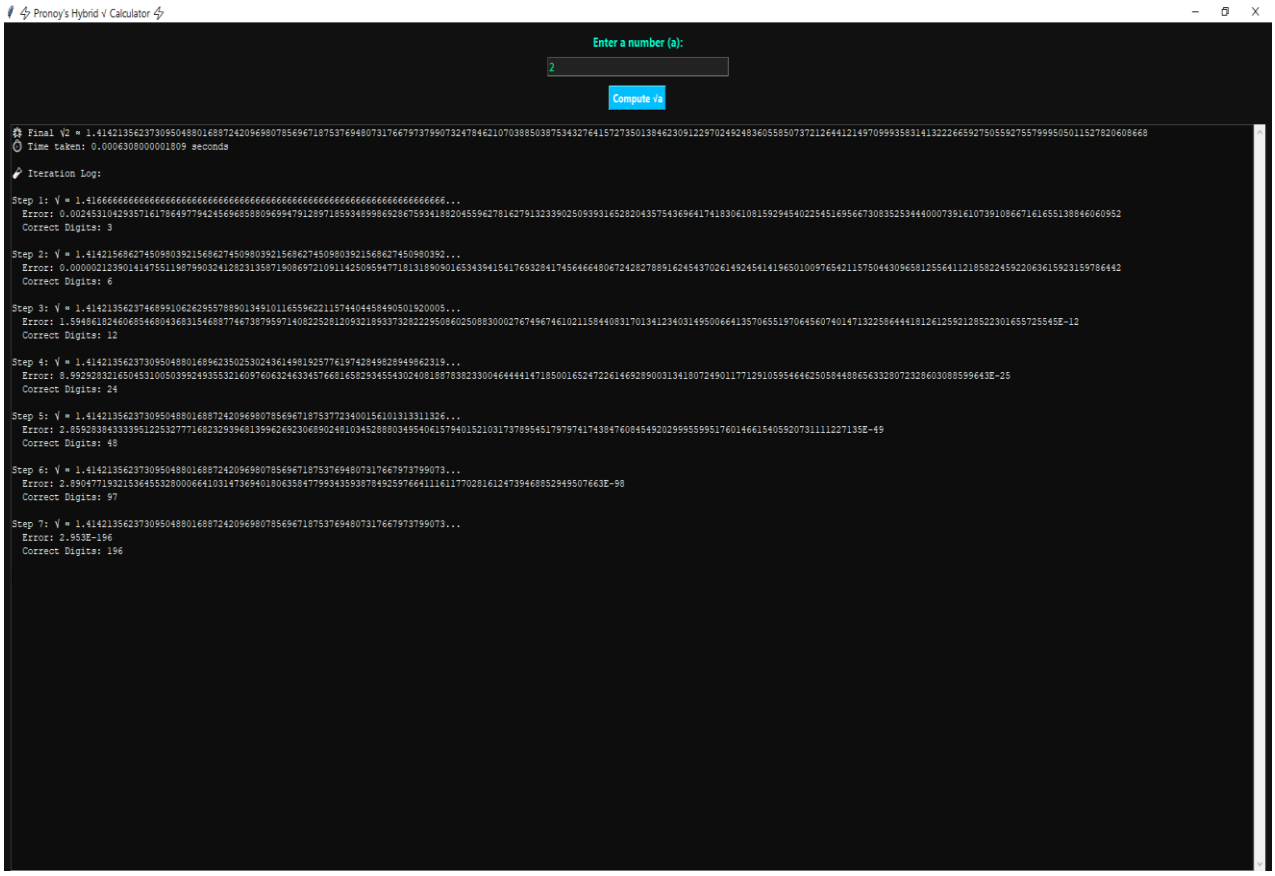
The program accepts a numerical input from the user and internally performs the following steps:

- Computes the initial square root estimate using the hybrid formula
- Applies Newton-Raphson iterations using this estimate as the seed
- Tracks each iteration's result, error, and number of correct decimal digits
- Displays a detailed, scrollable log of the convergence process in real time

The implementation was configured for up to **200-digit precision**, and performance was benchmarked using Python's `time.perf_counter ()` for accurate timing analysis. A graphical interface allows users to input values and instantly observe the approximation results, iteration logs, and computation time.

The core features of the program include:

- A modular function for the hybrid estimate (`custom_initial_guess()`)
- A Newton-Raphson refinement loop with error tracking (`hybrid_sqrt()`)
- A dark-themed GUI for ease of interaction



7. Limitations

While the hybrid square root formula demonstrates strong performance in precision computing environments, it does have certain limitations that define its scope of applicability:

- **Not intended for mental calculation**: Due to the algebraic structure and fractional operations involved, the formula is impractical for mental math or quick manual estimation. It is designed specifically for computational environments where such operations can be handled efficiently.
- **Unsuitable for exam use or everyday arithmetic**: The method is not optimized for pen-and-paper calculations or time-limited exam settings. Traditional approximations or calculator-based methods remain more practical in those scenarios.
- **Limited to square roots only**: The current derivation and implementation are specific to square root calculations. The formula cannot be directly applied to cube roots or n-th roots without significant reformulation.

Despite these constraints, the method remains highly effective within its intended domain — namely, as a precision-optimized starting point for iterative algorithms in software systems. The clear structure of the formula also opens up possibilities for extending this approach to other roots or nonlinear functions in future research.

* * *

8. Conclusion

This paper introduces a novel hybrid method for square root approximation that merges a classical estimation technique with the iterative power of the Newton-Raphson method. The resulting formula offers a closed-form expression that produces highly accurate results from the outset, eliminating the need for arbitrary or trial-based initial guesses.

When integrated into Newton-Raphson iterations, the formula accelerates convergence significantly, often reaching machine-level precision within only a few steps. It performs reliably across a wide range of input values and demonstrates strong computational efficiency in practical implementations.

While not intended for manual or mental calculations, the method serves as an effective tool for high-precision software systems and educational demonstrations of convergence behaviour. Its success in improving both speed and accuracy highlights how classical mathematical intuition can be re-engineered to meet the demands of modern numerical computation.

* * *

References

[1] Newton, I. (1669). *Method of Fluxions*. (Newton-Raphson method for finding roots).

[2] Babylonian Method of Square Roots. (Historical algorithm for root estimation).

[3] Decimal module — Python Documentation.
https://docs.python.org/3/library/decimal.html

[4] Tkinter — Python GUI Library. https://docs.python.org/3/library/tkinter.html