COMPUTARIUM
Lycée classique de Diekirch
32 av. de la gare L-9233 Diekirch

Playing with the Square Root

A Practical Study

Claude BAUMANN

Version 1.2
Last edited: July 2, 2024

Abstract

The computation of square roots has a fascinating history, dating back to ancient Babylonian mathematicians. Over time, various algorithms have been developed to calculate square roots, both for integer and floating-point numbers. In this paper, we explore several remarkable methods, uncovering hidden insights, examining their potential for computer implementation, and evaluating their computational efficiency. Specifically, we introduce a novel variant of the continued fraction approach, which can be implemented with minimal code length. From this method, we derive an unexpected algorithm that utilizes multiplication instead of division. Additionally, we present Toepler's shift-and-add algorithm, along with a binary adaptation optimized for fast and precise floating-point square roots.

Document History

- Start of the document: June 1, 2022 Version: 1.1
- First revision: July 2, 2024 V. 1.2: Minor corrections; added references to FRIDEN calculator patents; added link to Arithmeum (Daniel Meyer video); added mantissa restriction method to interval (1;1.5)

Contents

Introduction

Square roots in general and more specifically $\sqrt{2}$ have captivated the minds of brilliant mathematicians throughout history. Their contemplations invariably revolved around the irrational nature of these enigmatic numbers. On a practical level, they sought efficient methods for approximating their values. While Babylonian mathematicians laid the groundwork for square root approximation, it was Heron of Alexandria in first-century Egypt who devised the earliest algorithm for computing square roots (cf. Knuth (1972) and Flores (2014)).

Modern analytic methods emerged after the introduction of the Arabic numeral system to Western Europe during the early Renaissance preparing what is known today as Newton's square root method, a special case of the more general Newton-Rapson root-finding method (cf. Ypma (1995)) and -most strikingly- identical to Heron's algorithm.

What makes this historical approach extraordinary is the fact that there is hardly another concise method for calculating the square root so quickly. It surely must be regarded as the benchmark for all approximation methods. However, beyond Heron and the school long division method, which can be applied using pen and paper, a long list of algorithms has been developed for easier and more specific implementation into mechanical and digital calculators. Today, the choice of method depends on more parameters than just computation speed. Factors such as desired accuracy, number format, available computational resources, and programming ease also play a crucial role. Comparison of various methods can be found in Montuschi and Mezzalama (1990) and Dianov and Anuchin (2020b), for instance.

Relevant Algorithms:

- Heron's Method (Iterative): A special case of Newton's root-finding method, Heron's algorithm refines an initial estimate iteratively until a termination criterion is met.
- Digit-by-Digit Calculation: Some methods compute square roots digit by digit, allowing for mental calculation, paper-and-pencil and elementary computing machine work.
- Taylor Series Expansion: Taylor series can approximate square roots, providing accurate results based on successive terms.
- Continued Fraction Expansions: Rational approximations of square roots can be obtained using continued fractions.
- Convex Approximation: Hyperbolic approximation of square root for fixed-point arithmetic (cf. Dianov and Anuchin (2020a)).
- Goldschmidt Convergence: This very special method extends the Goldschmidt division using addition and multiplication (Goldschmidt (1964)) A computer friendly version can be in Dianov and Anuchin (2020b).

Efficiency Evaluation:

- Algorithms vary in their convergence rates, computational complexity, and error propagation.
- The choice of method depends on the desired accuracy and available computational resources.
- Some algorithms, like paper-and-pencil synthetic division and series expansion, do not require an initial value.
- Integer square roots, rounded or truncated to the nearest integer, also find applications in specific scenarios.

Understanding square root computation methods and evaluating the efficiency of algorithms are crucial for optimizing performance in various computational tasks and environments. This study holds significant value for designing compiler programs that adapt to specific CPUs or for developing applications where square roots are needed. Additionally, exploring square root algorithms provides insight into complex topics at the intersection of number theory and computer science, including convergence rates and number formats.

In the subsequent sections, we will begin by introducing the well-established Heron method. This iterative algorithm strikes a balance between quadratic convergence and a minimal number of calculations per iteration, making it one of the fastest known methods. Next, we'll explore an unexpectedly straightforward and effective algorithm based on specialized continued fractions. We'll analyze its efficiency and derive a remarkably simple method that uses multiplication instead of division. Finally, we'll discuss the Toepler algorithm—a paper-and-pencil method successfully implemented in both mechanical and digital computing machines for integer numbers. As a novel addition, we'll present an efficient adaptation of the Toepler algorithm for floating-point numbers.

Preliminary Note This study warrants consideration from practical and didactic perspectives. The paper emerged from the intention to shed light on lesser-known approaches to the square root, recognizing that much has likely already been said on this topic. Number theorists may encounter inconsistencies, incompleteness, or even stumble upon errors. Numerical mathematicians might yearn for greater rigor and could express reservations about not mentioning algorithms like the quartically convergent Bakhshali method. Programmers, too, might miss fail-proof implementations and detailed listings. We encourage our readers to share their questions and critiques with claude.baumann@education.lu.

Part I

Heron's Method

Let's start deriving Heron's famous approximation method using simple algebra:

$$\begin{aligned} \forall N \in \mathbb{R}_+^*, \quad & N = x^2 \\ \iff & \frac{N}{x} = x \\ \iff & \frac{N}{x} + x = 2x \\ \iff & x = \frac{\frac{N}{x} + x}{2} = \frac{1}{2}\left(\frac{N}{x} + x\right) \end{aligned} \tag{1}$$

At first glance, this equation doesn't make much sense, as it tautologically defines x with itself. In fact, the process can be repeated forever without any visible gain:

$$\begin{aligned} x &= \frac{1}{2}\left(\frac{N}{\frac{1}{2}\left(\frac{N}{x}+x\right)} + \frac{1}{2}\left(\frac{N}{x}+x\right)\right) \\ &= \frac{1}{2}\left(\frac{N}{\frac{1}{2}\left(\frac{N}{\frac{1}{2}\left(\frac{N}{x}+x\right)} + \frac{1}{2}\left(\frac{N}{x}+x\right)\right)} + \frac{1}{2}\left(\frac{N}{\frac{1}{2}\left(\frac{N}{x}+x\right)} + \frac{1}{2}\left(\frac{N}{x}+x\right)\right)\right) \\ &\ldots \end{aligned} \tag{2}$$

However, we may consider this Eq. 1 as a non-terminated recursive definition of x that may be turned into a recursive sequence:

$$x_{i+1} = \frac{\frac{N}{x_i} + x_i}{2} = \frac{N + x_i^2}{2x_i} \tag{3}$$

with initial value x_0 chosen in the vicinity of $\sqrt{N}$.

1 Convergence of Heron's algorithm

The sequence defined in Eq. 3 converges to a limit, $\sqrt{N}$ for instance, as illustrated in Table 1.

i	x_i	$x_{i+1} - x_i$
0	2	-0.5
1	1.5	-0.0833333
2	1.41667	-0.00245098
3	1.414216	-2.1239E-6
4	1.414214	-1.59495E-12
5	1.414214	

Table 1: Sequence values for $\sqrt{2}$ with starting value $x_0 = 2$

Let's consider:

$$f(x) = \frac{\frac{N}{x} + x}{2} \tag{4}$$

This function has a single fix point in the domain $\mathbb{R}_+^*$, which is $x_* = \sqrt{N}$.

The function Eq. 4 is continuous and differentiable.

A fixed point x_* is attracting, if it is located in the neighborhood interval I, where:

$$\forall x \in I, \ \lim_{i \to \infty} f_i(x) = x_*, \text{ with } f_i(x) = f \circ f \circ f \circ \ldots f(x) \tag{5}$$

The Attracting Fixed Point Theorem (cf. Ford (2005)) states that the condition:

$$| f'(x_*) | < 1 \tag{6}$$

is sufficient to prove the attracting property.

$$| f'(x_*) | = \frac{N}{2x_*^2} < 1 \text{ (!)} \tag{7}$$

This verification proves that the discussed sequence is convergent.

1.1 Rate of convergence

Because the sequence has a single fix point only in the domain $\mathbb{R}_+^*$, it may start with a value $x_0^2 > N$ or $x_0^2 < N$, if we ignore the trivial case, where $x_0^2 = N$. Let's consider:

$$\begin{aligned}\Delta &= x_1^2 - N \\ &= \frac{1}{4}\left(x_0 + \frac{N}{x_0}\right)^2 - N \\ &= \frac{1}{4}\left(x_0 - \frac{N}{x_0}\right)^2 > 0 \text{ !!! } \Longrightarrow x_1^2 > N \iff x_1 > \sqrt{N}\end{aligned} \tag{8}$$

This means that the sequence is bounded below by $\sqrt{N}$ beyond its second member –and consequently any subsequent member– regardless of its initial conditions.

$$x_{i+1} - x_i = \frac{\frac{N}{x_i} + x_i}{2} - x_i = \frac{N - x_i^2}{2x_i} \tag{9}$$

$$\begin{aligned}x_{i+2} - x_{i+1} = \frac{N - x_{i+1}^2}{2x_{i+1}} &= \frac{N - \left(\frac{N+x_i^2}{2x_i}\right)^2}{2\frac{N+x_i^2}{2x_i}} \\ &= \frac{N - \frac{N^2+2Nx_i^2+x_i^4}{4x_i^2}}{\frac{N+x_i^2}{x_i}} = \frac{\frac{4Nx_i^2-N^2-2Nx_i^2-x_i^4}{4x_i^2}}{\frac{N+x_i^2}{x_i}} \\ &= -\frac{\left(N - x_i^2\right)^2}{4x_i(N+x_i^2)} \\ &= -\frac{x_i}{(N+x_i^2)}(x_{i+1}-x_i)^2 \\ &= \mu_i(x_{i+1}-x_i)^2\end{aligned} \tag{10}$$

Because $\mu_i < 0, \ \forall i > 0$, Eq. 10 describes a *quadratic* monotone decreasing convergence.

For sufficiently large i, or for x_i in close vicinity of $\sqrt{N}$, we may estimate:

$$\hat{\mu} = \lim_{i \to \infty} \frac{-x_i}{N + x_i^2} = -\frac{\sqrt{N}}{2N} \tag{11}$$

Let

$$\delta_i = \mid x_{i+1} - x_i \mid, \ \forall i \geq 0 \tag{12}$$

Although δ_i is not exactly identical with the absolute approximation error $\psi_i = \mid x_i - \sqrt{N} \mid$, it can be regarded as a valuable estimation of that error.

We have $\forall i \geq 0$:

$$\delta_{i+1} = \mid \mu_i \mid \delta_i^2 \tag{13}$$

For sufficiently large i, or for x_i in close vicinity of $\sqrt{N}$, we may estimate:

$$\begin{aligned}
\hat{\delta}_{i+1} &= \mid \hat{\mu} \mid \hat{\delta}_i^2, \ \text{with } \hat{\delta}_0 = \mid x_1 - x_0 \mid \ \text{sufficiently small} \\
\hat{\delta}_1 &= \mid \hat{\mu} \mid \hat{\delta}_0^2 \\
\hat{\delta}_2 &= \mid \hat{\mu} \mid \hat{\delta}_1^2 = \mid \hat{\mu} \mid \left\{ \mid \hat{\mu} \mid \hat{\delta}_0^2 \right\}^2 = \mid \hat{\mu} \mid^3 \hat{\delta}_0^4 \\
\hat{\delta}_3 &= \mid \hat{\mu} \mid \hat{\delta}_2^2 = \mid \hat{\mu} \mid \left\{ \mid \hat{\mu}^3 \mid \hat{\delta}_0^4 \right\}^2 = \mid \hat{\mu} \mid^7 \hat{\delta}_0^8 \\
&\dots \\
\hat{\delta}_i &= \mid \hat{\mu} \mid^{2^i - 1} \hat{\delta}_0^{2^i}
\end{aligned} \tag{14}$$

Due to the large positive or negative exponents associated with the values in this estimation, it is advisable to switch to a logarithmic scale. This will provide an estimate of the magnitude of these residues:

$$\log(\hat{\delta}_i) = (2^i - 1) \log \mid \hat{\mu} \mid + 2^i \log \hat{\delta}_0 \tag{15}$$

Table 2 illustrates an example of convergence along with the corresponding values for the described parameters. Notably, there are significant discrepancies between the true δ_i and the estimated $\hat{\delta}_i$, which highlight the sensitivity of the method to initial conditions. If the initial value x_0 is sufficiently close to $\sqrt{N}$, these differences actually correspond to variations of a few units in the number of iterations needed to achieve a certain precision. Attention must be put here! If x_0 is distant from $\sqrt{N}$, the estimation $\hat{\delta}_i$ may be completely inaccurate.

i	x_i	δ_i	δ_i^2	$\mid \mu_i \mid$	$\hat{\delta}_i$
0	10	4.5	20.25	0.0909091	4.5
1	5.5	1.84091	3.38895	0.136646	3.20181
2	3.659091	0.463086	0.214448	0.156445	1.62091
3	3.196005	0.0335495	0.00112557	0.158105	0.415423
4	3.162456	1.77958E-4	3.16689E-8	0.158114	0.0272867
5	3.162278	5.0073E-9	2.5073E-17	0.158114	1.17726E-4
6	3.162278	-	-	0.158114	2.19135E-9

Table 2: Example: Approximate $\sqrt{10}$ starting with $x_0 = 10$.

If p is the given precision that the algorithm should reach, the number of iterations n required can be estimated as follows:

$$\begin{aligned} p &= \hat{\delta}_i \\ \Longrightarrow \log p &= (2^i - 1)\log|\hat{\mu}| + 2^i \log \hat{\delta}_0 \\ &= 2^i(\log|\hat{\mu}| + \log \hat{\delta}_0) - \log|\hat{\mu}| \\ \Longleftrightarrow 2^i &= \frac{\log p + \log|\hat{\mu}|}{\log|\hat{\mu}| + \log \hat{\delta}_0} = \eta \\ \Longrightarrow i &= \frac{\log \eta}{\log 2} \end{aligned} \tag{16}$$

We define the rounded number of iterations:

$$n = \lceil i \rceil + 1 \tag{17}$$

Example: Estimate the number of iterations needed to reach $p = 1E-7$ for $\sqrt{10}$ with $x_0 = 10$. (We will ignore the first very bad δ_0 and take instead $\delta_1 \approx 1.84091$, knowing that we must increment our result n because of this choice, see Table 2).

$$\begin{aligned} \eta &= \frac{-7 + \log 0.158114}{\log 0.158114 + \log 1.84091} \approx 14.55424 \\ n_1 &= \left\lceil \frac{\log \eta}{\log 2} \right\rceil + 2 = 5 \end{aligned} \tag{18}$$

Remark: The number of iterations n required for a given precision p depends essentially on the product:

$$\lambda = |\hat{\mu}|\,\delta_0 \tag{19}$$

2 Better choice of the starting value x_0

Clearly, the number of iterations n required for a desired precision depends on the initial guess x_0. Human intuition leads us to make an initial guess based on the magnitude of the problem, avoiding extreme values. However, instructing a machine to do this automatically and with minimal effort presents an interesting challenge.

2.1 Using the $\log_2$ function

We can exploit the following identity, where the base is chosen conforming to the binary architecture of digital computers:

$$\log_2 \sqrt{N} = \frac{1}{2}\log_2 N \tag{20}$$

In fact, because Heron's algorithm converges so rapidly, an initial value x_0 displaying the same magnitude as $\sqrt{N}$ will do an excellent job here.

For computational speed reasons, we as propose as a rounded candidate:

$$\begin{aligned} \kappa &= \left\lfloor \frac{1}{2}\lfloor \log_2 N \rfloor + 0.5 \right\rfloor \\ x_0 &= 2^{\kappa} \end{aligned} \tag{21}$$

We choose the function $L(x)=\lfloor\log_2 x\rfloor$, because this particular function can be calculated at bit level using elementary logic with no division or multiplication being implied. More specifically, the integer $\log_2$ is yielded using the NUMBER OF LEADING ZEROS **nlz** bit-function (here with 32-bit unsigned integers):

$$L(x)=\lfloor\log_2 x\rfloor=31-\mathbf{nlz}(x) \tag{22}$$

Please refer to Warren (2012), [pp.99-107] for further details on how to implement this elementary function.

Example: $x_0(N=10)=2^{\lfloor\frac{1}{2}\cdot 3+0.5\rfloor}=4$
This method might be indicated in the case of an implementation using integer numbers.

i	x_i	δ_i	δ_i^2	$\lvert\mu_i\rvert$	$\hat{\delta}_i$
0	4	0.75	0.5625	0.153846	0.75
1	3.25	0.086539	0.00748891	0.158055	0.0889391
2	3.163462	1.18366E-3	1.40104E-6	0.158114	0.00125071
3	3.162278	2.21524E-7	4.90731E-14	0.158114	2.47332E-7
4	3.162278	7.7592E-15	6.02052E-29	0.158114	9.67231E-15

Table 3: Example: Approximate $\sqrt{10}$ starting with $x_0=4$.

Table 3 displays a faster convergence due to a better initial value x_0. Now Eq. 16 and 17 give the number of iterations:

$$\begin{aligned}\eta &= \frac{-7+\log 0.158114}{\log 0.158114+\log 0.75}\approx 7.2235\\ n &= \left\lceil\frac{\log\eta}{\log 2}\right\rceil+1=4\end{aligned} \tag{23}$$

2.2 Intermezzo: Hacking the IEEE Standard Single Precision Floating Point Representation

It must be acknowledged that floating-point numbers used in computers are actually rounded rational numbers, despite their association with real numbers (cf.Boldo et al. (2023)). This specific representation, however, is most useful in mathematical applications, although not necessarily innocuous, as we can learn from Goldberg (1991).

The IEEE754 standard for floating-point representation of a number in base 2 combines three attributes of the number x (cf.Arnold, Jeff (2017)):

1. Sign s ($s=0\implies x\geq 0$, $s=1\implies x\leq 0$)
2. Biased exponent p
3. Mantissa $m=m_1m_2m_3m_4...m_{23}$ with implicit (hidden) leading bit (=normalized).

Practical implementation of floating-point arithmetic and functions requires non-negligible overhead operations, because special bit-values must be considered particularly. For instance, because the number 0 has no unequivocal representation, pre-function routines must offer solutions for both valid instances of that number.

Example: Single precision representation (32-bit)

$$x=(-1)^s\cdot 2^{(p-127)}\cdot 1.m \tag{24}$$

bit

31	30	29	28	27	26	25	24	23	22	21	20	19	18	...	4	3	2	1	0
s	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	m_1	m_2	m_3	m_4	m_5	...	m_{19}	m_{20}	m_{21}	m_{22}	m_{23}

Properties:

1. $p = 255 \wedge m \neq 0 \implies x = \text{NaN}$ ("Not a number")
2. $p = 255 \wedge m = 0 \implies x = (-1)^s \cdot \infty$
3. $0 < p < 255 \implies x = (-1)^s \cdot 2^{(p-127)} \cdot (1.m)$, with $(1.m)$ representing the binary fractional part of the number created by preceding m with a leading 1 and the binary point.
4. Normalized mantissas always describe numbers belonging to the interval $[1,2)$.
5. $p = 0 \wedge m \neq 0 \implies x = (-1)^s \cdot 2^{-126} \cdot (0.m)$, representing denormalized values
6. $p = 0 \wedge m = 0 \wedge s = 1 \implies x = -0$
7. $p = 0 \wedge m = 0 \wedge s = 0 \implies x = +0$

The particular single precision representation of the number 2.25 is:

- s = 0
- p = 1 + 127 = 128 = b'10000000'
- m = 0.125 = b'0010000 00000000 00000000'

2.3 Calculating $\sqrt{N}$ with separated exponent and mantissa

Typically imperceptible to the average user, floating-point operations necessitate the separation of the exponent and the mantissa at low execution level. This process is called **expansion**, whereas the inverse procedure is denoted **normalization** or sometimes re-normalization.

We illustrate this with the LabVIEW diagram (Fig. 1) using the subroutine shown in Fig.3 in order to compute $\sqrt{N}$. Because the exponent is considered positive only in this implementation, the program treats the case $0 < N < 1$ by simply inverting the argument, and re-inverting the result after operation. The trivial cases, where $N <= 0$ or $N = 1$ are not displayed. LabVIEW has an integrated function that expands floating point numbers into unbiased exponent and mantissa parts.

Let $N > 0$

$$N = 2^e \cdot M, \text{ where } M \in [1,2) \tag{25}$$

We can conclude that $\sqrt{M} \in [1,2)$ too.

$$\sqrt{N} = \sqrt{2^e \cdot M} = \sqrt{2^e}\sqrt{M} \tag{26}$$

where $e \in \mathbb{N}$

There are two cases to observe:

1. e is even, $e = 2s,\ s \in \mathbb{N} \implies s = e/2$:

$$\sqrt{N} = 2^s\sqrt{M} \tag{27}$$

2. e is odd, $e = 2s - 1,\ s \in \mathbb{N} \implies s = (e+1)/2$:

$$\begin{aligned} \sqrt{N} &= 2^{\frac{2s-1}{2}}\sqrt{M} \\ &= 2^s \frac{1}{\sqrt{2}}\sqrt{M} \\ &= 2^s \frac{\sqrt{2}}{2}\sqrt{M} \end{aligned} \tag{28}$$

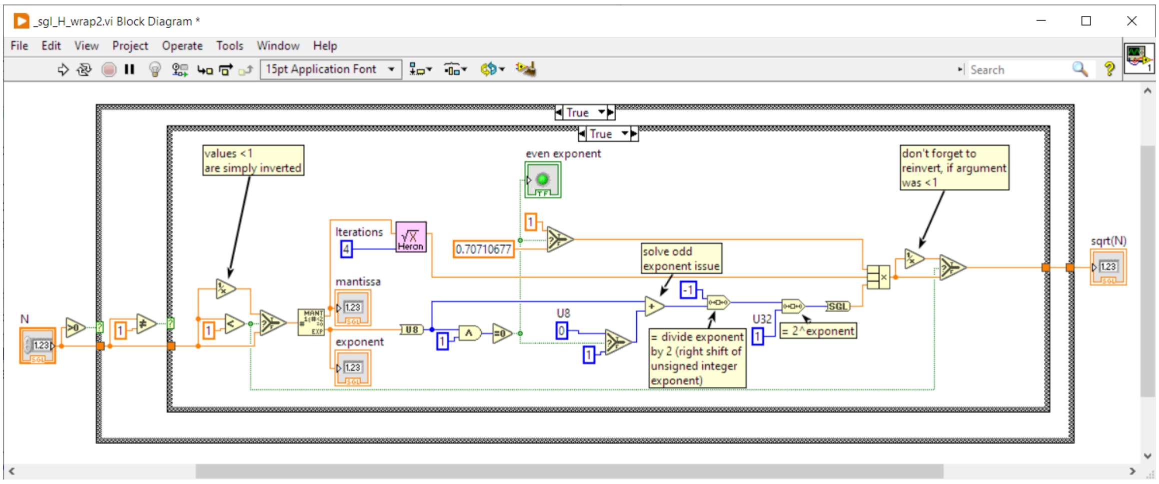

_sgl_H_wrap2.vi Block Diagram *
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
True
True
values <1 are simply inverted
even exponent
don't forget to reinvert, if argument was <1
Iterations
4
Heron
0.70710677
solve odd exponent issue
mantissa
exponent
N
sqrt(N)
1
U8
U32
0
-1
SGL
= divide exponent by 2 (right shift of unsigned integer exponent)
= 2^exponent

Figure 1: This LABVIEW program computes the square root on the exponent and the mantissa separately.

Eq. 28 can be interpreted as follows: $\forall N > 0$, its square root $\sqrt{N}$ can be calculated on the exponent and the mantissa separately, where the mantissa M is located in the interval $[1,2)$. In the case of an even exponent, the result has half the exponent of the radicand. In the other case, the exponent must incremented by 1 and then halved; **and** the square root of the mantissa must be scaled by constant $\frac{\sqrt{2}}{2}$.

Because the mantissa is bounded in the interval $[1,2)$, we can estimate the number of iterations required for reaching $p = 1E-7$ using Eq. 16, 17 and 19.

The first observation is:

$$1 \leq M < 2 \implies 1 \leq \sqrt{M} < \sqrt{2} \tag{29}$$

We might choose the mean of both extrema as the starting point $x_0 = \frac{1}{2}(1+\sqrt{2})$ for instance.

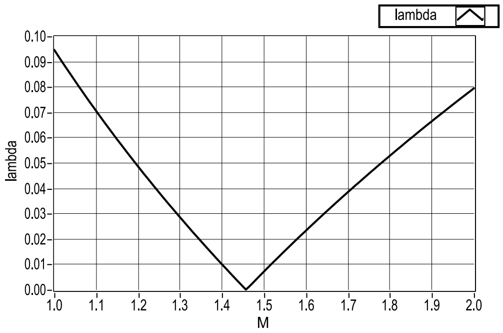


Figure 2: Evolution of $\lambda = |\hat{\mu}|\delta_0$ in the case of $x_0 = \frac{1}{2}(1+\sqrt{2})$, $\forall M \in [1,2)$.

If the starting value $x_0 = \frac{1}{2}(1+\sqrt{2})$ is applied for all values $M \in [1,2)$, the worst case appears for $M = 1$, as illustrated by Fig. 2 In that case $|\hat{\mu}| = \frac{1}{2}$ and $\delta_0 = 0.189334$.

$$\begin{aligned}\eta &= \frac{-7+\log 0.5}{\log 0.5+\log 0.189334} \approx 7.1313 \\ n &= \left\lceil \frac{\log \eta}{\log 2} \right\rceil + 1 = 4\end{aligned} \tag{30}$$

This is not a bad result, as it means that the square root can be calculated for any number in the range of the floating point representation using just 4 loops.

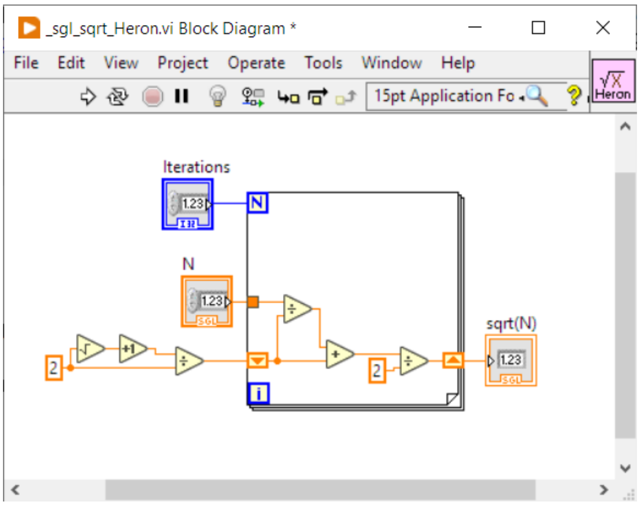


Figure 3: Basic implementation of Heron's square root algorithm for arguments $\in (1,2)$ using $x_0 = \frac{1}{2}(1+\sqrt{2})$.

2.4 Implementation of Heron's square root algorithm using linear approximation of the initial guess x_0

But, we can do better!

The improvement consists in using a linear approximation $x_0 = a(M) = \beta M + \gamma$, where β and γ are constants, as recommended by Crenshaw (2000) [p. 54ff.], in order to determine a value x_0 in the vicinity of $\sqrt{M}$. At this point, we impose the condition $x_0^2 < M$, which is essentially an arbitrary choice we make. We do this because we will require these equations in a subsequent section of this paper.

We proceed empirically, as deriving an analytical solution for the optimal function $a(M)$ that minimizes the number of iterations would be excessively complex and not worth the effort. A good solution should suffice.

We have seen in Eq. 16 that the number of iterations n varies in function of $\hat{\mu}$ and δ_0, where $\hat{\mu}$ depends on M alone and δ_0 on both, the initial value x_0 and M.

Now, we seek for a linear function that approximates $\sqrt{M}$ so well that the error $|\sqrt{M} - x_0|$ is possibly smallest. Therefore, we (arbitrarily) choose the points $(1,1)$ and $(2,\sqrt{2})$ as reference points, for which the respective values δ_0 are 0.

Admitting $a(M) = \beta M + \gamma$, where β and γ are constants determined by:

$$\begin{cases} 2\beta + \gamma = \sqrt{2} \\ \alpha + \beta = 1 \end{cases} \tag{31}$$

$$\Longrightarrow \begin{cases} \beta = \sqrt{2}-1 \\ \gamma = 2-\sqrt{2} \end{cases} \tag{32}$$

This linear fit guarantees that, except for the extrema, $a(N)^2 < N$, as shown in Fig. 4.

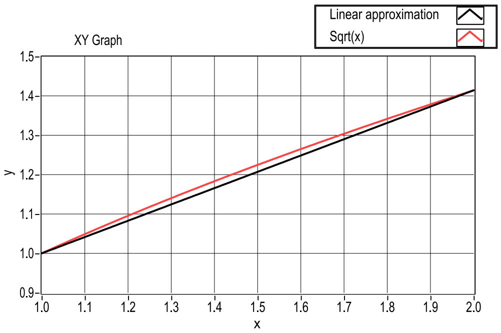


Figure 4: Linear approximation of the square root function in the interval [1,2).

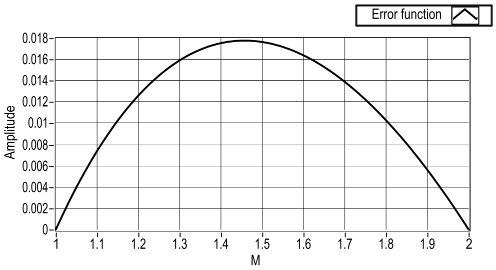


Figure 5: Error of the linear approximation of the square root function in the interval [1,2).

The error function is:

$$h(M) = \sqrt{M} - a(M) = \sqrt{M} - \beta M - \gamma \tag{33}$$

This function is maximal for

$$\begin{aligned} h'(M_m) &= 0 \\ h'(M_m) &= \frac{1}{2\sqrt{M_m}} - \beta \\ \Longrightarrow M_m &= \frac{1}{4\beta^2} \approx 1.457107 \\ a(M_m) &= \beta M_m + \gamma = \frac{\beta}{4\beta^2} + \gamma = \frac{1}{4\beta} + \gamma \\ &= \frac{\sqrt{2}+1}{4(2-1)} + 2 - \sqrt{2} \\ &= \frac{\sqrt{2}+1+8-4\sqrt{2}}{4} = \frac{-3\sqrt{2}+9}{4} \approx 1.189334 \end{aligned} \tag{34}$$

We can easily verify that $a(M_m) < \sqrt{M_m}$.

From the sign of $h'(M)$ we can conclude that $\forall M \in (1,2)\ \ a(M) < \sqrt{M}$:

Supposing:

$$\begin{aligned} & M < M_m \\ \iff & \sqrt{M} < \sqrt{M_m} \\ \iff & \frac{1}{2\sqrt{M}} > \frac{1}{2\sqrt{M_m}} \\ \iff & \frac{1}{2\sqrt{M}} - \beta > 0 \end{aligned} \tag{35}$$

The error function $h(M)$ therefore is increasing for all $M < M_m$ and conversely decreasing for $M > M_m$. Hence, $a(M) < \sqrt{M}$, $\forall M \in (1,2)$.

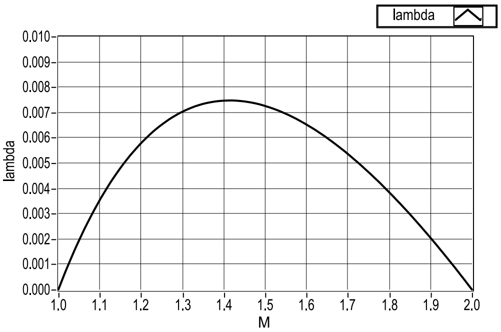


Figure 6: Evolution of $\lambda = |\hat{\mu}|\delta_0$ in the case of $x_0 = a(M) = \beta M + \gamma, \quad \forall M \in [1,2)$.

On Fig.6 we can observe that λ is maximal for $M \approx 1.414$. This yields the worst case: $x_0 = a(M) = 1.1715$, $|\hat{\mu}| = 0.42048$, $\delta_0 = 0.017749$.

$$\begin{aligned} \eta &= \frac{-7 + \log 0.42048}{\log 0.42048 + \log 1.17749E - 2} \approx 3.467814 \\ n &= \left\lceil \frac{\log \eta}{\log 2} \right\rceil + 1 = 3 \end{aligned} \tag{36}$$

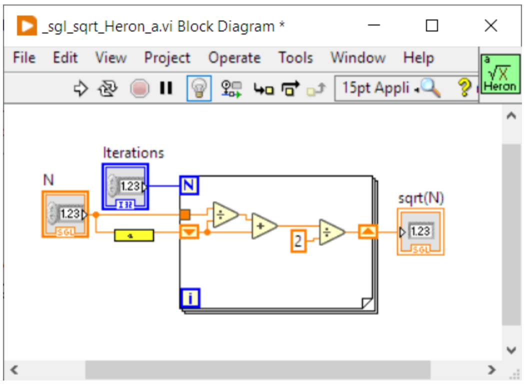


Figure 7: Basic implementation of Heron's square root algorithm for $M \in (1,2)$ using linear approximation of the initial guess.

2.5 Fitting the approximation line

The linear approximation can possibly be improved, if we choose the line so that errors are not one-sided. In that case, the line cuts the $\surd$ curve in two points, and the maximum error is halved. The equation of the new line can be obtained by a linear fit method, or pragmatically by simply adding half the maximal error (≈ 0.09) to $a(M)$, as shown in Fig. 8 and 9, which results in a vertical shift of the approximation line:

$$a_1(M) = \beta M + \gamma + 0.09 \tag{37}$$

Note that the initial arbitrary condition $a(M) \leq \sqrt{M}$ is no longer fulfilled.

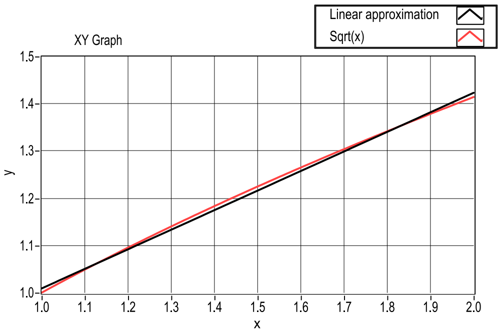


Figure 8: Better linear approximation $a_1(M)$ of the square root function in the interval $[1,2)$.

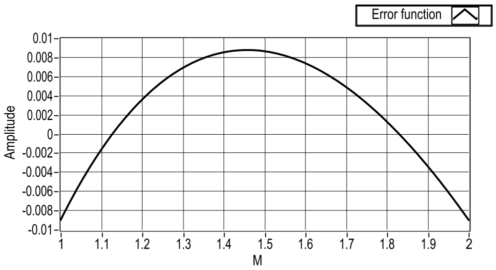


Figure 9: The error pattern of the improved linear approximation of the square root function in the interval $[1,2)$ does not change, although absolute errors are halved.

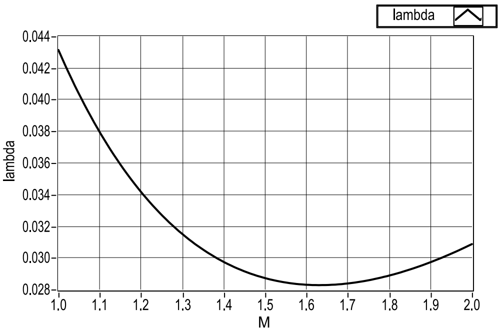


Figure 10: Evolution of λ in the case of $x_0 = a(M) = \beta M + \gamma + 0.09, \quad \forall M \in [1,2)$.

In this case, we have to evaluate δ_{max} and $\hat{\mu}$ for the three points, for which the absolute error is maximal $(1,1)$, $(M_m, \sqrt{M_m}) \approx (1.457107, 1.189334)$ and $(2, \sqrt{2})$:

M	$a(M)+0.09$	δ_0	$\lvert\hat{\mu}\rvert$	$\delta_0\lvert\hat{\mu}\rvert$	η
1	1.09	0.086284	0.5	0.04	5.348357
1.457107	1.279334	0.070188	0.414214	0.028995	4.8048956
2	1.504214	0.087308	0.353553	0.030868	4.933187

Table 4: Reducing the approximation error of the initial value x_0 does not necessarily produce a smaller number of iterations in the worst case.

Although approximation error of the initial guess x_0 now is smaller, the efficiency of the algorithm has unexpectedly worsened (slightly).

$$\begin{aligned} \eta &= \frac{-7+\log 0.5}{\log 0.5+\log 0.86284} \approx 5.348357 \\ n &= \left\lceil \frac{\log \eta}{\log 2} \right\rceil + 1 = 3 \end{aligned} \tag{38}$$

Important notice: By try and error, we could empirically define an excellent linear approximation function:

$$a_{1bis}(M) = \beta M + \gamma + 0.01 \tag{39}$$

2.6 Quadratic estimation for x_0

We may choose three points (1,1); $(2,\sqrt{2})$ and $(M_m, \sqrt{M_m})$[1] as reference points for the determination of the quadratic estimation $a_2(M) = AM^2 + BM + C$:

$$\begin{aligned} &\begin{cases} 1. \quad 4A+2B+C = \sqrt{2} \\ 2. \quad A+B+C = 1 \\ 3. \quad M_m^2 A + M_m B + C = \sqrt{M_m} \end{cases} \\ &\Longrightarrow \begin{cases} 1'. = 1. - 2. \quad 3A+B = \sqrt{2}-1 \\ 2'. = 3. - 2. \quad (M_m^2-1)A + (M_m-1)B = \sqrt{M_m}-1 \end{cases} \\ &Det = 3(M_m-1) - (M_m^2-1) = (M_m-1)(2-M_m) \approx 0.248160 \\ &A = \frac{(\sqrt{2}-1)(M_m-1) - \sqrt{M_m}+1}{Det} \approx -0.0715947 \\ &B = \frac{3(\sqrt{M_m}-1) - (\sqrt{2}-1)(M_m^2-1)}{Det} \approx 0.628998 \\ &C = 1-A-B \approx 0.442597 \end{aligned} \tag{40}$$

The function a_2 can be cascaded by Horner scheme:

$$a_2(M) = (AM+B)M + C \tag{41}$$

[1] cf Eq. 34

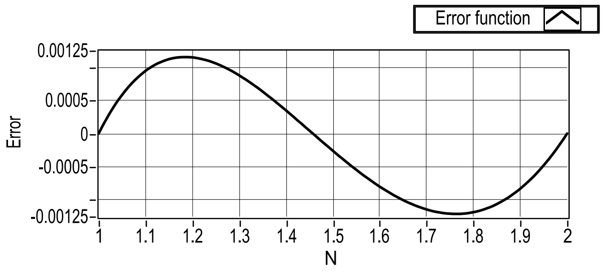


Figure 11: Error of the quadratic approximation of the square root function in the interval [1,2).

From Fig. 11 we may estimate taking $M_q \approx 1.18$ (We leave it to the reader to calculate the exact value):

$$a_2(M_q) = (AM_q + B)M_q + C \approx 1.085126 \tag{42}$$

$$\begin{aligned} \delta_0 &\approx 1.152661E-3 \\ \mu &= \frac{\sqrt{M_q}}{2M_q} \approx 0.460287 \end{aligned} \tag{43}$$

$$\begin{aligned} \eta &= \frac{-7+\log 0.460287}{\log 0.460287 + \log 1.152661E-3} \approx 1.163571 \\ n &= \left\lceil \frac{\log \eta}{\log 2} \right\rceil + 1 = 2 \text{ !!} \end{aligned} \tag{44}$$

3 Comparison of the different x_0 approximation methods

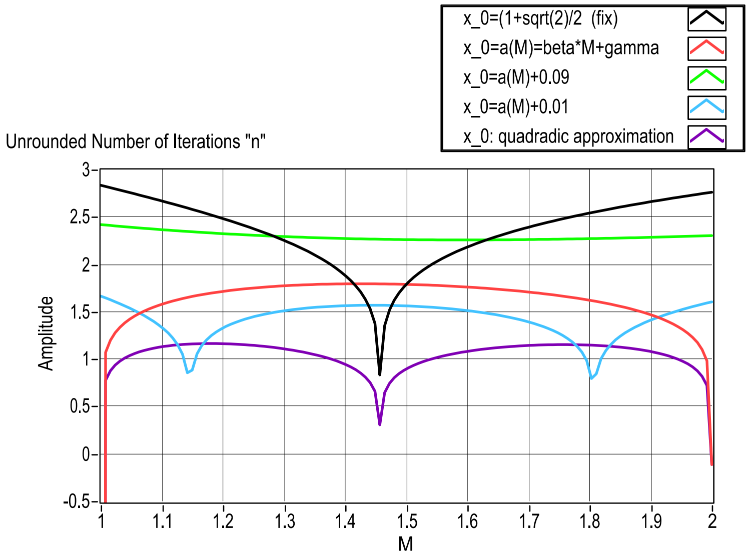


Figure 12: Comparing the (unrounded) number of iterations yielded empirically using Eq. 16 and 17 (single precision: $\epsilon_{SGL} = 2^{-23} \approx 1.19E-07$).

Fig. 12 compares the effects of the different approximation methods for the initial guess x_0 for $M \in [1,2)$ on the number of required iterations of Heron's algorithm. Interestingly the simple linear approximation and the minimally corrected linear approximation method (by adding 0.01) produce comparable results to the quadratic approximation.

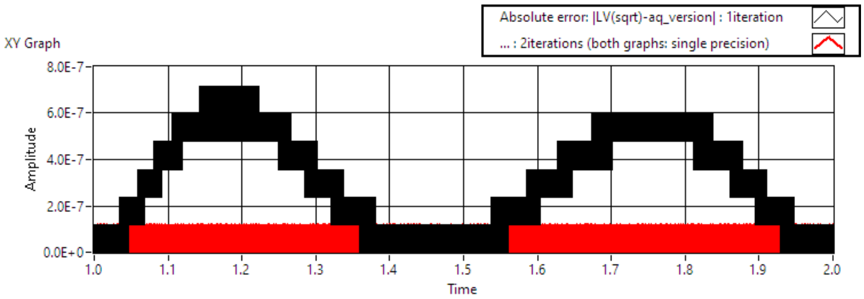


Figure 13: Absolute error between LABVIEW 2021 single precision square root and the presented algorithm using quadratic approximation for x_0 and either 1 or 2 Heron iterations (200000 samples)

Fig. 13 shows the absolute error between the standard single precision square root in the interval $[1,2)$, calculated in the LABVIEW 2021 environment with machine epsilon $\epsilon_{SGL} = 1.19209E-7$ and the applied algorithm with quadratic approximation of the initial value x_0, and either 1 or 2 Heron iterations.

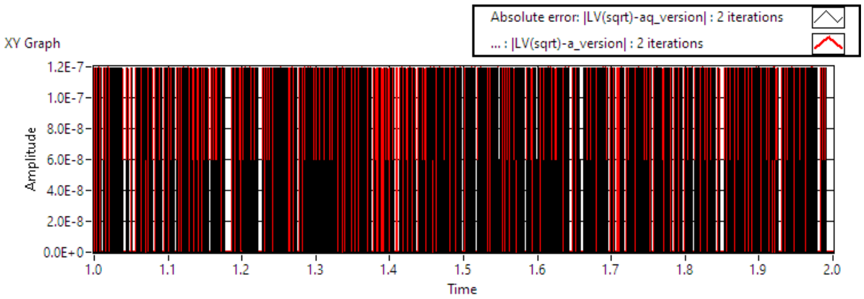


Figure 14: Absolute error in the cases of the linear and the quadratic approximation methods of x_0 using 2 iterations.

Fig. 14 confirms the prediction of accuracy through the present analysis: using the simple linear approximation method together with 2 Heron iterations delivers the same result as the quadratic version.

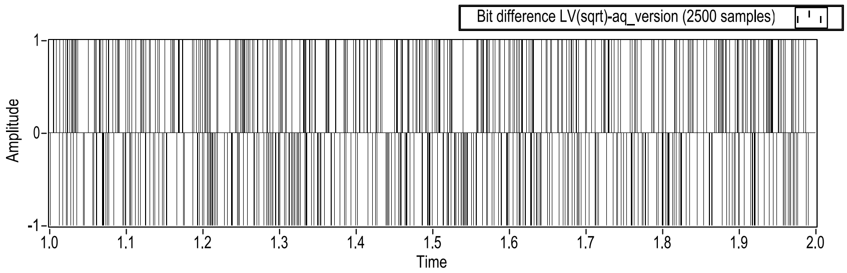


Figure 15: Error in the lowest mantissa bits (ulp) in the case of 2 Heron iterations (2500 samples).

Fig. 15 illustrates that the error introduced by the linear approximation of x_0, using two Heron iterations, never exceeds the lowest mantissa bit. However, the algorithm occasionally either overshoots or underestimates the standard LabVIEW square root by a single bit. Notably, the current implementation of the algorithm is not foolproof in terms of error propagation, as evidenced by differences observed in the ϵ bit. While this paper acknowledges the issue, a detailed discussion about rounding at the least significant bit level falls outside its scope.

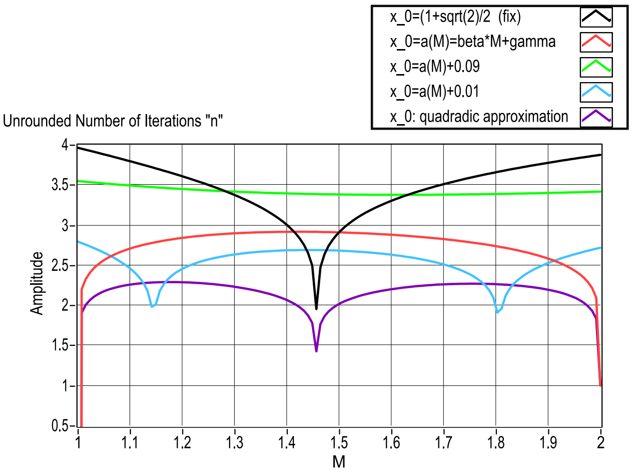


Figure 16: Comparing the (unrounded) number of iterations yielded empirically using Eq. 16 and 17 (double precision: $\epsilon_{DBL} = 2^{-52} \approx 2.22E-16$).

Fig. 16 shows that adding a single Heron iteration delivers double precision, while Fig. 17 suggests that the slightly altered linear approximation (adding 0.01) yields extended precision with 3 Heron iterations only. Definitely, Heron is hard to beat!

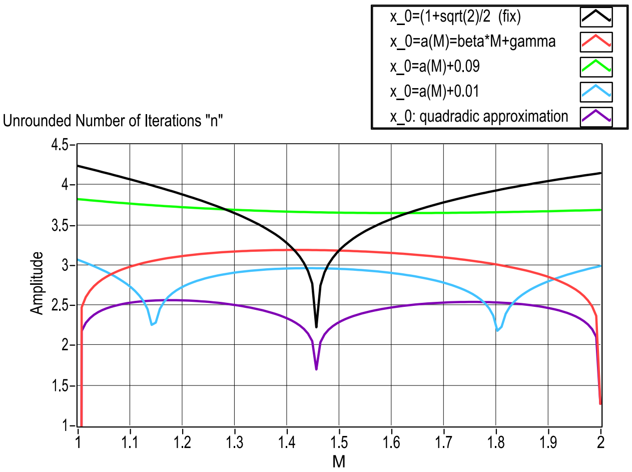


Figure 17: Comparing the (unrounded) number of iterations yielded empirically using Eq. 16 and 17 (extended precision: $\epsilon_{EXT} = 2^{-63} \approx 1.19E-19$).

Part II
Continued Fractions

We pursue our analysis with an approach that fascinated geniuses like Fermat, Huygens, Wallis, Euler, Lagrange, Gauss, Galois and many more. The method is known as CONTINUED FRACTIONS, which is commonly used together with Euclide's algorithm to convert decimal numbers into rational fractions.

Continued fractions are masterfully explored in the work by Khinchin (1964). For an excellent introduction in French, refer to the work by Catalan (1849). A concise history of continued fractions is outlined in the publication by Widz (2009). The method has ancient origins, where it was employed in a concealed manner to approximate square roots, among other applications. In Western Europe, Fibonacci in the 12th century was among the first to specifically study continued fractions. During the Renaissance, Rafael Bombelli used this method to approximate the square root of 13. Euler established the fundamental theorem, which states that every real number can be represented by a simple continued fraction. If the number is rational, it has a finite simple continued fraction expansion. However, if it is irrational, its simple continued fraction expansion is infinite. Lagrange later delivered a formal proof that the square root of a non-square number exhibits a periodic continued fraction expansion.

4 Example 1: Calculate the positive root of $x^2-3x-1=0$

Olds (1963) [p. 5] introduces continued fractions with an example using Euler's method of calculating the roots of the quadratic equation. We want to develop this example here:

$$x^2-3x-1=0 \tag{45}$$

Obviously $x \neq 0$. We may rearrange the equation and divide both sides by x. It now has the form:

$$x=3+\frac{1}{x} \tag{46}$$

This equation can be regarded as a recursive definition of the unknown variable x:

$$x=3+\frac{1}{x}=3+\frac{1}{3+\frac{1}{x}} \tag{47}$$

which can be continued as many times as desired.

$$x=3+\frac{1}{3+\frac{1}{3+\frac{1}{3+\frac{1}{x}}}} \tag{48}$$

Visibly, the recursion has no termination condition. At first sight, there is no real gain of rewriting Eq. 45 in the described manner: we don't know, if x is unique; we don't know its sign; we have no clue, whether the sequence of successively stopped (reduced) fractions converges.

Let's assume $x>1$, and try out by stopping the fraction at the ith term, which consists in simply ignoring the last appearance of $\frac{1}{x}$:

$$\begin{array}{llll} x_0=3 & x_1=3+\frac{1}{3}\approx 3.333333 & x_2=3+\frac{1}{3+\frac{1}{3}}=3.300000 & x_3=3+\frac{1}{3+\frac{1}{3+\frac{1}{3}}}\approx 3.303030 \\ x_4\approx 3.302752 & x_5\approx 3.302777 & x_6\approx 3.3027756 & \dots \quad \dots \end{array} \tag{49}$$

Clearly, the sequence converges to a limit, which must represent the positive root of Eq. 45.

From algebra we know how to find the quadratic roots, which are conjugates:

$$x_1 = \frac{3+\sqrt{9+4}}{2} \approx 3.3027756 \qquad x_2 = \frac{3-\sqrt{13}}{2} \approx -0.3027756 \tag{50}$$

5 Definition and properties

A **simple infinite continued fraction** is defined as:

$$\begin{aligned} x &= a_0 + \cfrac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}, \quad a_i \in \mathbb{N}^* \\ x &= [a_0; a_1, a_2, a_3, \dots] \end{aligned} \tag{51}$$

Some properties:

1. The sequence of a reduced infinite continued fraction converges to a limit.
2. The square root of a non-square positive real number can be represented by a simple infinite continued fraction.
3. The continued fractions expansion of the square root of a non-square positive real number is periodic.

6 Special form of infinite continued fraction

We introduce a special form of infinite continued fraction that keeps the convergence property and may be used to find quadratic roots effortlessly.

Let

$$\begin{aligned} x &= a + \cfrac{b}{a + \frac{b}{a + \frac{b}{a + \dots}}}, \quad a, b \in \mathbb{Q}^* \\ x &= [a, b] \end{aligned} \tag{52}$$

7 Example 2: Calculate $1+\sqrt{2}$ using special continued fractions

We may calculate $\sqrt{2}$ easily by writing the quadratic equation:

$$\big(x-(1-\sqrt{2})\big)\big(x-(1+\sqrt{2})\big) = x^2-2x-1=0 \tag{53}$$

Following Olds (1963) example, we can rearrange this equation to a recursive form:

$$x = 2 + \frac{1}{x} \tag{54}$$

from which we can conclude that:

$$\sqrt{2}+1 = 2 + \cfrac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \dots}}}} = [2, 1] \tag{55}$$

8 Method of computing $1+\sqrt{N}$ using special continued fractions

If we consider the conjugates $1+\sqrt{N}$ and $1-\sqrt{N} \quad N \in \mathbb{R}^*$ with $N > 1$, we may set up the quadratic equation:

$$\big(x-(1-\sqrt{N})\big)\big(x-(1+\sqrt{N})\big)=x^2-2x-(N-1)=0 \tag{56}$$

from which we get through rearrangement and division by x:

$$x=2+\frac{N-1}{x} \tag{57}$$

hence

$$\begin{aligned}\sqrt{N}+1=2+\cfrac{N-1}{2+\frac{N-1}{2+\frac{N-1}{2+\frac{N-1}{2+\dots}}}}&=[2,N-1] \quad \text{or}\\ \sqrt{N}=[2,N-1]-1&\end{aligned} \tag{58}$$

We may write Eq. 57 as a recursive sequence:

$$x_{i+1}=2+\frac{N-1}{x_i}, \quad i \in \mathbb{N}, \quad \text{starting with } x_0>0 \tag{59}$$

Practically, we choose as the starting value $x_0 = 2$ (Discussion later in Section 11).

As we observe, the decision to seek an expansion for $\sqrt{N}+1$ rather than N alters the well-documented algebraic periodicity of standard continued fractions, resulting in a constant period length of 1 for any square root (cf. Theorem 6.15 in Wagstaff Jr. (2013) [p.150]).

9 Implementation of the algorithm

Preliminary note: In the following, we provide algorithm implementations without explicit termination conditions. Instead, we employ a fixed number of iterations. While this approach may appear unconventional initially, it becomes evident that it is effective for the described algorithms. By determining the worst-case number of iterations a priori and limiting it to a few cycles, we eliminate the need to check for desired precision. The benefit is twofold: fewer computations per iteration and constant computing time.

Eq. 59 leads to the minimalist program code shown in Fig. 18 and Listing 1.

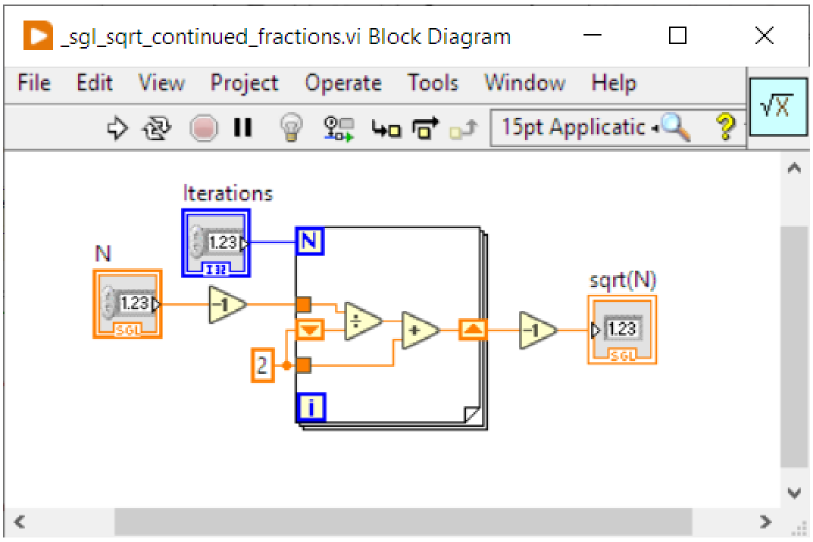

_sgl_sqrt_continued_fractions.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Applicatic
√X
Iterations
1.23
I32
N
1.23
SGL
N
-1
÷
+
2
i
-1
sqrt(N)
1.23
SGL

Figure 18: LabVIEW adaptation of the algorithm.

```
def my_sqrt(N, iterations):
    a=2; b=N-1
    x=a
    for i in range (iterations):
        x=b/x+a
    result=x-1
```

Listing 1: Python implementation

10 Convergence of the algorithm

i	x_i	$x_{i+1}-x_i$
0	2	0.5
1	2.5	-0.1
2	2.4	0.0166667
3	2.166667	-0.002874
4	2.413793	0.000493
5	2.414286	-8.453085E-5
6	2.414201	1.450284E-5
7	2.414216	-2.488305E-6
8	2.414213	4.269253E-7

Table 5: Sequence values

If we consider the function:

$$f(x)=2+\frac{N-1}{x} \tag{60}$$

we get the fixed points: $x_{*,1}=1+\sqrt{N}$ and $x_{*,2}=1-\sqrt{N}$ by solving:

$$\begin{aligned} x &= 2 + \frac{N-1}{x} \\ \Longleftrightarrow\ x^2 &= 2x + (N-1) \\ \Longleftrightarrow\ x^2 - 2x - (N-1) &= 0 \end{aligned} \tag{61}$$

This is our initial equation Eq. 56. The function Eq. 60 is continuous and differentiable.

Using the Attracting Fixed Point Theorem, stating that:

$$\mid f'(x_*) \mid < 1 \tag{62}$$

is sufficient to prove the attracting property.

We are interested in $x_{*,1} = 1 + \sqrt{N}$ only:

$$\mid f'(x_{*,1}) \mid = \frac{N-1}{x_{*,1}^2} = \frac{N-1}{\left(1+\sqrt{N}\right)^2} < 1 \ (!) \tag{63}$$

which proves the convergence of sequence Eq. 59.

10.1 Rate of convergence

$$x_{i+1} - x_i = \frac{N-1}{x_i} + 2 - x_i = -\frac{x_i^2 - 2x_i - (N-1)}{x_i} = \frac{N-(1-x_i)^2}{x_i} \tag{64}$$

$$\delta_i = \mid x_{i+1} - x_i \mid \tag{65}$$

$$\begin{aligned} x_{i+2} - x_{i+1} &= \frac{N-1}{x_{i+1}} + 2 - x_{i+1} \\ &= \frac{N-1}{\frac{N-1}{x_i} + 2} + 2 - \left(\frac{N-1}{x_i} + 2\right) \\ &= \frac{x_i^2(N-1)}{x_i(N-1+2x_i)} - \frac{(N-1)(N-1+2x_i)}{x_i(N-1+2x_i)} \\ &= \frac{(N-1)\left(x_i^2 - 2x_i - (N-1)\right)}{(N-1+2x_i)x_i} \\ &= -\frac{N-1}{N-1+2x_i}(x_{i+1} - x_i) \end{aligned} \tag{66}$$

Eq. 66 describes a *linear* convergence and shows that the sequence (Eq. 59) is oscillating about the fixed point, because the sign of the iterated differences change from step to step, and the factor:

$$\mu_i = \frac{N-1}{N-1+2x_i} > 0 \tag{67}$$

For sufficiently large i, or x_i sufficiently close to $1+\sqrt{N}$ we have:

$$\begin{aligned} & x_i \approx 1 + \sqrt{N} \\ \Longrightarrow\ & \hat{\mu} = \frac{N-1}{N-1+2(1+\sqrt{N})} = \frac{N-1}{N+2\sqrt{N}+1} = \frac{\sqrt{N}-1}{\sqrt{N}+1} \end{aligned} \tag{68}$$

Examples:

$$\begin{aligned} \hat{\mu}(N=2) &= \frac{\sqrt{2}-1}{\sqrt{2}+1} \approx 0.171573 \\ \hat{\mu}(N=5) &= \frac{\sqrt{5}-1}{\sqrt{5}+1} \approx 0.381966 \end{aligned} \tag{69}$$

Therefore, the difference $\delta_i = \mid x_{i+1} - x_i \mid$ may be approximated with:

$$\delta_i \approx \hat{\mu}^i \mid x_1 - x_0 \mid \tag{70}$$

10.2 Example 3: Convergence in the case of $N = 2$

Using Eq. 67 and Table 5, we get:

i	$\mid x_{i+1} - x_i \mid$	μ_i
0	0.5	0.2
1	0.1	0.166667
2	0.166667	0.172414
3	0.002874	0.171429
4	0.000493	0.171598
5	8.453085E-5	0.171569
6	1.450284E-5	0.171574
7	2.488305E-6	0.171573
8	4.269253E-7	0.171573

Table 6: Error development

As can be seen in Table 6, we almost gain one digit precision per iteration.

10.3 Example 4: Convergence in the case of $N = 5$

i	x_i	$x_{i+1} - x_i$	μ_i
0	2	2	0.5
1	4	-1	0.333333
2	3	0.333333	0.4
3	3.333333	-0.133333	0.375
4	3.2	0.05	0.384615
5	3.25	-0.019231	0.380952
6	3.230769	0.007326	0.382353
7	3.238095	-0.002801	0.381818
8	3.235294	0.00107	0.382022
9	3.236364	-0.000409	0.381944
10	3.235955	0.000156	0.381974
11	3.236111	-5.960897E-5	0.381963
12	3.236052	2.276841E-5	0.381967
13	3.236074	-8.696787E-6	0.381966
14	3.236074	3.321873E-6	0.381966

Table 7: The number of iterations needed to reach high precision increases with growing N.

11 Using algorithm $x_{i+1} = 2 + \frac{M-1}{x_i}$ on the mantissa $M \in [1,2)$

We already saw that normalized mantissas of single precision floating point numbers all belong to the interval $[1,2)$. We want to calculate $\sqrt{M}$ using Eq. 59.

The number of worst case iterations in the case of $x_0 = 2$ can be calculated as follows from Eq. 59:

$$\delta_0 = \mid x_1 - x_0 \mid = \frac{M-1}{2} \tag{71}$$

This value is maximal for $M = 2$ with $\delta_{max} = \frac{1}{2}$.

The worst case number of iterations n required to get maximal single precision precision $p \approx 1E-7$ is:

$$\begin{aligned} p &= \delta_n = \delta_{max} \cdot \hat{\mu}^n_{M=2} \\ \Longrightarrow\ n &= \left\lceil \frac{\log p - \log \delta_{max}}{\log \hat{\mu}_{M=2}} \right\rceil \approx \left\lceil \frac{-7 + 0.30103}{-0.765551} \right\rceil = 9 \end{aligned} \tag{72}$$

We can improve this by using the linear approximation of x_0 presented in Section 2.4. Note however that we must set x_0 close to $1+\sqrt{M_m}$, so each appearance of $a(M_m) = \beta M_m + \gamma$ must be incremented by 1.

$$\begin{aligned} \delta_{max} &= \left| \frac{(1 - a(M_m) - 1)^2 - M_m}{a(M_m) + 1} \right| \approx \left| \frac{(1.189334)^2 - 1.457107}{2.189334} \right| \approx 0.0194542 \\ \hat{\mu}_{M_m} &= \frac{\sqrt{M_m} - 1}{\sqrt{M_m} + 1} \approx \frac{\sqrt{1.457107} - 1}{\sqrt{1.457107} + 1} \approx 0.093836 \\ n &= \left\lceil \frac{\log p - \log \delta_{max}}{\log \hat{\mu}_{M_m}} \right\rceil \approx \left\lceil \frac{-7 + 1.710988}{-1.027629} \right\rceil = 6 \end{aligned} \tag{73}$$

12 Generalized form $x_{i+1} = 2a + \frac{N-a^2}{x_i}$

First, we allow $a \in \mathbb{Q}_+^*$.

$$\left(x - (a - \sqrt{N})\right)\left(x - (a + \sqrt{N})\right) = x^2 - 2ax - (N - a^2) = 0 \ \text{ with } 1 \le a^2 \le N \tag{74}$$

from which we get through rearrangement and division by x:

$$x = 2a + \frac{N - a^2}{x} \tag{75}$$

We call a the initial guess for the square root.

This delivers the general algorithm:

$$x_{i+1} = 2a + \frac{N - a^2}{x_i} \quad \text{starting with } x_0 > 0 \tag{76}$$

$$x_{i+1} - x_i = 2a + \frac{N - a^2}{x_i} - x_i = -\frac{x_i^2 - 2ax_i - (N - a^2)}{x_i} = \frac{N - (a - x_i)^2}{x_i} \tag{77}$$

We leave it to the reader to perform the convergence calculation. It turns out that the generalized case also has *linear* convergence with:

$$\mu_i = \frac{N - a^2}{N - a^2 + 2ax_i} \tag{78}$$

and for sufficiently large i:

$$\begin{aligned} & x_i \approx a + \sqrt{N} \\ \Longrightarrow\ & \hat{\mu} = \frac{N - a^2}{N - a^2 + 2a(a + \sqrt{N})} = \frac{N - a^2}{N + a^2 + 2a\sqrt{N}} = \frac{\sqrt{N} - a}{\sqrt{N} + a} \end{aligned} \tag{79}$$

12.1 Minimizing $\hat{\mu} = \frac{\sqrt{N}-a}{\sqrt{N}+a}$

If we consider $1 \leq y \leq \sqrt{N}$ and the function:

$$\zeta(y) = \frac{\sqrt{N} - y}{\sqrt{N} + y} \tag{80}$$

we may conclude from its derivative:

$$\zeta'(y) = -\frac{2\sqrt{N}}{(\sqrt{N} + y)^2} \tag{81}$$

that $\zeta(y)$ is strictly decreasing, and is 0 for $y = \sqrt{N}$.

Therefore, we may keep $\hat{\mu}$ small, if a is a good approximation of $\sqrt{N}$.

12.2 Minimizing δ_0 by choosing a good initial value x_0

If we consider the function:

$$\zeta(y) = \frac{N - (a - y)^2}{y} \tag{82}$$

Given the discussion about a, this function may be considered smallest, if $y = 2a$.

13 Using algorithm $x_{i+1} = 2a + \frac{M - a^2}{x_i}$ on the mantissa $M \in [1, 2)$

If we restrict the algorithm Eq. 76 to the interval of the floating point mantissa, it is possible to optimize the number of iterations as can be seen in Fig. 19, 20 and 21.

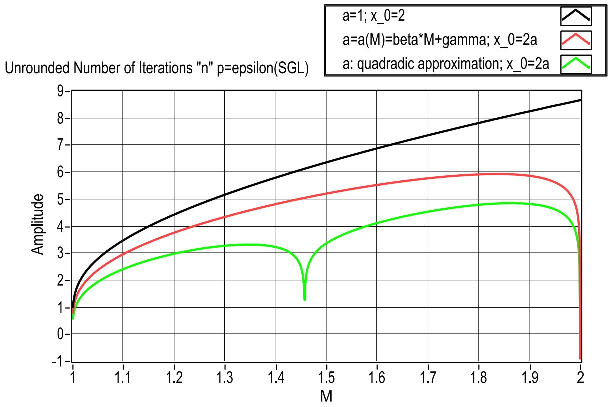


Figure 19: Comparing the (unrounded) number of iterations yielded empirically using Eq. 72 (single precision).

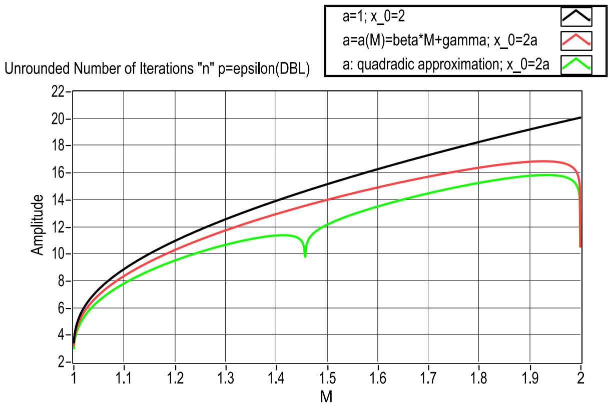


Figure 20: Comparing the (unrounded) number of iterations (double precision).

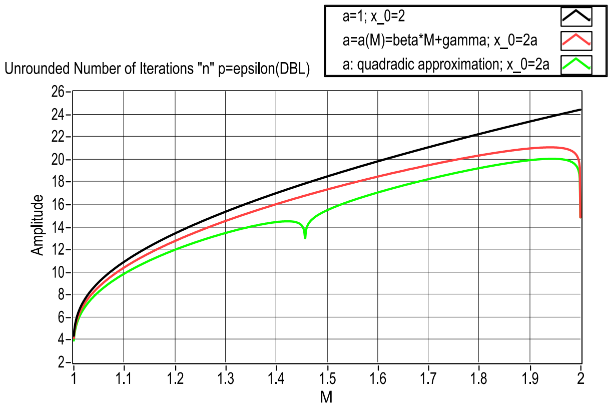


Figure 21: Comparing the (unrounded) number of iterations (extended precision).

13.0.1 Implementation in LabVIEW

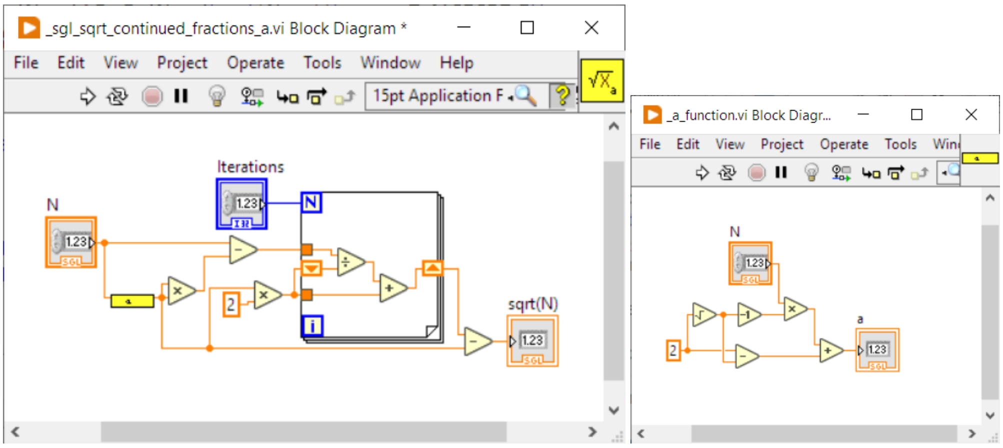


Figure 22: The basic square root function for $N \in (1,2)$ using linear approximation for the initial guess x_0.

13.1 Conclusion

This algorithm can be used to calculate floating-point square roots. Before performing the iterations, a few elementary operations are required. Each iteration involves a single division and an addition only. However, due to the linear convergence of the sequence and the dependence of the convergence rate on the argument N (or M, if the mantissa is being used), there is no real interest of following this method instead of Heron's, except for the demonstration that advanced continued fractions exploiting the 1-period property of improper square roots $a+\sqrt{N}$ really approximate the square root.

Part III
No division method

Among the basic arithmetic operations, division holds a special place. Generally, it is more time-consuming than other operations. However, the most significant issues with division are rounding errors and the risk of dividing by zero. Even the simplest form of division, dividing by 2, which is often replaced by a simple right shift, is not immune to problems, as documented by Steele (1977).

It's no surprise that efforts have been made to find algorithms for calculating square roots without using division. Blanchard and Chamberland (2023) applied Newton's root-finding method to the inverse function with remarkable results. First, they show that the function $f(x)=1/x-a$ can be approximated to any precision using the quadratically convergent sequence $x_{i+1}=x_i(2-ax_i)$. Furthermore, the authors prove an efficient two-step division-free algorithm for computing square roots, credited to Lischka et al. (2012).

$$\begin{cases} y_{i+1} & = y_i(2-2x_i y_i) \\ x_{i+1} & = x_i-(x_n^2-N)y_{n+1} \end{cases} \tag{83}$$

where x_0 is an intial guess to $\sqrt{N}$ and $y_0=\frac{1}{2x_0}$.

This algorithm keeps Heron's quadratic convergence feature. We will not delve into this method here. Please refer to Blanchard and Chamberland (2023) for the details.

Instead, we will develop an even simpler method that avoids division. However, it's important to note from the outset that there subsists a single division by 2, which we recommend replacing with a right shift operation or subtraction of the exponent. So, very strictly speaking, the division is not totally eliminated. Also, the algorithm presented here has *linear* convergence only.

14 Example 3: Calculate the negative root of $x^2-3x-1=0$

We may rewrite $x^2-3x-1=0$ differently this time:

$$3x=x^2-1 \tag{84}$$

and turn the equation into a sequence starting with arbitrary $x_0=-0.75$:

$$x_{i+1}=\frac{x_i^2-1}{3} \tag{85}$$

i	x_i
0	-0.75
1	-0.145833
2	-0.326244
3	-0.297855
4	-0.303760
5	-0.302576
6	-0.302816
7	-0.302767
8	-0.302777
9	-0.302775

Table 8: The algorithm converges to the negative root of the equation $x^2 - 3x - 1 = 0$.

Table 8 shows the convergence of the algorithm to the negative root. It seems counter-intuitive that a root can be found by iteratively squaring numbers. There still is a division by a constant number implied.

15 Method of computing $1 - \sqrt{N}$ using multiplication, subtraction and shift operations

If we go back to Eq. 56, we may rewrite:

$$\begin{aligned} x^2 - 2x - (N-1) &= 0 \\ \Longleftrightarrow x &= \frac{x^2 - (N-1)}{2} \end{aligned} \tag{86}$$

Regarding the previous example, the question arises, whether the corresponding recursive sequence is capable of yielding $1 - \sqrt{N}$. Note that we still admit $N > 1$.

$$x_{i+1} = \frac{x_i^2 - (N-1)}{2} \tag{87}$$

If we consider:

$$\begin{aligned} f(x) &= \frac{x^2 - (N-1)}{2} \\ f'(x) &= x \end{aligned} \tag{88}$$

We know that $f(x)$ has two fixed points $1 + \sqrt{N}$ and $1 - \sqrt{N}$. We also see that $1 + \sqrt{N}$ is repelling, as:

$$| f'(1 + \sqrt{N}) | > 1 \tag{89}$$

However, we can find an interval, where $| f'(1 - \sqrt{N}) | < 1$. Condition:

$$| 1 - \sqrt{N} | < 1 \tag{90}$$

which is the case for all $1 < N < 4$.

Conclusion: We have found a counter-intuitive method for calculating the square root of numbers $N \in (0,4)$ using single multiplication, subtraction and division by 2. The latter can be replaced by a right shift in the case of unsigned integers and a simple decrement of the exponent in the case of floating point arithmetic, so that no division at all is implied.

16 Rate of convergence

$$\begin{aligned}
x_{i+1} - x_i &= \frac{x_i^2 - 2x_i - (N-1)}{2} = \frac{(x_i-1)^2 - N}{2} \\
x_{i+2} - x_{i+1} &= \frac{(x_{i-1}-1)^2 - N}{2} \\
&= \frac{\left(\frac{x_i^2-(N-1)}{2} - 1\right)^2 - N}{2} \\
&= \frac{x_i^4 - 2(N+1)x_i^2 + (N-1)^2}{8}
\end{aligned} \tag{91}$$

Because

$$\left(x_i^2 - 2x_i + (N-1)\right)^2 = x_i^4 - 2(N+1)x_i^2 + (N-1)^2 - 4x_i^3 + 8x_i^2 + 4(N-1)x_i \tag{92}$$

$$\begin{aligned}
\implies x_{i+2} - x_{i+1} &= \frac{((x_i - 2x_i - (N-1))^2 + 4x_i\left(x_i^2 - 2x_i - (N-1)\right)}{8} \\
&= \frac{\left((x_i^2 - 2x_i - (N-1)\right)\left(x_i^2 - 2x_i - (N-1) + 4x_i\right)}{8} \\
&= (x_{i+1} - x_i)\frac{x_i^2 + 2x_i - (N-1)}{4}
\end{aligned} \tag{93}$$

which describes a *linear* convergence with $\mu_i = \frac{x_i^2+2x_i-(N-1)}{4} = \frac{(x_i+1)^2-N}{4}$.

For sufficiently large i or x_i close to $1-\sqrt{N}$, we have:

$$\begin{aligned}
\hat{\mu} &= \frac{(2-\sqrt{N})^2 - N}{4} \\
&= 1 - \sqrt{N}
\end{aligned} \tag{94}$$

17 Using algorithm $x_{i+1} = \frac{x_i^2-(N-1)}{2}$ on the mantissa $M \in [1,2)$

First we note that the mantissa region of interest, i.e. the interval $[1,2)$ lies within the convergence boundary $(0,4)$. We use our method of finding the number of iterations needed to reach a certain given precision applying Eq. 87.

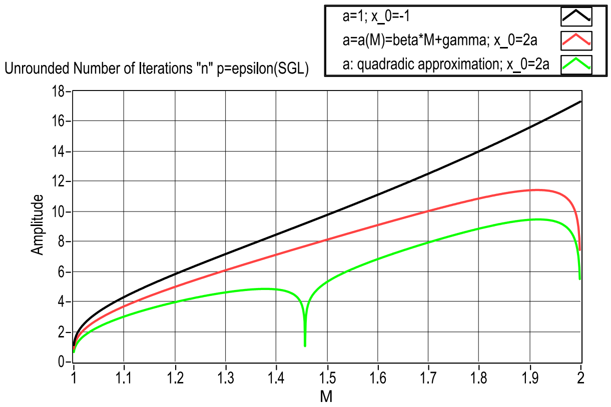


Figure 23: Comparing the (unrounded) number of iterations yielded using Eq. 72 (single precision).

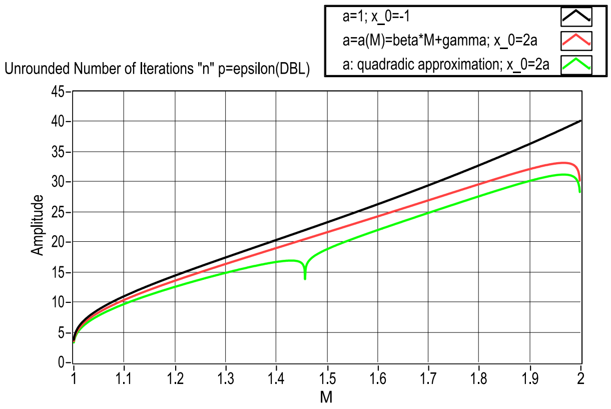


Figure 24: Comparing the (unrounded) number of iterations (double precision).

Fig. 23 shows that this algorithm might be interesting for the single-precision implementation in devices with limited arithmetic functions, especially if time-consuming division are not desired, whereas Fig. 24 clearly demonstrates that the algorithm is not indicated for higher precision representations due to the high number of iterations required.

Note that an improvement of the algorithm could be obtained by limiting the mantissa to the interval $[1, 1.5]$ using elementary operations on the expanded floating point data. For instance, we can apply the following equation on the mantissa:

$$\begin{aligned} &\text{constants } c_1 = \frac{\sqrt{2}}{2}, \quad c_2 = \sqrt[4]{2} \\ &\text{if } M > \sqrt{2} \implies \hat{M} = c_1 M \\ &\qquad\qquad\qquad \sqrt{M} = c_2 \sqrt{\hat{M}} \end{aligned} \tag{95}$$

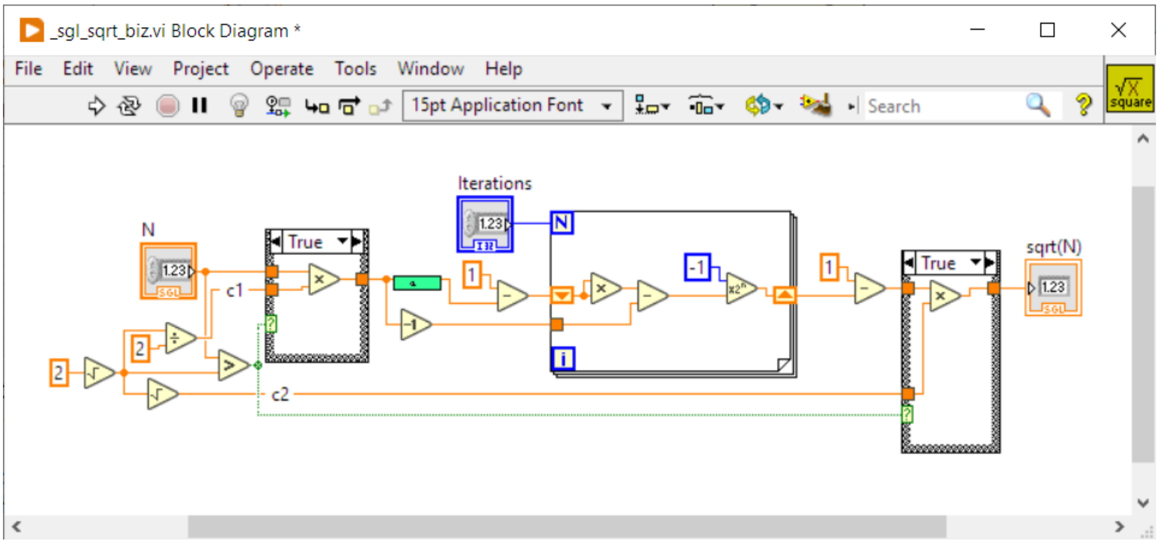


Figure 25: LabVEIW implementation of the no-division method using $x_{i+1} = \frac{x_i^2-(N-1)}{2}$ and Eq. 95 on the mantissa. 5 iterations are sufficient to gain ϵ_{SGL} precision.

Part IV
Toepler's Algorithm

In this section we will describe an algorithm that often appears in the literature without reference to its inventor August J. I. Toepler (1836-1912), a German chemist and physicist (cf. Releaux (1866)). For instance, Warren (2012) [p. 285] attributes the algorithm to von Neumann (1993); whereas Crenshaw (2000) [p. 72-87] calls it the Friden Algorithm, as it was implemented in the mechanical Friden SRW calculator.[2] Both authors describe binary transcriptions of the algorithm for the digital calculation of integer or fixed point square roots. The web-site of the HP-Museum (2024) presents an interpretation of the Friden algorithm that uses multiplications by 5 in order to simplify products by 2, which change to products by 10, which represent simple left shifts in the decimal system. Interestingly, this algorithm has been integrated into the legendary HP-35, the world's first hand-held scientific calculator, which was based on Binary-coded decimal (BCD) numbers (cf. Egbert (1977)). The excellent French wikipédia (2024) page refers to the spigot (*goutte à goutte*) method and draws a formal proof of the correctness of the algorithm.

We will exclusively speak of Toepler's Algorithm, since there may be no doubt about the authorship. A practical introduction to the algorithm can be found in Gärtner (2008). A more rigorous explanation is delivered by Jarvis AF (2005) and Goldberg (2023). Martin (1985) , Rolfe (1987) (based on Ming and Woo (2012)), Crenshaw (2000) [p. 86] and Warren (2012) present binary C implementations.

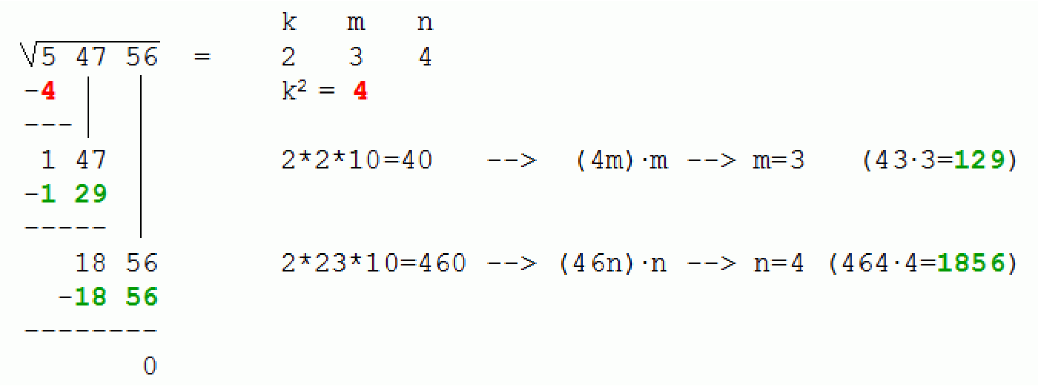
18 Learning Toepler's Algorithm in 5 Minutes (Decimal System)

Toepler's algorithm bears a resemblance to the school method. Just like in school, we group the digits of the number N in pairs. Now, here's where Toepler's ingenuity comes into play. He realized that to compute the square root of a number, we should look for a perfect square that is smaller than the highest pair of digits in the table. However, instead of guessing that number, Toepler employed a clever approach: he subtracted successive odd numbers, starting with 1, as long as the result remained positive. This strategy leverages the fact that the sum of the first n odd numbers is equal to n^2, making the method suitable for implementation in mechanical calculators.

At this point we'd like to cite Crenshaw (2000) [p.76], who wrote:

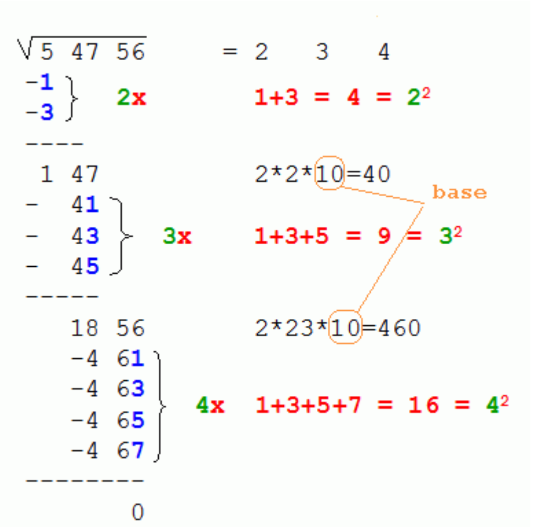
> „*In the following development... I urge you not to just look at it, but to actually perform the calculations yourself. That's the only way you can truly see how the algorithm works. In short, pretend you're a* Friden.“

[2] It is important to underline that Crenshaw's presentation may lead to some confusion. In fact, he describes an adapted version of Toepler's algorithm that was manually run on a non-square root Friden. It does not represent the embedded mechanical method that controlled the mechanical Friden SWR-10 square root model. Later however, the method had been integrated in the electronic Friden EC132.



```
                    k    m    n
√5 47 56    =       2    3    4
-4 |    |           k^2 = 4
---|    |
 1 47   |           2*2*10=40    -->  (4m)·m --> m=3    (43·3=129)
-1 29   |
-----   |
   18 56            2*23*10=460 --> (46n)·n --> n=4  (464·4=1856)
  -18 56
--------
       0
```

Figure 26: Tradition school method



```
√5 47 56      = 2    3    4
-1 }  2x        1+3 = 4 = 2^2
-3 }
----
 1 47           2*2*10=40
-   41 }                    base
-   43 }  3x    1+3+5 = 9 = 3^2
-   45 }
-----
    18 56       2*23*10=460
    -4 61 }
    -4 63 }  4x  1+3+5+7 = 16 = 4^2
    -4 65 }
    -4 67 }
--------
         0
```

Figure 27: Toepler's method

19 Formal description of the algorithm

Suppose we were able with some magic to determine the digits n_i of $\sqrt{N}$ one by one from the left to the right. Let m be the magnitude of that square root. It can be calculated by:

$$m = m(N) = \left\lfloor \frac{\log N}{2} \right\rfloor \qquad (96)$$

Examples: $m(132548) = 2$ and $m(54756) = 2$

We write the integer number as a finite series in base-10. Also, we admit at this point that N is a perfect square. (Note that Toepler's method can evidently be applied to fractional numbers. However, for clarity reasons, we admit that the numbers are aligned here as integer numbers. If N is not a perfect square, the final remainder defined in the following lines is non-zero and represents the fractional part of the square root.)

$$\begin{aligned}\sqrt{N} &= \overline{n_m n_{m-1} n_{m-2} ... n_2 n_1 n_0} \\ &= n_m 10^m + n_{m-1} 10^{m-1} + n_{m-2} 10^{m-2} + ... + n_2 10^2 + n_1 10^1 + n_0 10^0\end{aligned} \tag{97}$$

From the digits $n_m n_{m-1}, n_{m-2}, ...$ we can define successive perfect squares that approximate $\sqrt{N}$ with growing precision, one decade per step:

$$\begin{aligned}N_0 &= \overline{n_m 00..000}^2 \\ N_1 &= \overline{n_m n_{m-1} 0...000}^2 = \left(\overline{n_m 00...000} + \overline{0 n_{m-1} 0...000}\right)^2 \\ N_2 &= \overline{n_m n_{m-1} n_{m-2} ...000}^2 = \left(\overline{n_m n_{m-1} 0...000} + \overline{00 n_{m-2} ...000}\right)^2 \\ &... \\ N_{m-1} &= \overline{n_m n_{m-1} n_{m-2} ... n_1 0}^2 = \left(\overline{n_m n_{m-1} n_{m-2} ... n_2 00} + \overline{000...0 n_1 0}\right)^2 \\ N_m &= \overline{n_m n_{m-1} n_{m-2} ... n_2 n_1 n_0}^2 = \left(\overline{n_m n_{m-1} n_{m-2} ... n_1 0} + \overline{000...00 n_0}\right)^2 \\ & \\ \text{with } & N_0 \leq N_1 \leq N_2 \leq ... \leq N_{m-2} \leq N_{m-1} \leq N_m = N\end{aligned} \tag{98}$$

Applying the binomial equations, we can write the differences:

$$\begin{aligned}q_0 &= N_0 = \overline{n_m 00..000}^2 \\ q_1 &= N_1 - N_0 = \overline{0 n_{m-1} 0...000}^2 + 2 \cdot \overline{n_m 00...000} \cdot \overline{0 n_{m-1} 0...000} \\ q_2 &= N_2 - N_1 = \overline{00 n_{m-2} ...000}^2 + 2 \cdot \overline{n_m n_{m-1} 0...000} \cdot \overline{00 n_{m-2} ...000} \\ &... \\ q_m &= N_m - N_{m-1} = \overline{000...00 n_0}^2 + 2 \cdot \overline{n_m n_{m-1} n_{m-2} ... n_1 0} \cdot \overline{000...00 n_0}\end{aligned} \tag{99}$$

Or more generally:

$$\begin{aligned}& q_0 = n_m^2 10^{2m} \\ \forall k \geq 1 \quad & q_k = (n_{m-k} 10^{m-k})^2 + 2 n_{m-k} 10^{m-k} \sum_{i=0}^{k-1} n_{m-i} 10^{m-i} \\ & \quad = n_{m-k}^2 10^{2(m-k)} + n_{m-k} \cdot 2 \sum_{i=0}^{k-1} n_{m-i} 10^{2m-k-i}\end{aligned} \tag{100}$$

We can define successive decreasing remainders:

$$\begin{aligned}R_0 &= N \\ R_1 &= N - N_0 = R_0 - q_0 \\ R_2 &= N - N_1 = N - (N_0 + q_0) = R_1 - q_1 \\ &... \\ R_{m-1} &= N - N_{m-1} = R_{m-2} - q_{m-2} \\ R_m &= N - N_m = R_{m-1} - q_{m-1} \\ R_0 &\geq R_1 \geq R_2 ... \geq R_{m_1} \geq R_m \geq 0\end{aligned} \tag{101}$$

Note that for all N being a perfect square, R_m=0. This represents a recursive process:

$$\forall k \in \mathbb{N},\ k \leq m \quad R_{k+1} = R_k - q_k \tag{102}$$

Example:

$$\begin{aligned}
R_0 &= 54756 \\
R_1 &= 54756 - 200^2 = 54756 - 40000 = 14756 \\
R_2 &= 14756 - 30^2 - 30 \cdot 2 \cdot 200 = 14756 - 900 - 12000 = 1856 \\
R_3 &= 1856 - 4^2 - 4 \cdot 2 \cdot 230 = 1856 - 16 - 1840 = 0
\end{aligned} \tag{103}$$

We started with the assumption that there exists a mystical method for determining the digits n_k. The convention-al school approach encourages students to guess these numbers. In contrast, Toepler employed a straightforward trial-and-error method, systematically subtracting odd numbers to generate intermediate remainders profiting from the property that the sum of the first n odd numbers equals its square n^2. Reminding:

$$\sum_{j=1}^{n} (2j-1) = n^2 \tag{104}$$

Let

$$\begin{aligned}
p_{k=0} &= 0 \\
\forall k \geq 1,\ p_k &= 2 \sum_{i=0}^{k-1} n_{m-i} 10^{2m-k-i}
\end{aligned} \tag{105}$$

$$\begin{aligned}
r_{k,0} &= R_k \\
r_{k,1} &= r_{k,0} - (p_k + 1 \cdot 10^{2(m-k)}) \\
r_{k,2} &= r_{k,1} - (p_k + 3 \cdot 10^{2(m-k)}) \\
r_{k,3} &= r_{k,2} - (p_k + 5 \cdot 10^{2(m-k)}) \\
&\dots \\
r_{k,j} &= r_{k,j-1} - (p_k + (2j-1) \cdot 10^{2(m-k)}) \\
\text{until } r_{k,j} &< 0 \\
&\implies (n_{m-k} = j-1 \text{ and } R_{k+1} = r_{k,j-1})
\end{aligned} \tag{106}$$

Suppose we have determined n_{m-k}. We may consider the last valid odd number entry at the relevant decimal place:

$$w = (2n_{m-k} - 1)10^{2(m-k)} = 2n_{m-k}10^{2(m-k)} - 10^{2(m-k)} \tag{107}$$

$$\begin{aligned}
p_{k+1} &= 2 \sum_{i=0}^{k+1-1} n_{m-i} 10^{2m-k-1-i} \\
&= 2n_{m-k}10^{2m-2k-1} + 2\sum_{i=0}^{k-1} n_{m-i} 10^{2m-k-i-1} \\
&= 10^{-1}\left(w + 10^{2(m-k)} + p_k\right)
\end{aligned} \tag{108}$$

This means that the following p_{k+1} can be obtained by incrementing the relevant digit of p_k augmented by the n_{m-k}th odd number, and dividing the result by 10. No multiplication by 2 must be applied.

Example:

$$
\begin{aligned}
R_{k=0} &= 54756 \\
r_{0,j=0} &= 54756 \\
r_{0,1} &= 54756-(0+1\cdot 10^{4-0}) = 54756-10000 = 44756 \\
r_{0,2} &= 44756-(0+3\cdot 10^{4-0}) = 44756-\underline{30000} = 14756 \\
r_{0,3} &= 14756-(0+5\cdot 10^{4-0}) = 14756-50000 = -35244 \\
n_2 &= j-1 = 2 \\
R_1 &= r_{0,2} = 14756 \\[1em]
p_1 &= 2n_0 10^{4-k} = \underline{\underline{4000}} \\
r_{1,j=0} &= R_1 = 14756 \\
r_{1,1} &= 14756-(4000+1\cdot 10^{4-2}) = 14756-4100 = 10656 \\
r_{1,2} &= 10656-(4000+3\cdot 10^{2}) = 10756-4300 = 6356 \\
r_{1,3} &= 6356-(4000+5\cdot 10^{2}) = 6356-\underline{4500} = 1856 \\
r_{1,4} &= 1856-(4000+7\cdot 10^{2}) = 1856-4700 = -2844 \\
n_1 &= j-1 = 3 \\
R_2 &= r_{0,3} = 1856 \\[1em]
p_2 &= 2(n_0 10^{4-2}+n_1 10^{4-3}) = 2(200+39) = \underline{\underline{460}} \\
r_{2,0} &= R_2 = 1856 \\
r_{2,1} &= 1856-(460+1\cdot 10^{4-4}) = 1856-461 = 1395 \\
r_{2,2} &= 1395-463 = 932 \\
r_{2,3} &= 932-465 = 467 \\
r_{2,4} &= 467-467 = 0 \\
n_0 &= 4
\end{aligned}
\tag{109}
$$

Observation: Imagine that this algorithm was applied to a mechanical calculator with carriage, keyboard, operation counter and accumulator. At each kth main operation step, negative results of intermediate subtractions must be undone through a neutralizing addition. The operation counter register displays the discovered square root digits $n_m n_{m-1} n_{m-2}...$, which drop from the algorithm digit after digit. Note that the underlined values in example Eq. 109 represent the keyboard entries at the end of the kth operation. As can be seen, a left shift of the carriage will realign the input and accumulator numbers, *de facto* dividing the current keyboard input by 10. If 1 is added at the relevant digit, the keyboard is ready for the next operation. Also note that if the operation result is zero, which happens for perfect squares, the relevant digit of the keyboard can be incremented by 1, displaying $2\sqrt{N}$.

20 Convergence

Because the algorithm delivers successive digits of the square root one by one, it is evident that the convergence is *linear*. This requires no further discussion. The interest of this pseudo-division algorithm lies in the fact that it can be reduced to a shift and add method, involving neither multiplication nor division, which makes it suitable both for hard- and software implementation at elementary level. Because each operation treats two bits of the input argument, the algorithm executes at double speed of a normal paper-and-pencil division.

21 Crenshaw's Variant

Crenshaw (2000) [pp.72-78] presents a variant of the algorithm that was used by NASA engineers on a non-square root FRIDEN calculator during the early GEMINI and APOLLO era. For practical reasons they preferred subtracting $j-1$ and subsequently j at the relevant digits. Summed up, the difference evidently is $2j-1$. The question arises, what could have been the gain of this curious alternative? For the sake of clarity, we will illustrate the variant with an example. Note that the method has been implemented in 1965 in the electronic FRIDEN EC-132 Calculator (patent US3526760 by Ragen (1970)):

$$
\begin{aligned}
r_{k=0,j=0} &= 54756\\
r_{0,1} &= 54756-(0+0\cdot 10^{4-0}) = 54756-00000 = 54756\\
r_{0,2} &= 54756-(0+1\cdot 10^{4-0}) = 54756-10000 = 44756\\
r_{0,3} &= 44756-(0+1\cdot 10^{4-0}) = 44756-10000 = 34756\\
r_{0,4} &= 34756-(0+2\cdot 10^{4-0}) = 34756-20000 = 14756\\
r_{0,5} &= 14756-(0+2\cdot 10^{4-0}) = 14756-20000 = -5244\\
\Longrightarrow\ r_{0,\underline{4}} &= -5244+\underline{20000} = 14756\\
\\
r_{1,1} &= 14756-\underline{\underline{2000}} = 12756\\
r_{1,2} &= 12756-2100 = 10656\\
r_{1,3} &= 10656-2100 = 8556\\
r_{1,4} &= 8556-2200 = 6356\\
r_{1,5} &= 6356-2200 = 4156\\
r_{1,6} &= 4145-2300 = 1856\\
r_{1,7} &= 1856-2300 = -444\\
\Longrightarrow\ r_{1,\underline{6}} &= -444+\underline{2300} = 1856\\
\\
r_{2,1} &= 1856-\underline{\underline{230}} = 1626\\
r_{2,2} &= 1626-231 = 1395\\
r_{2,3} &= 1395-231 = 1164\\
r_{2,4} &= 1164-232 = 932\\
r_{2,5} &= 932-232 = 700\\
r_{2,6} &= 700-233 = 467\\
r_{2,7} &= 467-233 = 234\\
r_{2,\underline{8}} &= 234-\underline{234} = 0
\end{aligned}
\tag{110}
$$

As can be seen in the example Eq. 110, the square root $\sqrt{N}$ now is visible on the entry (=keyboard) side, whereas $2\sqrt{N}$ appears in the successive j values, which on the FRIDEN calculator are displayed in the operation counter register, because the number of operations has doubled. Also, no further keyboard manipulation must be made after the undoing of the negative subtraction, so that a left shift of the carriage is the only thing to do. At new k value, subtractions with the same keyboard number is done once, then 1 is added at the relevant digit; the subtraction key is pressed twice; again increment; two identical subtractions; a.s.o. Finally, the keyboard holds $\sqrt{N}$, from which it can be used for further calculations.

22 FRIDEN and HP-35's Artfulness

Mechanical calculators like the FRIDEN-SWR10 are based on the decimal system, and early digital calculators, for instance the HP-35, often used BCD-architecture, which made Toepler's method attractive.

FRIDEN and HP engineers used the same intriguing method of eliminating the multiplication by 2 from the algorithm. In fact, they scaled the numbers by factor 5 right at the beginning, changing the multiplication by 2 to a multiplication by 10, which represents a simple left shift. Further details can be found in Arithmeum (2023), HP-Museum (2024) and Egbert (1977).

Jarvis AF (2005) and Goldberg (2023) give short-hand description of the applied algorithm that we reproduce here as is:

1. ***Initial step:*** *Let $a = 5n$ (this multiplication by 5 is the only time when an operation other than addition and subtraction is involved!), and put $b = 5$.*
2. ***Repeated steps:***
   - *(R1) If $a \geq b$, replace a with $a - b$, and add 10 to b.*
   - *(R2) If $a < b$, add two zeroes to the end of a, and add a zero to b just before the final digit (which will always be 5).*
3. ***Conclusion:*** *Then the digits of b approach more and more closely the digits of the square root of n.*

The authors associate this algorithm with ancient Japan, where it appears to have been used alongside the abacus. Since no infinite decimals are involved in any of the calculations, there is no loss due to rounding errors. This method is referred to as the SPIGOT ALGORITHM by Goldberg (2023). Let's explore how the algorithm can be applied using Table 9 and compare it with Toepler's method in Figure 28. Notably, multiplying by 5 also resolves the initial alignment issue. It's important to disregard the last digit of the Japanese result. Similar to Crenshaw's variant, the outcome appears in the keyboard entry when the algorithm is implemented in a mechanical calculator.

	a	b
	$= 5 \cdot 2 = 10$	5
$a \geq b \implies$	$a - b = 5$	$b + 10 = 15$
$a < b \implies$	500	105
$a \geq b \implies$	$500 - 105 = 395$	$105 + 10 = 115$
	$395 - 115 = 280$	125
	155	135
	20	145
$a < b \implies$	2000	1405
$a \geq b \implies$	595	1415

Table 9: Computing $\sqrt{2}$ using the Japanese Spigot Algorithm

```
√10                           √2 00 00 00 00  = 1 4 1 4 2
 -5  -->10                    -1
 -----                        -----
  5 00                         1 00
 -1 05                          -21
 -1 15                          -23
 -1 25                          -25
 -1 35  -->1 45                 -27
 --------                     --------
    20 00                         4 00
   -14 05  -->14 15              -2 81
 -----------                  -----------
     5 95 00                      1 19 00
    -1 41 05                       -28 21
    -1 41 15                       -28 23
    -1 41 25                       -28 25
    -1 41 35  -->1 41 45           -28 27
 --------------               --------------
       30 20 00                     6 04 00
      -14 14 05                    -2 82 81
      -14 14 15  -->14 14 25       -2 82 83
```

Figure 28: Japanese Spigot Algorithm vs. Toepler's Method

23 Binary Implementation of Toepler's Algorithm

Toepler's algorithm can be rewritten using base-2 arithmetic. In base-10 there can be maximally 9 odd numbers being used in Eq. 106. If the method is transcribed to base-2, there is only a single odd number per step. This greatly simplifies the algorithm. Also, the product by 2 can be replaced with a simple left shift.

We reproduce here the C-version presented by Warren (2012). Note that this version is more condensed than Crenshaw (2000) [p.86], Rolfe (1987) and Martin (1985) implementations. Observe that the left shift is substituted with a right shift of the corresponding operand. This is feasible because operand alignments are relative to each other:

```c
int isqrt(unsigned x) {
   unsigned m, y, b;

   m = 0x40000000;
   y = 0;
   while(m != 0) {              // Do 16 times.
       b = y | m;
       y = y >> 1;
       if (x >= b) {
          x = x - b;
          y = y | m;
       }
       m = m >> 2;
   }
   return y;
}
```

Listing 2: C implementation of Toepler's Algorithm (32-bit)

```
x =  10  10  10  01
m =   1  00  00  00
y =               0
========================================================================
b = y | m;          1  00  00  00     1  01  00  00     11  01  00     1  10  01
y = y >> 1;                     0        10  00  00      1  10  00        11  00
if (x >= b)  {
    x = x - b;      1  10  10  01        1  10  01      ----------             0
    y = y | m;      1  00  00  00       11  00  00      ----------        11  01
   }
   m = m >> 2;          1  00  00            1  00               1             0
```

Listing 3: 8-bit example in binary notation

24 Floating Point Adaptation of the Binary Toepler Algorithm

The binary Toepler method can be applied to the mantissa of a floating point number. In order to obtain standard precision $p \approx 1E-7$ in the case of single precision floating point representation, the mantissa must be operated as an unsigned 64-bit variable.

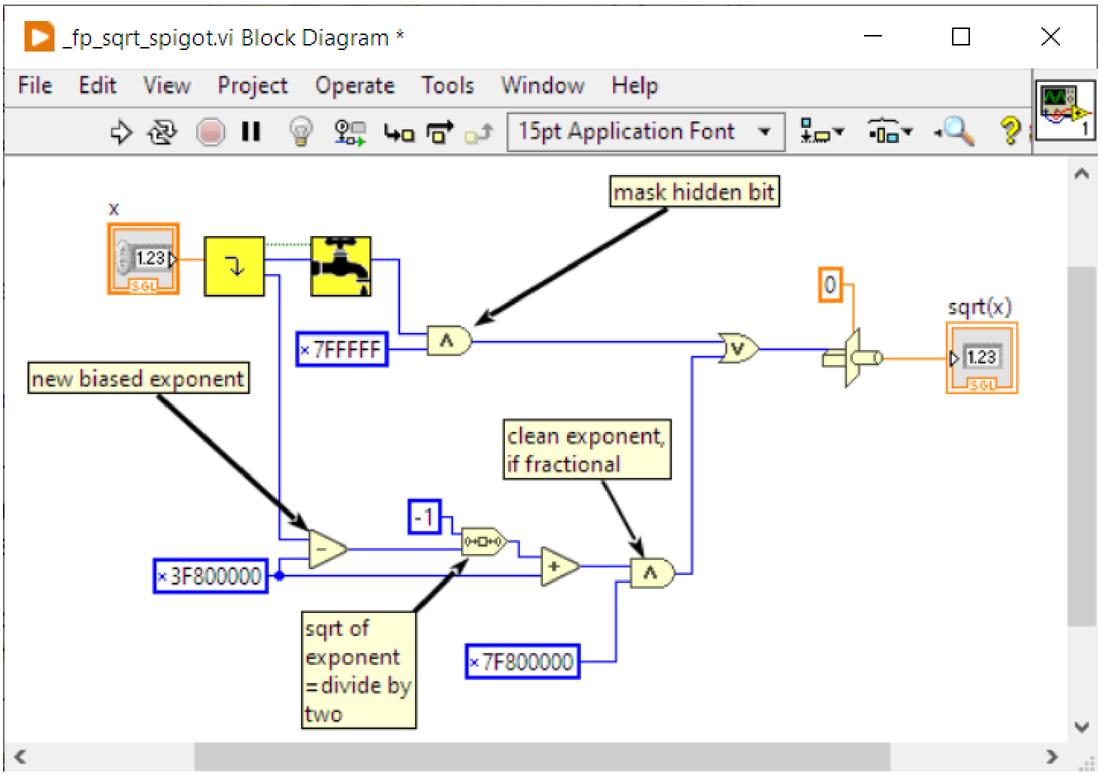
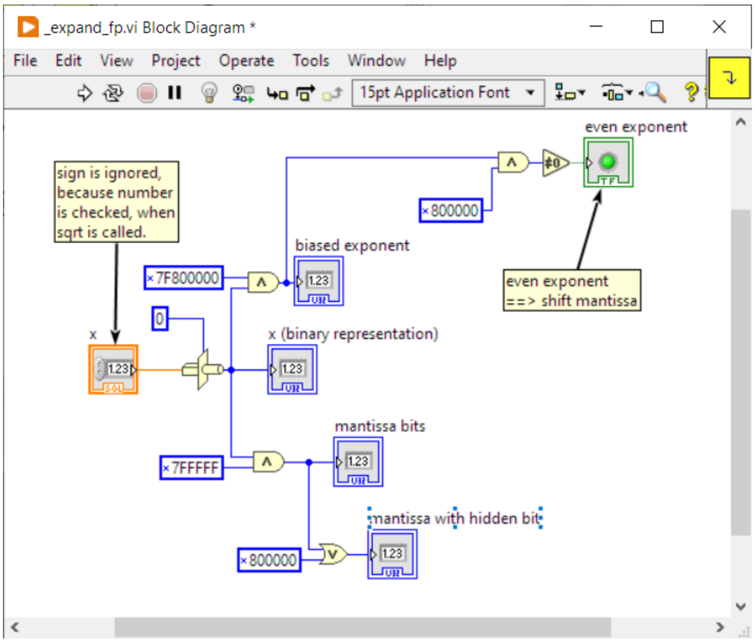


Figure 29: Implementation in LABVIEW of Toepler's Algorithm, applied to single precision floating point values.



sign is ignored, because number is checked, when sqrt is called.

Figure 30: This function expands single precision floating point values.

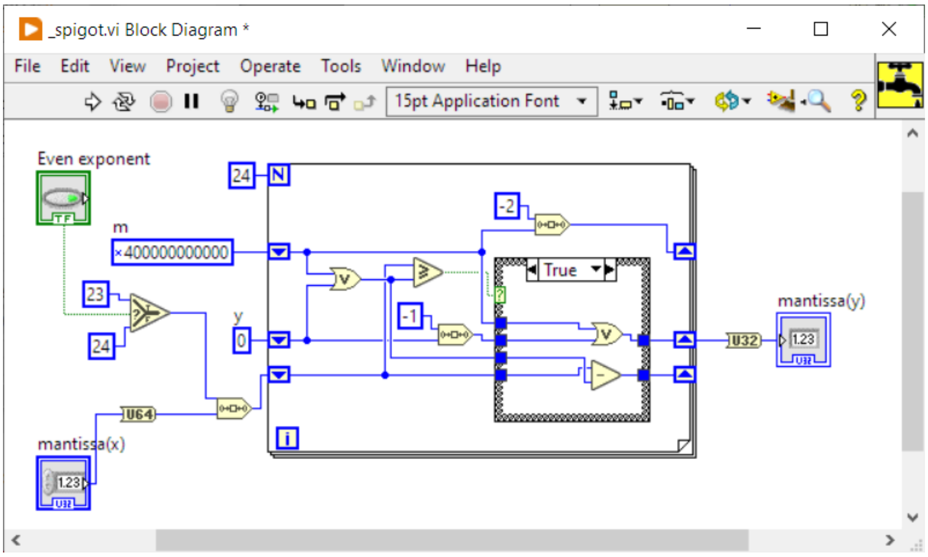


Figure 31: This subroutine calculates the binary Toepler square root on the mantissa.

Fig. 29 shows the main routine calling first the floating point **expand** subroutine (Fig. 30) and the **spigot** subroutine (Fig. 30), then calculating the square root of the unbiased exponent and finally re-normalizing the square root as a floating point. Note that the mantissa must be aligned differently, depending on the exponents oddity.

By contrast to Listing 2, the main loop must run for all of the 24 mantissa bits, in order to guarantee the desired precision of the square root.

Observation: Due to Toepler's algorithm, which rounds to negative infinity at the least significant bit, the result of the floating-point square root may differ by 1 *unit in the last place* (ulp). While improvements are possible, we believe this topic lies outside the scope of this document.

24.1 Conclusion

The binary adaptation of Toepler's Algorithm offers several advantages. Firstly, it avoids both division and multiplication operations, which simplifies the computation process. Additionally, the algorithm's speed is independent of the initial value chosen, eliminating the need for complex and time-consuming initialization. Furthermore, precision remains unaffected by other rounding or error-propagating effects, with the final rounding occurring at the least significant bit. Due to its high execution speed, this algorithm is particularly well-suited for hardware implementation or applications in microcontrollers (MCUs) or central processing units (CPUs) lacking hardware division capabilities, especially in the context of field-programmable gate arrays (FPGAs).

References

Arithmeum (2023). Das Wurzelwerk des Friden SRW Wurzelautomats. `https://www.youtube.com/watch?v=ZqNvABmpgZs`. [retrieved July 2024].

Arnold, Jeff (2017). An introduction to floating-point arithmetic and computation. `https://indico.cern.ch/event/626147/attachments/1456066/2247140/FloatingPoint.Handout.pdf`. retrieved June 2024.

Blanchard, J. D. and Chamberland, M. (2023). Newton's method without division. The American Mathematical Monthly, 130(7):606–617.

Boldo, S., Jeannerod, C.-P., Melquiond, G., and Muller, J.-M. (2023). Floating-point arithmetic. Acta Numerica, 32:203–290.

Catalan, E. (1849). Théorie des fractions continues. Nouvelles annales de mathématiques : journal des candidats aux écoles polytechnique et normale, 1e série, 8:154–202.

Crenshaw, J. W. (2000). Math Toolkit for Real-Time Programming. CMP Books, Lawrence, KS.

Dianov, A. and Anuchin, A. (2020a). Fast square root calculation for control systems of power electronics. 2020 23rd International Conference on Electrical Machines and Systems (ICEMS), pages 438–443.

Dianov, A. and Anuchin, A. (2020b). Review of fast square root calculation methods for fixed point microcontroller-based control systems of power electronics. International Journal of Power Electronics and Drive System (IJPEDS), 11(3):1153 1164.

Egbert, W. E. (1977). Personal calculator algorithms. I. square roots. 28(9):22–23.

Flores, A. (2014). The Babylonian method for approximating square roots: Why is it so efficient? The Mathematics Teacher, 108.

Ford, G. A. (2005). Graphical analysis of orbits, fixed points and functions. `https://ocw.mit.edu/courses/18-091-mathematical-exposition-spring-2005/pages/lecture-notes/`.

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv., 23(1):5–48. corrigendum: ACM Computing Surveys 23(3): 413 (1991), comments: ACM Computing Surveys 24(2): 319 (1992).

Goldberg, M. (2023). A spigot-algorithm for square-roots: Explained and extended. CoRR, abs/2312.15338.

Goldschmidt, R. E. (1964). Applications of division by convergence.

Gärtner, M. (2008). Analyse des Toepler-Algorithmus zum Berechnen von Quadratwurzeln mit mechanischen Vierspezies-Rechenmaschinen. https://rechnerlexikon.de/files/toepler1.pdf. [retrieved June 2024].

HP-Museum (2024). HP-Museum. https://www.hpmuseum.org/root.htm. retrieved June 2024.

Jarvis AF (2005). Mathematical Spectrum, 37:119–122. Accessed on 2024/06/30.

Khinchin, A. (1964). Continued Fractions. Phoenix books. University of Chicago Press.

Knuth, D. E. (1972). Ancient Babylonian algorithms. Commun. ACM, 15(7):671–677.

Lischka, C., Arndt, J., Lischka, D., and Haenel, C. (2012). Pi - Unleashed. Springer Berlin Heidelberg.

Martin, G. (1985). Square root by abacus algorithm. https://web.archive.org/web/20120306040058/http://medialab.freaknet.org/martin/src/sqrt/sqrt.c. [retrieved June 2024].

Ming, K. and Woo, C. (2012). The Fundamental Operations In Bead Arithmetic: How To Use The Chinese Abacus. Literary Licensing, LLC.

Montuschi, P. and Mezzalama, M. (1990). Survey of square rooting algorithms. IEE Proceedings E (Computers and Digital Techniques), 137:31–40(9).

Olds, C. (1963). Continued Fractions. Number Bd. 9 in Anneli Lax New Mathematical Library. Mathematical Association of America.

Ragen, R. A. (U.S. Patent 2 526 760, Sep. 1970). Square root calculator employing a modified method sum of the odd integers method.

Releaux, F. (1866). Prof. Toepler's Verfahren der Wurzelziehung mittelst der Thomas'schen Rechenmaschine. Polytechnisches Journal, 179:261–264.

Rolfe, T. J. (1987). On a fast integer square root algorithm. SIGNUM Newsl., 22(4):6–11.

Steele, G. L. (1977). Arithmetic shifting considered harmful. SIGPLAN Not., 12(11):61–69.

von Neumann, J. (1993). First draft of a report on the edvac. IEEE Annals of the History of Computing, 15(4):27–75.

Wagstaff Jr., S. S. (2013). The Joy of Factoring. American Mathematical Society.

Warren, H. S. (2012). Hacker's Delight. Addison-Wesley Professional, USA, 2nd edition.

Widz, J. (2009). From the history of continued fractions. WDS'09, Proceeding of Contributed Papers Part I, Prague, 2-5 June 2009, pages 176–181.

wikipédia (2024). Extraction de racine carrée par la méthode du goutte à goutte. https://fr.wikipedia.org/wiki/Extraction_de_racine_carr%C3%A9e_par_la_m%C3%A9thode_du_goutte_%C3%A0_goutte. [retrieved June 2024].

Ypma, T. J. (1995). Historical development of the Newton-Raphson method. SIAM Rev., 37:531–551.