A Modification of the Fast Inverse Square Root Algorithm

Cezary J. Walczyk 1, Leonid V. Moroz 2 and Jan L. Cieśliński 1,*

1 Wydział Fizyki, Uniwersytet w Białymstoku, ul. Ciołkowskiego 1L, 15-245 Białystok, Poland
2 Department of Security Information and Technology, Lviv Polytechnic National University, st. Kn. Romana 1/3, 79000 Lviv, Ukraine
* Correspondence: j.cieslinski@uwb.edu.pl

Received: 28 June 2019; Accepted: 14 August 2019; Published: 18 August 2019

Abstract: We present a new algorithm for the approximate evaluation of the inverse square root for single-precision floating-point numbers. This is a modification of the famous fast inverse square root code. We use the same “magic constant” to compute the seed solution, but then, we apply Newton–Raphson corrections with modified coefficients. As compared to the original fast inverse square root code, the new algorithm is two-times more accurate in the case of one Newton–Raphson correction and almost seven-times more accurate in the case of two corrections. We discuss relative errors within our analytical approach and perform numerical tests of our algorithm for all numbers of the type float.

Keywords: floating-point arithmetic; inverse square root; magic constant; Newton–Raphson method

1. Introduction

Floating-point arithmetic has become widely used in many applications such as 3D graphics, scientific computing and signal processing [1–5], implemented both in hardware and software [6–10]. Many algorithms can be used to approximate elementary functions [1, 2, 10–18]. The inverse square root function \( (x \rightarrow 1/\sqrt{x}) \) is of particular importance because it is widely used in 3D computer graphics, especially in lightning reflections [19–21], and has many other applications; see [22–36]. All of these algorithms require an initial seed to start the approximation. The more accurate is the initial seed, the fewer iterations are needed. Usually, the initial seed is obtained from a look-up table (LUT), which is memory consuming.

In this paper, we consider an algorithm for computing the inverse square root using the so-called magic constant instead of an LUT [37–40]. The zeroth approximation (initial seed) for the inverse square root of a given floating-point number is obtained by a logical right shift by one bit and subtracting this result from an specially-chosen integer (“magic constant”). Both operations are performed on bits of the floating-point number interpreted as an integer. Then, a more accurate value is produced by a certain number (usually one or two) of standard Newton–Raphson iterations. The following code realizes the fast inverse square root algorithm in the case of single-precision IEEE Standard 754 floating-point numbers (type float).

The code \texttt{InvSqrt} (see Algorithm 1) consists of two main parts. Lines 4 and 5 produce in a very inexpensive way a quite good zeroth approximation of the inverse square root of a given positive floating-point number \( x \). Lines 6 and 7 apply the Newton–Raphson corrections twice (often, a version with just one iteration is used, as well). Originally, \( R \) was proposed as 0x5F3759DF; see [37, 38]. More details, together with a derivation of a better magic constant, are given in Section 2.
Algorithm 1: InvSqrt.

1. float InvSqrt(float x)
2.   float halfnumber = 0.5f*x;
3.   int i = *(int*) &x;
4.   i = R - (i >> 1);
5.   y = *(float*) &i;
6.   y = y*(1.5f - halfnumber*y*y);
7.   y = y*(1.5f - halfnumber*y*y);
8.   return y ;
9. }

InvSqrt is characterized by a high speed, more that three-times higher than computing the inverse square root using library functions. This property was discussed in detail in [41]. The errors of the fast inverse square root algorithm depend on the choice of the “magic constant” $R$. In several theoretical papers [38,41–44] (see also Eberly’s monograph [19]), attempts were made to determine analytically the optimal value of the magic constant (i.e., to minimize errors). In general, this optimal value can depend on the number of iterations, which is a general phenomenon [45]. The derivation and comprehensive mathematical description of all the steps of the fast inverse square root algorithm were given in our recent paper [46]. We found the optimum value of the magic constant by minimizing the final maximum relative error.

In the present paper, we develop our analytical approach to construct an improved algorithm (InvSqrt1) for fast computing of the inverse square root; see Algorithm 2 in Section 4. The proposed modification does not increase the speed of data processing, but increases, in a significant way, the accuracy of the output. In both codes, InvSqrt and InvSqrt1, magic constants serve as a low-cost way of generating a reasonably accurate first approximation of the inverse square root. These magic constants turn out to be the same. The main novelty of the new algorithm is in the second part of the code, which is changed significantly. In fact, we propose a modification of the Newton–Raphson formulae, which has a similar computational cost, but improve the accuracy by several fold.

2. Analytical Approach to the Algorithm InvSqrt

In this paper, we confine ourselves to positive single-precision floating-point numbers (type float). Normal floating-point numbers can be represented as:

$$x = (1 + m_x)2^{e_x}$$

where $m_x \in [0, 1)$ and $e_x$ is an integer (note that this formula does not hold for subnormal numbers). In the case of the IEEE-754 standard, a floating-point number is encoded by 32 bits. The first bit corresponds to a sign (in our case, this bit is simply equal to zero); the next eight bits correspond to an exponent $e_x$; and the last 23 bits encode a mantissa $m_x$. The integer encoded by these 32 bits, denoted by $I_x$, is given by:

$$I_x = N_m (B + e_x + m_x)$$

where $N_m = 2^{23}$ and $B = 127$ (thus $B + e_x = 1, 2, \ldots, 254$). Lines 3 and 5 of the InvSqrt code interpret a number as an integer (2) or float (1), respectively. Lines 4, 6, and 7 of the code can be written as:

$$I_{y_0} = R - \lfloor I_x/2 \rfloor, \quad y_1 = \frac{1}{2}y_0(3 - y_0^2x), \quad y_2 = \frac{1}{2}y_1(3 - y_1^2x).$$

The first equation produces, in a surprisingly simple way, a good zeroth approximation $y_0$ of the inverse square root $y = 1/\sqrt{x}$. Of course, this needs a very special form of $R$. In particular, in the single precision case, we have $e_R = 63$; see [46]. The next equations can be easily recognized as the
Newton–Raphson corrections. We point out that the code \textit{InvSqrt} is invariant with respect to the scaling:

\[ x \rightarrow \tilde{x} = 2^{-2n}x, \quad y_k \rightarrow \tilde{y}_k = 2^n y_k \quad (k = 0, 1, 2), \] (4)

like the equality \( y = 1/\sqrt{x} \) itself. Therefore, without loss of the generality, we can confine our analysis to the interval:

\[ \tilde{A} := [1, 4). \] (5)

The tilde will denote quantities defined on this interval. In [46], we showed that the function \( \tilde{y}_0 \) defined by the first equation of (3) can be approximated with a very good accuracy by the piece-wise linear function \( \tilde{y}_{00} \) given by:

\[
\tilde{y}_{00}(\tilde{x}, t) = \begin{cases} 
-\frac{1}{4} \tilde{x} + \frac{3}{2} \frac{1}{t} & \text{for } \tilde{x} \in [1, 2) \\
-\frac{1}{8} \tilde{x} + \frac{1}{2} + \frac{1}{8} t & \text{for } \tilde{x} \in [2, t) \\
-\frac{1}{16} \tilde{x} + \frac{1}{2} + \frac{1}{16} t & \text{for } \tilde{x} \in [t, 4)
\end{cases}
\] (6)

where:

\[ t = 2 + 4m_R + 2N_m^{-1}, \] (7)

and \( m_R := N_m^{-1}R - \lfloor N_m^{-1}R \rfloor \) (\( m_R \) is the mantissa of the floating-point number corresponding to \( R \)). Note that the parameter \( t \), defined by (7), is uniquely determined by \( R \).

The only difference between \( y_0 \) produced by the code \textit{InvSqrt} and \( y_{00} \) given by (6) is the definition of \( t \), because \( t \) related to the code depends (although in a negligible way) on \( x \). Namely,

\[ |y_{00} - y_0| \leq \frac{1}{4} N_m^{-1} = 2^{-25} \approx 2.98 \times 10^{-8}. \] (8)

Taking into account the invariance (4), we obtain:

\[ \left| \frac{y_{00} - y_0}{y_0} \right| \leq 2^{-24} \approx 5.96 \times 10^{-8}. \] (9)

These estimates do not depend on \( t \) (in other words, they do not depend on \( R \)). The relative error of the zeroth approximation (6) is given by:

\[ \tilde{\delta}_0(\tilde{x}, t) = \sqrt{\tilde{x}} \tilde{y}_{00}(\tilde{x}, t) - 1 \] (10)

This is a continuous function with local maxima at:

\[ \tilde{x}_0^I = (6 + t) / 6, \quad \tilde{x}_0^{II} = (4 + t) / 3, \quad \tilde{x}_0^{III} = (8 + t) / 3, \] (11)

given respectively by:

\[
\tilde{\delta}_0(\tilde{x}_0^I, t) = -1 + \frac{1}{2} \left( 1 + \frac{t}{6} \right)^{3/2}, \\
\tilde{\delta}_0(\tilde{x}_0^{II}, t) = -1 + 2 \left( \frac{1}{3} \left( 1 + \frac{t}{4} \right) \right)^{3/2}, \\
\tilde{\delta}_0(\tilde{x}_0^{III}, t) = -1 + \left( \frac{2}{3} \left( 1 + \frac{t}{8} \right) \right)^{3/2}.
\] (12)
In order to study the global extrema of $\delta_0(\bar{x},t)$, we need also boundary values:

$$
\delta_0(1, t) = \delta_0(4, t) = \frac{1}{8} (t - 4), \quad \delta_0(2, t) = \frac{\sqrt{t}}{4} \left(1 + \frac{t}{2}\right) - 1, \quad \delta_0(t, t) = \frac{\sqrt{t}}{2} - 1,
$$

which are, in fact, local minima. Taking into account:

$$
\delta_0(1, t) - \delta_0(t, t) = \frac{1}{8} \left(\sqrt{t} - 2\right)^2 \geq 0, \quad \delta_0(2, t) - \delta_0(t, t) = \frac{\sqrt{2}}{8} \left(\sqrt{t} - \sqrt{2}\right)^2 \geq 0,
$$

we conclude that:

$$
\min_{x \in A} \delta_0(\bar{x}, t) = \delta_0(t, t) < 0. \quad (15)
$$

Because $\delta_0(\bar{x}^{|II}, t) < 0$ for $t \in (2, 4)$, the global maximum is one of the remaining local maxima:

$$
\max_{x \in A} \delta_0(\bar{x}, t) = \max \{\delta_0(\bar{x}^I, t), \delta_0(\bar{x}^{|II}, t)\}. \quad (16)
$$

Therefore,

$$
\max_{x \in A} |\delta_0(\bar{x}, t)| = \max \{|\delta_0(t, t)|, |\delta_0(\bar{x}^I, t), |\delta_0(\bar{x}^{|II}, t)|. \quad (17)
$$

In order to minimize this value with respect to $t$, i.e., to find $t^*_0$ such that:

$$
\max_{x \in A} |\delta_0(\bar{x}, t)| < \min_{x \in A} |\delta_0(\bar{x}, t)| \quad \text{for} \quad t \neq t^*_0, \quad (18)
$$

we observe that $|\delta_0(t, t)|$ is a decreasing function of $t$, while both maxima ($\delta_0(\bar{x}^I, t)$ and $\delta_0(\bar{x}^{|II}, t)$) are increasing functions. Therefore, it is sufficient to find $t = t^I_0$ and $t = t^{|II}_0$ such that:

$$
|\delta_0(t^I_0, t^I_0)| = \delta_0(\bar{x}^I, t^I_0), \quad |\delta_0(t^{|II}_0, t^{|II}_0)| = \delta_0(\bar{x}^{|II}, t^{|II}_0),
$$

and to choose the greater of these two values. In [46], we showed that:

$$
|\delta_0(t^I_0, t^I_0)| < |\delta_0(t^{|II}_0, t^{|II}_0)|. \quad (20)
$$

Therefore, $t^*_0 = t^{|II}_0$, and:

$$
\delta_{0, max} := \min_{t \in (2, 4)} \left(\max_{x \in A} |\delta_0(\bar{x}, t)|\right) = |\delta_0(t^*_0, t^*_0)|. \quad (21)
$$

The following numerical values result from these calculations [46]:

$$
t^*_0 \approx 3.7309796, \quad R_0 = 0x5F37642F, \quad \delta_{0, max} \approx 0.03421281. \quad (22)
$$

Newton–Raphson corrections for the zeroth approximation ($y_{00}$) will be denoted by $\tilde{y}_{0k}$ ($k = 1, 2, \ldots$). In particular, we have:

$$
\tilde{y}_{01}(\bar{x}, t) = \frac{1}{2} y_{00}(\bar{x}, t)(3 - y_{00}^2(\bar{x}, t) \bar{x}),
$$

$$
\tilde{y}_{02}(\bar{x}, t) = \frac{1}{2} y_{01}(\bar{x}, t)(3 - y_{01}^2(\bar{x}, t) \bar{x}). \quad (23)
$$

and the corresponding relative error functions will be denoted by $\delta_k(\bar{x}, t)$:

$$
\tilde{\delta}_k(\bar{x}, t) := \frac{\tilde{y}_{0k}(\bar{x}, t) - \bar{y}}{\bar{y}} = \sqrt{\tilde{y}} \tilde{y}_{0k}(\bar{x}, t) - 1, \quad (k = 0, 1, 2, \ldots), \quad (24)
$$

where we included also the case $k = 0$; see (10). The obtained approximations of the inverse square root depend on the parameter $t$ directly related to the magic constant $R$. The value of this parameter can be
estimated by analyzing the relative error of \( \hat{y}_{0k}(\hat{x}, t) \) with respect to \( 1/\sqrt{x} \). As the best estimation, we consider \( t = t_k^{(r)} \) minimizing the relative error \( \hat{\delta}_k(\hat{x}, t) \):

\[
\forall t \neq t_k^{(r)} \left( \hat{\delta}_{k, \max} \equiv \max_{\hat{x} \in A} |\hat{\delta}_k(\hat{x}, t_k^{(r)})| < \max_{\hat{x} \in A} |\hat{\delta}_k(\hat{x}, t)| \right). \tag{25}
\]

We point out that in general, the optimum value of the magic constant can depend on the number of Newton–Raphson corrections. Calculations carried out in [46] gave the following results:

\[
t'_1 = t'_2 = 3.7298003, \quad R'_1 = R'_2 = 0x5F375A86, \quad \delta_{1, \max} \approx 1.75118 \times 10^{-3}, \quad \delta_{2, \max} \approx 4.60 \times 10^{-6}. \tag{26}
\]

We omit the details of the computations except one important point. Using (24) to express \( \hat{y}_{0k} \) by \( \hat{\delta}_k \) and \( \sqrt{x} \), we can rewrite (23) as:

\[
\hat{\delta}_k(\hat{x}, t) = -\frac{1}{2} \frac{d_1}{\sqrt{A}} (\hat{x}, t) (3 + \hat{\delta}_{k-1}(\hat{x}, t)), \quad (k = 1, 2, \ldots). \tag{27}
\]

The quadratic dependence on \( \hat{\delta}_{k-1} \) means that every Newton–Raphson correction improves the accuracy by several orders of magnitude (until the machine precision is reached); compare (26).

The Formula (27) suggests another way of improving the accuracy because the functions \( \hat{\delta}_k \) are always non-positive for any \( k \geq 1 \). Roughly speaking, we are going to shift the graph of \( \hat{\delta}_k \) upwards by an appropriate modification of the Newton–Raphson formula. In the next section, we describe the general idea of this modification.

3. Modified Newton–Raphson Formulas

The Formula (27) shows that errors introduced by Newton–Raphson corrections are nonpositive, i.e., they take values in intervals \([-\hat{\delta}_{k, \max}, 0]\), where \( k = 1, 2, \ldots \). Therefore, it is natural to introduce a correction term into the Newton–Raphson formulas (23). We expect that the corrections will be roughly half of the maximal relative error. Instead of the maximal error, we introduce two parameters, \( d_1 \) and \( d_2 \). Thus, we get modified Newton–Raphson formulas:

\[
\hat{y}_{11}(\hat{x}, t, d_1) = 2^{-1} \hat{y}_{00}(\hat{x}, t) (3 - \hat{y}_{00}(\hat{x}, t) \hat{x}) + \frac{d_1}{2\sqrt{A}}, \\
\hat{y}_{12}(\hat{x}, t, d_1, d_2) = 2^{-1} \hat{y}_{11}(\hat{x}, t, d_1) (3 - \hat{y}_{11}(\hat{x}, t, d_1) \hat{x}) + \frac{d_2}{2\sqrt{A}}, \tag{28}
\]

where zeroth approximation is assumed in the form (6). In the following section, the term \( 1/\sqrt{x} \) will be replaced by some approximations of \( \hat{y} \), transforming (28) into a computer code. In order to estimate a possible gain in accuracy, in this section, we temporarily assume that \( \hat{y} \) is the exact value of the inverse square root. The corresponding error functions,

\[
\hat{\delta}_{k, \max}^{(r)}(\hat{x}, t, d_1, \ldots, d_k) = \sqrt{x} \hat{y}_{1k}(\hat{x}, t, d_1, \ldots, d_k) - 1, \quad k \in \{0, 1, 2, \ldots\}, \tag{29}
\]

(where \( \hat{y}_{10}(\hat{x}, t) := \hat{y}_{00}(\hat{x}, t) \)), satisfy:

\[
\hat{\delta}_{k, \max}^{(r)} = -\frac{1}{2} \hat{\delta}_{k-1}^{(r)} (3 + \hat{\delta}_{k-1}^{(r)}) + \frac{d_k}{2}, \tag{30}
\]

where: \( \hat{\delta}_0^{(r)}(\hat{x}, t) = \hat{\delta}_0(\hat{x}, t) \). Note that:

\[
\hat{\delta}_1^{(r)}(\hat{x}, t, d_1) = \hat{\delta}_1(\hat{x}, t) + \frac{1}{2} d_1. \tag{31}
\]
In order to simplify notation, we usually will suppress the explicit dependence on \( d_i \). We will write, for instance, \( \delta_2'(\hat{x}, t) \) instead of \( \delta_2'(\hat{x}, t, d_1, d_2) \).

The corrections of the form (28) will decrease relative errors in comparison with the results of earlier papers [38,46]. We have three free parameters \((d_1, d_2, \text{ and } t)\) to be determined by minimizing the maximal error (in principle, the new parameterization can give a new estimation of the parameter \( t \)). By analogy to (25), we are going to find \( t = t(0) \) minimizing the error of the first correction (25):

\[
\forall t \neq t(0) \max_{\hat{x} \in \hat{A}} |\delta_1''(\hat{x}, t(0))| < \max_{\hat{x} \in \hat{A}} |\delta_1'(\hat{x}, t)|,
\]

where, as usual, \( \hat{A} = [1, 4] \).

The first of Equation (30) implies that for any \( t \), the maximal value of \( \delta_1''(\hat{x}, t) \) equals \( \frac{1}{2}d_1 \) and is attained at zeros of \( \delta_0''(\hat{x}, t) \). Using the results of Section 2, including (15), (16), (20), and (21), we conclude that the minimum value of \( \delta_1''(\hat{x}, t) \) is attained either for \( \hat{x} = t \) or for \( \hat{x} = \hat{x}_0^{ll} \) (where there is the second maximum of \( \delta_0''(\hat{x}, t) \)), i.e.,

\[
\min_{\hat{x} \in \hat{A}} \delta_1''(\hat{x}, t) = \min \left\{ \delta_1''(t, t), \delta_1''(\hat{x}_0^{ll}, t) \right\}
\]

Minimization of \( |\delta_1''(\hat{x}, t)| \) can be done with respect to \( t \) and with respect to \( d_1 \) (these operations obviously commute), which corresponds to:

\[
\max_{\hat{x} \in \hat{A}} \delta_1''(\hat{x}, t(0)) = -\min_{\hat{x} \in \hat{A}} \delta_1''(\hat{x}, t(0)).
\]

Taking into account:

\[
\max_{\hat{x} \in \hat{A}} \delta_1''(\hat{x}, t(0)) = \frac{d_1}{2}, \quad \min_{\hat{x} \in \hat{A}} \delta_1''(\hat{x}, t(0)) = \delta_1''(t(0), t(0)) = -\delta_{1\text{max}} + \frac{d_1}{2},
\]

we get from (34):

\[
\delta_{1\text{max}} = \frac{1}{2}d_1 = \frac{1}{2}\delta_{1\text{max}} \simeq 8.7559 \times 10^{-4},
\]

where:

\[
\delta_{1\text{max}} := \min_{t \in (24)} \left( \max_{\hat{x} \in \hat{A}} |\delta_1'(\hat{x}, t)| \right).
\]

and the numerical value of \( \delta_{1\text{max}} \) is given by (26). These conditions are satisfied for:

\[
t(0) = t(1) \simeq 3.7298003.
\]

In order to minimize the relative error of the second correction, we use equation analogous to (34):

\[
\max_{\hat{x} \in \hat{A}} \delta_2''(\hat{x}, t(0)) = -\min_{\hat{x} \in \hat{A}} \delta_2''(\hat{x}, t(0)),
\]

where from (30), we have:

\[
\max_{\hat{x} \in \hat{A}} \delta_2''(\hat{x}, t(0)) = \frac{d_2}{2}, \quad \min_{\hat{x} \in \hat{A}} \delta_2''(\hat{x}, t(0)) = -\frac{1}{2}\delta_{1\text{max}} \left( 3 + \delta_{1\text{max}} \right) + \frac{d_2}{2}.
\]

Hence:

\[
\delta_{2\text{max}} = \frac{1}{4}\delta_{1\text{max}} \left( 3 + \delta_{1\text{max}} \right).
\]
Expressing this result in terms of formerly computed $\delta_1^\text{max}$ and $\delta_2^\text{max}$, we obtain:

$$
\delta_2^\prime \max = \frac{1}{8} \delta_2^\max + \frac{3}{32} \delta_1^\max \simeq 5.75164 \times 10^{-7} \simeq \frac{\delta_2^\max}{7.99},
$$

where:

$$
\delta_2^\max = \frac{1}{2} \delta_1^\max (3 - \delta_1^\max).
$$

Therefore, the above modification of Newton–Raphson formulas decreases the relative error two times after one iteration and almost eight times after two iterations as compared to the standard InvSqrt algorithm.

In order to implement this idea in the form of a computer code, we have to replace the unknown $1/\sqrt{t}$ (i.e., $\tilde{y}$) on the right-hand sides of (28) by some numerical approximations.

### 4. New Algorithm with Higher Accuracy

Approximating $1/\sqrt{t}$ in Formulas (28) by values on the left-hand sides, we transform (28) into:

$$
\tilde{y}_{21} = \frac{1}{2} \tilde{y}_{20} (3 - \tilde{y}_{20}^2) + \frac{1}{2} d_1 \tilde{y}_{21}, \\
\tilde{y}_{22} = \frac{1}{2} \tilde{y}_{21} (3 - \tilde{y}_{21}^2) + \frac{1}{2} d_2 \tilde{y}_{22},
$$

where $\tilde{y}_{2k}$ ($k = 1, 2, \ldots$) depend on $\tilde{x}$, $t$ and $d_j$ (for $1 \leq j \leq k$). We assume $\tilde{y}_{20} = \tilde{y}_{00}$, i.e., the zeroth approximation is still given by (6). We can see that $\tilde{y}_{21}$ and $\tilde{y}_{22}$ can be explicitly expressed by $\tilde{y}_{20}$ and $\tilde{y}_{21}$, respectively.

Parameters $d_1$ and $d_2$ have to be determined by minimization of the maximum error. We define error functions in the usual way:

$$
\Delta_{k}^{(1)} = \frac{\tilde{y}_{2k} - \tilde{y}}{\tilde{y}} = \sqrt{t} \tilde{y}_{2k} - 1.
$$

Substituting (44) into (43), we get:

$$
\Delta_{1}^{(1)}(\tilde{x}, t, d_1) = \frac{d_1}{2 - d_1} - \frac{1}{2 - d_1} \delta_0^2 (\tilde{x}, t) (3 + \delta_0 (\tilde{x}, t)) = \frac{d_1 + 2 \delta_1 (\tilde{x}, t)}{2 - d_1},
$$

$$
\Delta_{2}^{(1)}(\tilde{x}, t, d_2) = \frac{d_2}{2 - d_2} - \frac{1}{2 - d_2} \left( \Delta_{1}^{(1)}(\tilde{x}, t, d_1) \right)^2 \left( 3 + \Delta_{1}^{(1)}(\tilde{x}, t, d_1) \right).
$$

The equation (45) expresses $\Delta_{1}^{(1)}(\tilde{x}, t, d_1)$ as a linear function of the nonpositive function $\delta_1(\tilde{x}, t)$ with coefficients depending on the parameter $d_1$. The optimum parameters $t$ and $d_1$ will be estimated by the procedure described in Section 3. First, we minimize the amplitude of the relative error function, i.e., we find $t^{(1)}$ such that:

$$
\max_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t^{(1)}) - \min_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t^{(1)}) \leq \max_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t) - \min_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t)
$$

for all $t \neq t^{(1)}$. Second, we determine $d_1^{(1)}$ such that:

$$
\max_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)}) = - \min_{\tilde{x} \in A} \Delta_{1}^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)}).
$$

Thus, we have:

$$
\max_{\tilde{x} \in A} |\Delta_{1}^{(1)}(\tilde{x}, t^{(1)}, d_1^{(1)})| \leq \max_{\tilde{x} \in A} |\Delta_{1}^{(1)}(\tilde{x}, t, d_1)|
$$
for all $d_1$ and $t \in (2, 4)$. $\Delta_1^{(1)}(\tilde{x}, t)$ is an increasing function of $\delta_1(\tilde{x}, t)$; hence:

$$-\frac{d_1^{(1)} - 2 \max_{x \in A} |\delta_1(\tilde{x}, t_1^{(1)})|}{2 - d_1^{(1)}} = \frac{d_1^{(1)}}{2 - d_1^{(1)}},$$  \hspace{1cm} (50)

which is satisfied for:

$$d_1^{(1)} = \max_{x \in A} |\delta_1(\tilde{x}, t_1^{(1)})| = \delta_{1,\text{max}}.$$  \hspace{1cm} (51)

Thus, we can find the maximum error of the first correction $\Delta_1^{(1)}(\tilde{x}, t_1^{(1)})$ (presented in Figure 1):

$$\max_{x \in A} |\Delta_1^{(1)}(\tilde{x}, t_1^{(1)})| = \frac{\max_{x \in A} |\delta_1(\tilde{x}, t_1^{(1)})|}{2 - \max_{x \in A} |\delta_1(\tilde{x}, t_1^{(1)})|},$$  \hspace{1cm} (52)

which assumes the minimum value for $t_1^{(1)} = t_1^{(r)}$:

$$\Delta_1^{(1)} = \frac{\max_{x \in A} |\delta_1(\tilde{x}, t_1^{(r)})|}{2 - \max_{x \in A} |\delta_1(\tilde{x}, t_1^{(r)})|} = \frac{\delta_{1,\text{max}}}{2 - \delta_{1,\text{max}}} \simeq 8.7636 \times 10^{-4} \simeq \frac{\delta_{1,\text{max}}}{2.00}. \hspace{1cm} (53)$$

This result practically coincides with $\delta_{1,\text{max}}^{(r)}$ given by (36).

Analogously, we can determine the value of $d_2^{(1)}$ (assuming that $t = t_1^{(1)}$ is fixed):

$$-\frac{d_2^{(1)} - \max_{x \in A} |\Delta_1^{(1)}(\tilde{x}, t_1^{(1)})(3 + \Delta_1^{(1)}(\tilde{x}, t_1^{(1)}))|}{2 - d_2^{(1)}} = \frac{d_2^{(1)}}{2 - d_2^{(1)}},$$  \hspace{1cm} (54)

Now, the deepest minimum comes from the global maximum:

$$\max_{x \in A} |\Delta_1^{(1)}(\tilde{x}, t_1^{(1)})(3 + \Delta_1^{(1)}(\tilde{x}, t_1^{(1)}))| = \frac{2\delta_{1,\text{max}}^{(3)}(3 - \delta_{1,\text{max}})}{(2 - \delta_{1,\text{max}})^3}. \hspace{1cm} (55)$$

Therefore, we get:

$$d_2^{(1)} = \frac{\delta_{1,\text{max}}^{(3)}(3 - \delta_{1,\text{max}})}{(2 - \delta_{1,\text{max}})^3} \simeq 1.15234 \times 10^{-6}, $$  \hspace{1cm} (56)

and the maximum error of the second correction is given by:

$$\Delta_2^{(1)} = \frac{d_2^{(1)}}{2 - d_2^{(1)}} \simeq 5.76173 \times 10^{-7} \simeq \frac{\delta_{2,\text{max}}^{(7,98)}}{7.98}, \hspace{1cm} (57)$$

which is very close to the value of $\delta_{2,\text{max}}^{(r)}$ given by (42).
Thus, we have obtained the algorithm \texttt{InvSqrt1}, see Algorithm 2, which looks like \texttt{InvSqrt} with modified values of the numerical coefficients.

\begin{algorithm}
\caption{\texttt{InvSqrt1}.}
1. \texttt{float InvSqrt1(float x)\{}
2. \hspace{1em} \texttt{float simhalfnumber = 0.500438180f*x;}
3. \hspace{1em} \texttt{int i = *(int*) \&x;}
4. \hspace{1em} \texttt{i = 0x5F375A86 - (i>>1);}
5. \hspace{1em} \texttt{y = *(float*) \&i;}
6. \hspace{1em} \texttt{y = y*(1.50131454f - simhalfnumber*y*y);}
7. \hspace{1em} \texttt{y = y*(1.50000086f - 0.999124984f*simhalfnumber*y*y);}
8. \hspace{1em} \texttt{return y ;}
9. \hspace{1em} \}\}
\end{algorithm}

Comparing \texttt{InvSqrt1} with \texttt{InvSqrt}, we easily see that the number of algebraic operations in \texttt{InvSqrt1} is greater by one (an additional multiplication in Line 7, corresponding to the second iteration of the modified Newton–Raphson procedure). We point out that the magic constants for \texttt{InvSqrt} and \texttt{InvSqrt1} are the same.

5. Numerical Experiments

The new algorithm \texttt{InvSqrt1} was tested on the processor Intel Core i5-3470 using the compiler TDM-GCC 4.9.2 32-bit. Using the same hardware for testing the code \texttt{InvSqrt}, we obtained practically the same values of errors as those obtained by Lomont [38]. The same results were obtained also on Intel i7-5700. In this section, we analyze the rounding errors for the code \texttt{InvSqrt1}.

Applying algorithm \texttt{InvSqrt1}, we obtain relative errors InvSqrt1($\tilde{x}$) characterized by “oscillations” with a center slightly shifted with respect to the analytical approximate solution $\tilde{y}_{22}(\tilde{x}, t^{(1)})$; see
Figure 2. The observed blur can be expressed by a relative deviation of the numerical result from \( \hat{y}_{22}(\hat{x}) \):

\[
e^{(1)}(\hat{x}) = \frac{\text{InvSqrt1}(\hat{x}) - \hat{y}_{22}(\hat{x}, I^{(1)})}{\hat{y}_{22}(\hat{x}, I^{(1)})}.
\]  

(58)

The values of this error are distributed symmetrically around the mean value \( \langle e^{(1)} \rangle \):

\[
\langle e^{(1)} \rangle = 2^{-1} N_m^{-1} \sum_{x \in [1,4]} e^{(1)}(\hat{x}) = -1.398 \times 10^{-8}
\]  

(59)

enclosing the range:

\[
e^{(1)}(\hat{x}) \in [-9.676 \times 10^{-8}, 6.805 \times 10^{-8}],
\]  

(60)

see Figure 3. The blur parameters of the function \( e^{(1)}(\hat{x}, t) \) show that the main source of the difference between analytical and numerical results is the use of precision \texttt{float} and, in particular, rounding of constant parameters of the function InvSqrt1. We point out that in this case, the amplitude of the error oscillations is about 40% greater than the amplitude of oscillations of \( (\hat{y}_0 - \hat{y}_0) / \hat{y}_0 \) (i.e., in the case of InvSqrt); see the right part of Figure 2 in [46]. It is worth noting that for the first Newton–Raphson correction, the blur is of the same order, but due to a much higher error value in this case, its effect is negligible (i.e., Figure 1 would be practically the same with or without the blur). The maximum numerical errors practically coincide with the analytical result (53), i.e.,

\[
\Delta^{(1)}_{1,N_{\min}} \approx -8.76 \times 10^{-4}, \quad \Delta^{(1)}_{1,N_{\max}} \approx 8.76 \times 10^{-4}.
\]  

(61)

In the case of the second Newton–Raphson correction, we compared results produced by InvSqrt1 with exact values of the inverse square root for all numbers \( x \) of the type \texttt{float} such that \( e^x \in [-126, 128] \). The range of errors was the same for all these intervals (except \( e^x = -126 \)):

\[
\Delta^{(1)}_{2,N}(x) = \text{sqrt}(x) * \text{InvSqrt1}(x) - 1. \in (-6.62 \times 10^{-7}, 6.35 \times 10^{-7})
\]  

(62)

For \( e^x = -126 \), the interval of errors was slightly wider: \((-6.72 \times 10^{-7}, 6.49 \times 10^{-7})\), which can be explained by the fact that the analysis presented in this paper is not applicable to subnormal numbers; see (1). Therefore, our tests showed that relative errors for all numbers of the type \texttt{float} belong to the interval \( (\Delta^{(1)}_{2,N_{\min}, \Delta^{(1)}_{2,N_{\max}}}) \), where:

\[
\Delta^{(1)}_{2,N_{\min}} \approx -6.72 \times 10^{-7}, \quad \Delta^{(1)}_{2,N_{\max}} \approx 6.49 \times 10^{-7}.
\]  

(63)

These values are significantly higher than the analytical result \( 5.76 \times 10^{-7} \) (see (57)), but are still much lower than the analogous error for the algorithm InvSqrt \( 4.60 \times 10^{-6} \); see [46].
Figure 2. Solid lines represent function $\Delta^1_2(\tilde{x}, t^{(1)})$. Its vertical shifts by $\pm 6 \times 10^{-8}$ are denoted by dashed lines. Finally, dots represent relative errors for 4000 random values $x \in (2^{-126}, 2^{128})$ produced by algorithm $\text{InvSqrt1}$.

Figure 3. Relative error $\epsilon^{(1)}$ arising during the float approximation of corrections $\tilde{y}_{22}(\tilde{x}, t)$. Dots represent errors determined for 2000 random values $\tilde{x} \in [1, 4)$. Solid lines represent maximum ($\max_i$) and minimum ($\min_i$) values of relative errors (intervals $[1, 2)$ and $[2, 4)$ were divided into 64 equal intervals, and then, extremum values were determined in all these intervals).

6. Conclusions

In this paper, we presented a modification of the famous code $\text{InvSqrt}$ for fast computation of the inverse square root of single-precision floating-point numbers. The new code had the same magic constant, but the second part (which consisted of Newton–Raphson iterations) was modified. In the case of one Newton–Raphson iteration, the new code $\text{InvSqrt1}$ had the same computational cost as
InvSqrt and was two-times more accurate. In the case of two iterations, the computational cost of the new code was only slightly higher, but its accuracy was higher by almost seven times.

The main idea of our work consisted of modifying coefficients in the Newton–Raphson method and demanding that the maximal error be as small as possible. Such modifications can be constructed if the distribution of errors for Newton–Raphson corrections is not symmetric (like in the case of the inverse square root, when they are non-positive functions).

Author Contributions: Conceptualization, L.V.M.; formal analysis, C.J.W.; investigation, C.J.W., L.V.M., and J.L.C.; methodology, C.J.W. and L.V.M.; visualization, C.J.W.; writing, original draft, J.L.C.; writing, review and editing, J.L.C.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References


