Mark Allie
and Richard Lyons

A Root of Less Evil

In this article, we discuss several useful "tricks" for estimating the square root of a number. Our focus will be on high-speed (minimum computations) techniques for approximating the square root of a single value as well as the square root of a sum of squares for quadrature (I/Q) vector magnitude estimation.

In the world of DSP, the computation of a square root is found in many applications: calculating root mean squares, computing the magnitudes of fast Fourier transform (FFT) results, implementing automatic gain control (AGC) techniques, estimating the instantaneous envelope of a signal (AM demodulation) in digital receivers, and implementing three-dimensional (3-D) graphics algorithms. The fundamental tradeoff when choosing a particular square root algorithm is execution speed versus algorithm accuracy. In this article, we disregard the advice of legendary lawman Wyatt Earp (1848–1929), "Fast is important, but accuracy is everything. . . ," and concentrate on various high-speed square root approximations. In particular, we focus on algorithms that can be implemented efficiently in fixed-point arithmetic. We have also set other constraints: no divisions are allowed; only a small number of computational iterations are permitted; and only a small look-up table (LUT), if necessary, is allowed.

The first two methods below describe ways to estimate the square root of a single value using iterative methods. The last two techniques are methods for estimating the magnitude of a complex number.

NEWTON-RAPHSON INVERSE METHOD

A venerable technique for computing the square root of x is the so-named "Newton-Raphson square root" iterative technique to find $y(n)$, the approximation of

$$\sqrt{x} \approx y(n+1) = [y(n) + x/y(n)]/2. \quad (1)$$

Variable n is the iteration index, and $y(0)$ is an initial guess of $\sqrt{x}$ used to compute $y(1)$. Successive iterations of (1) yield more accurate approximations to $\sqrt{x}$.

However, to avoid the computationally expensive division by $y(n)$ in (1), we can use the iterative Newton-Raphson inverse (NRI) method. First, we find the approximation to the inverse square root of x using (2), where $p = 1/\sqrt{x}$. Next, we compute the square root of x by multiplying the final p by x

$$p(n+1) = 0.5\, p(n)[3 - xp(n)^2]. \quad (2)$$

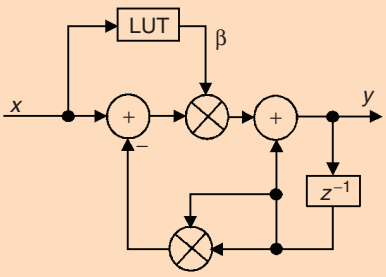
Two iterations of (2) provide surprisingly good accuracy. The initial inverse square root value $p(0)$, as defined by (3), is used in the first iteration

$$p(0) = 1/(2x/3 + 0.354167). \quad (3)$$

Indeed, (3) was selected because it is the reciprocal of the initial value expression used in the next iterative square root method presented below.

The square root function looks more linear when we restrict the range of our x input. As it turns out, it's convenient to limit the range of x to $0.25 \leqslant x < 1$. So if $x < 0.25$, then it must be *normalized* and, fortunately, we have a slick way to do so. When x needs normalization, then x is multiplied by 4^n until $0.25 \leqslant x < 1$. A factor of four is two bits of arithmetic left shift. The final square root result is *denormalized* by a factor of 2^n (the square root of 4^n). A denormalizing factor of two is a single arithmetic right shift. Implementing this normalization ensures $0.25 \leqslant x < 1$. The error curve for this two-iteration NRI square root method is shown in Figure 1, where we see the maximum error is roughly 0.0008%. The curve is a plot of the estimated square root divided by the true square root of x. That maximum error value is impressively small; it is almost worth writing home about because no arithmetic division is needed. But what if our data throughput requirements only allow us to perform one iteration of (2)? In that case, the maximum NRI method error is 0.24%. We see a truism of signal processing here—improved accuracy requires additional computations. There's no free lunch in DSP.

"DSP Tips and Tricks" introduces practical tips and tricks of design and implementation of signal processing algorithms so that you may be able to incorporate them into your designs. We welcome readers who enjoy reading this column to submit their contributions. Contact Associate Editors Rick Lyons (r.lyons@ieee.org) or Amy Bell (abell@vt.edu).



[FIG1] NIIRF implementation.

1053-5888/05/$20.00©2005IEEE

[TABLE1] β VERSUS NORMALIZED-X LUT.

4 MSBs OF x	β FIXED-POINT	β FLT-POINT
0100	0x7b20	0.961914
0101	0x6b90	0.840332
0110	0x6430	0.782715
0111	0x5e10	0.734869
1000	0x5880	0.691406
1001	0x53c0	0.654297
1010	0x4fa0	0.622070
1011	0x4c30	0.595215
1100	0x4970	0.573731
1101	0x4730	0.556152
1110	0x4210	0.516113
1111	0x4060	0.502930

This NRI method is not recommended for fixed-point format, with its sign and fractional bits. The coefficients, $p(n)$, and intermediate values in (2) are greater than one. In fixed-point math, using bits to represent internal results greater than one increases the error by a factor of two per bit. (Certainly, using more bits for the integer part of intermediate values without taking them away from the fractional part can be done, but only at the cost of more CPU cycles and memory.)

NONLINEAR IIR FILTER METHOD

The next iterative technique, by Mikami et al. [1], is specifically aimed at fixed-point implementation. It is configured as a non-linear IIR filter (NIIRF) as depicted in Figure 2, where again input x has been normalized to the range $0.25 \leqslant x < 1$. The output is given by (4) where the constant β, called the iteration acceleration factor, depends on the value of input x

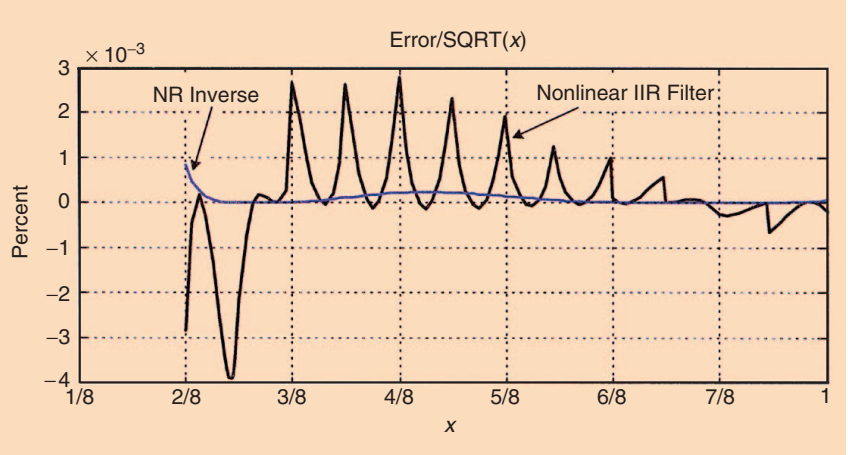
$$y(n+1) = \beta[x - y(n)^2] + y(n). \quad (4)$$

The magnitude of the error of (4) is known in advance when $\beta = 1$. The β acceleration factor scales the square root update term, $x - y(n)^2$, so the new estimate of y has less error. The acceleration factor results in reduced error for a given number of iterations.

In its standard form, this technique uses an LUT to provide β and performs two iterations of (4). For the first iteration, $y(0)$ is initialized using

$$y(0) = 2x/3 + 0.354167. \quad (5)$$

The constants in (5) were determined empirically. The acceleration constant β, based on x, is stored in an LUT made up of 12 subintervals of x with an equal width of 1/16. This square root method is implemented with the range of x set between 4/16 and 16/16 (12 regions), as was done for the NRI method. This convenient interval directly yields the offset for the LUT when using the four most significant mantissa bits of the normalized x value as a pointer, as shown in Table 1.



[FIG2] Normalized error for the NRI and NIIRF methods.

(A listing of a fixed-point DSP assembly code, simulated as an ADSP218x part implementing this NIIRF. It is available at http://www.cspl.umd.edu/spm/tips-n-tricks/. Various MATLAB square root modeling routines are also available at this Web site.)

Figure 1 shows the *normalized* NIIRF error curves as a function of x. (Normalized, in this instance, means the estimated square root is divided by the true square of x.) The NRI square root method is the most accurate in floating-point math, followed by the NIIRF method. This is not surprising considering the greater number of operations needed to implement the NRI method. The effect of the LUT for β is clearly seen in Figure 1, where the curve shows discontinuities (spikes) at the table look-up boundaries. These are, of course, missing from the NRI error curve.

One sensible step, when presented with algorithms such as the NIIRF square root method, is to experiment with variations of the algorithm to investigate accuracy versus computational cost. (We did that for the above NRI method by determining the maximum error when only one iteration of (2) was performed.) With this *explore and investigate* thought in mind, we examined seven variations of this NIIRF algorithm. The first variation, a simplification, is the original NIIRF algorithm using only one iteration. We then studied the following quadratic function of x to find β

$$\beta = 0.763x^2 - 1.5688x + 1.314. \quad (6)$$

Next, we used a linear function of x to find β, as defined by

$$\beta = -0.61951x + 1.0688. \quad (7)$$

For the quadratic and linear variations used to compute β, we investigated their performance when using both one and two iterations of (4).

For the final NIIRF method variation, we set β equal to the constants 0.633 and 0.64 for two and one iteration variations, respectively. Table 2 provides a comparison of the error magnitude behavior of the original two-iteration (LUT) NIIRF algorithm and the variations detailed above.

For all the variations listed in Table 2, the range of the input x was limited to $0.25 \leqslant x < 1$. (The term *normalized,* as used in Table 2, means the estimated square root divided by the true square of x.)

The error data in Table 2 was generated using double-precision floating-point math. Comparing the accuracy and complexity of the NRI and NIIRF methods, we might ask why ever use the NIIRF method? Oftentimes, double-precision floating-point math is not available to us. Fixed-point data is one of the assumptions of this article. This fixed-point constraint requires that the algorithms must be reevaluated for the error when fixed-point math is used. It turns out this is the reason why Mikami et al. developed the NIIRF square root method in the first place [1].

In fixed-point math, most of the internal values are close to each other and are less than one. Thus, these values result in a lower error using the NIIRF method compared to the NRI method when both are implemented in fixed-point format. In fact, there is less error in the fixed-point implementation of the NIIRF method than in the floating-point implementation. All of the NIIRF variations are well-suited to fixed-point math.

Next, we look at high-speed square root approximations used to estimate the magnitude of a complex number.

BINARY-SHIFT MAGNITUDE ESTIMATION

When we want to compute the magnitude M of the complex number $I + jQ$, the exact solution is

$$M = \sqrt{I^2 + Q^2}. \qquad (8)$$

To approximate the square root operation in (8), the following *binary-shift magnitude estimation* algorithm can be used to estimate M using

$$M \approx A\mathrm{M_{ax}} + B\mathrm{M_{in}} \qquad (9)$$

where A and B are constants, $\mathrm{M_{ax}}$ is the maximum of either $|I|$ or $|Q|$, and $\mathrm{M_{in}}$ is the minimum of $|I|$ or $|Q|$.

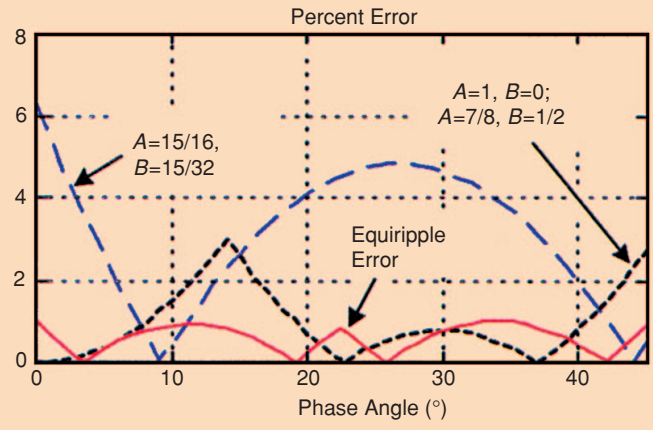
Many combinations of A and B are provided in [2], yielding various accuracies for estimating M. However, of special interest is the combination $A = 15/16$ and $B = 15/32$ using

$$M \approx \frac{15}{16}\mathrm{M_{ax}} + \frac{15}{32}\mathrm{M_{in}}. \qquad (10)$$

This algorithm's appeal is that it requires no explicit multiples because the A and B values are binary fractions and the formula is implemented with simple binary right-shifts, additions, and subtractions. (For example, a 15/32 times z multiply can be performed by first shifting z right by four bits and subtracting that number from z to obtain 15/16 times z. That result is then shifted right one bit). This algorithm estimates the magnitude M with

[TABLE2] ITERATIVE ALGORITHM NORMALIZED ABSOLUTE ERROR.

ALGORITHM	MAX. NORMALIZED ERROR (%)	MEAN NORMALIZED ERROR (%)
NR INVERSE (NRI)		
NRI, 2 Iters	8.4E-4	8.3E-5
NRI, 1 Iter	0.24	0.057
NIIRF FLOATING-PT		
LUT β, 2 Iters	0.004	5.4E-4
LUT β, 1 Iter	0.099	0.026
Quad. β, 2 Iters	0.0013	2.8E-4
Quad. β, 1 Iter	0.056	0.019
Linear β, 2 Iters	0.024	0.0061
Linear β, 1 Iter	0.28	0.088
$\beta = 0.633$, 2 Iters	0.53	0.05
$\beta = 0.64$, 1 Iter	1.44	0.23
NIIRF FIXED-PT		
NIIRF, 2 Iters	0.0035	5.1E-4
Quad. β, 2 Iters	0.0019	4.1E-4
Linear β, 2 Iters	0.011	0.0029



[FIG3] **Error of binary-shift and equiripple-error methods.**

a maximum error of 6.2% and a mean error of 3.1%. The percent error of this binary-shift magnitude estimation scheme is shown by the dashed curve in Figure 3 as a function of the angle between I and Q. (The curves in Figure 3 repeat every 45°.)

At the expense of a compare operation, we can improve the accuracy of (10) [3]. If $M_{in} \leqslant M_{ax}/4$, we use the coefficients $A = 1$ and $B = 0$ to compute (9); otherwise, if $M_{in} > M_{ax}/4$, we use $A = 7/8$ and $B = 1/2$. This dual-coefficient (and still multiplier-free) version of the binary-shift square root algorithm has a maximum error of 3.0% and a mean error of 0.95%, as shown by the dotted curve in Figure 3.

EQUIRIPPLE-ERROR MAGNITUDE ESTIMATION

Another noteworthy scheme for computing the magnitude of the complex number $I + jQ$ is the *equiripple-error magnitude estimation* method by Filip [4]. This technique, with a maximum error of roughly 1%, is sweet in its simplicity: if $M_{in} \leqslant 0.4142135 M_{ax}$, the complex number's magnitude is estimated using

$$M \approx 0.99 M_{ax} + 0.197 M_{in}. \quad (11)$$

On the other hand, if $M_{in} > 0.4142135 M_{ax}$, M is estimated by

$$M \approx 0.84 M_{ax} + 0.561 M_{in}. \quad (12)$$

This algorithm is so named because its maximum error is 1.0% for both (11) and (12). Its mean error is 0.6%. Because its coefficients are not simple binary fractions, this method is best suited for implementations on programmable hardware. The percent error of this equiripple-error magnitude estimation method is also shown in Figure 3, where we see the more computationally intensive method, equiripple-error, is more accurate. As usual, accuracy comes at the cost of computations. Although these methods are less accurate than the iterative square root techniques, they can be useful in those applications where high accuracy is not needed, such as when M is used to scale (control the amplitude of) a system's input signal.

One implementation issue to keep in mind when using integer arithmetic is that, even though values $|I|$ and $|Q|$ may be within your binary word width range, the estimated magnitude value may be too large to be contained within the numeric range. The practitioner must limit $|I|$ and $|Q|$ in some way to ensure that the estimated M value does not cause overflows.

SUMMARY

We have discussed several square root and complex vector magnitude approximation algorithms, with a focus on high-throughput (high-speed) algorithms as opposed to high-accuracy methods. We investigated the performance of several variations of iterative NRI and NIIRF square root methods and found, not surprisingly, that the number of iterations has the most profound effect on accuracy. The NRI method is not appropriate for implementation using fixed-point fractional binary arithmetic, while the NIIRF technique and the two magnitude estimation schemes lend themselves nicely to fixed-point implementation. The three magnitude estimation methods are less accurate but more computationally efficient than the NRI and NIIRF schemes. All algorithms described here are open to modification and experimentation. This will make them more accurate and computationally expensive, or less accurate and computationally "cheap." Your accuracy and data throughput needs determine the path you take to a root of less evil.

AUTHORS

Mark Allie is presently an assistant faculty associate in the Department of Electrical and Computer Engineering at the University of Wisconsin at Madison. He received his B.S. and M.S. degrees in 1981 and 1983, respectively, in electrical engineering from the same department. He has been employed as an engineer and consultant in the design of analog and digital signal processing systems for over 20 years.

Richard Lyons is a consulting systems engineer and lecturer with Besser Associates in Mt. View, California. He is the author of *Understanding Digital Signal Processing 2/E* (Prentice-Hall, 2004) and an associate editor for *IEEE Signal Processing Magazine*.

REFERENCES

[1] N. Mikami et al., "A new DSP-oriented algorithm for calculation of square root using a nonlinear digital filter," *IEEE Trans. Signal Processing*, pp. 1663–1669, July 1992.

[2] R. Lyons, *Understanding Digital Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2004, pp. 481–482.

[3] W. Adams and J. Brady, "Magnitude approximations for microprocessor implementation," *IEEE Micro*, pp. 27–31, Oct. 1983.

[4] A. Filip, "Linear approximations to $\sqrt{x2 + y2}$ having equiripple error characteristics," *IEEE Trans. Audio Electroacoust.*, pp. 554–556, Dec. 1973.

SP