"Partially rounded" Small-Order Approximations for Accurate, Hardware-Oriented, Table-Based Methods

Jean-Michel Muller
CNRS - Laboratoire LIP
ENSL/CNRS/INRIA Arenaire Project
Ecole Normale Supérieure de Lyon
46 Allee d'Italie
69364 Lyon Cedex 07
FRANCE
Jean-Michel.Muller@ens-lyon.fr

Keywords Computer arithmetic, elementary and special functions, table-based methods, polynomial approximations.

Abstract

*We aim at evaluating elementary and special functions using small tables and small, rectangular, multipliers. To do that, we show how accurate polynomial approximations whose order-*1 *coefficients are small in size (a few bits only) can be computed. We compare the obtained results with similar work in the recent literature.*

1 Introduction

This paper deals with hardware-oriented methods for implementing elementary (sine, cosine, exponential, etc.), special (gamma, erf, Bessel, etc.), or special-purpose functions. We assume that a rather low precision (say, from 10 to 32 bits) is required.

Various methods have been suggested for tackling with this problem. CORDIC-like algorithms [15, 17, 2, 6, 16, 1] may be attractive for some functions, but they are made up using some algebraic relations (such as $\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y)$) that are satisfied by the elementary functions only. The other methods are almost all built from at least one of the following two ideas:

- Since we can easily implement additions, multiplications (and possibly, divisions), the first idea that springs in mind is to approximate a function by combinations of these basic operations, that is, by polynomial (and possibly, rational) functions;
- the continuing progress of VLSI technology allows the implementation of larger and larger tables, it therefore makes sense to directly tabulate a function (when very low precision is at stake), or to combine tabulation and a few arithmetic operations.

Let us now quickly present some recent methods, that make use of the above ideas in quite different ways.

1.1 The bipartite method

The bipartite method was originally introduced by Das Sarma and Matula [10], with the aim of getting accurate reciprocals. Later on, generalizations to "symmetric" and "multipartite" tables and/or improvements have been suggested by Schulte and Stine [11, 12, 13, 14], Muller [7], and De Dinechin and Tisserand [3].

Assume an n-bit, binary fixed-point system, and – to simplify the presentation – assume that n is a multiple of 3, $n = 3k$. We wish to design a table-based implementation of function f. The straightforward method would consist in tabulating all possible 2^n values of $f(x)$. This would lead to a table of size $n \times 2^n$ bits. Instead of that, let us split the binary representation of the input value into 3 k-bit words x_0, x_1 and x_2, that is,

$$x = x_0 + x_1 2^{-k} + x_2 2^{-2k}$$

where x_0, x_1 and x_2 are multiples of 2^{-k} that are less than 1. The original bipartite method consists in approximating the order-1 Taylor expansion

$$\begin{gathered} f(x) = f\left(x_0 + x_1 2^{-k}\right) \\ + x_2 2^{-2k} f'\left(x_0 + x_1 2^{-k}\right) \\ + x_2^2 2^{-4k} f''(\xi), \\ \xi \in [x_0 + x_1 2^{-k}, x] \end{gathered}$$

by

$$f(x) = f\left(x_0 + x_1 2^{-k}\right) + x_2 2^{-2k} f'(x_0)\,.$$

That is, $f(x)$ is approximated by the sum of two functions $\alpha(x_0, x_1)$ and $\beta(x_0, x_2)$, where

$$\begin{cases} \alpha(x_0, x_1) &= f\left(x_0 + x_1 2^{-k}\right) \\ \beta(x_0, x_2) &= x_2 2^{-2k} f'(x_0) \end{cases}$$

The error of this approximation is roughly proportional to 2^{-3k} (see references [10, 11, 12, 13, 14, 7] for a more detailed error analysis). Instead of directly tabulating function f, we tabulate functions α and β. Since they are functions of $2k$ bits only, each of these tables has $2^{2n/3}$ entries. This results in a total table size of $2n \times 2^{2n/3}$ bits, which is a very significant improvement.

1.2 Methods using tabulation and a few multiplications.

Another solution is to split the input interval into some number of small sub-intervals, and store in a table, for each sub-interval, the coefficients of a low-degree polynomial approximation. The rationale behind that choice is that, for a given degree, the accuracy of an approximation is drastically improved if the size of the interval of approximation decreases. This is illustrated by Fig. 1.

Many variants to this general idea have been suggested. For instance, Piñeiro et al. [8] divide the input interval into around 2^8 subintervals. They store, for each subinterval, a degree-2 minimax approximation, and accumulate the partial terms in a fused accumulation tree. Cao et al. [5] store function values instead of coefficients and perform interpolation, using fewer look-up table memory entries, at the expense of additional hardware and extra time for calculating the coefficients on-the-fly.

Before introducing our method, we need to recall some classical results on "minimax" polynomial approximation, that will be used in the sequel of this paper.

1.3 Some reminders on minimax approximation

We denote by $\mathcal{P}_n$ the set of the polynomials of degree less than or equal to n. In the following, $||f - p||_\infty$ denotes the *distance*:

$$||f - p||_\infty = \max_{a \le x \le b} |f(x) - p(x)|.$$

We look for a polynomial p^* that satisfies:

$$||f - p^*||_\infty = \min_{p \in \mathcal{P}_n} ||f - p||_\infty.$$

The polynomial p^* is called the *minimax* degree-n polynomial approximation to f on $[a, b]$. The following result, due to Chebyshev, gives a characterization of the minimax approximations to a function[1].

Theorem 1 (Chebyshev) *p^* is the minimax degree-n approximation to f on $[a, b]$ if and only if there exist at least $n + 2$ values*

$$a \le x_0 < x_1 < x_2 < \ldots < x_{n+1} \le b$$

such that:

$$\begin{aligned} p^*(x_i) - f(x_i) &= (-1)^i \left[p^*(x_0) - f(x_0)\right] \\ &= \pm ||f - p^*||_\infty. \end{aligned}$$

An algorithm, due to Remez [4, 9], computes the minimax degree-n approximation to a continuous function iteratively. That algorithm is implemented on many packages such as Maple or Mathematica. For instance, to compute some of the approximations used in this paper, we have used the `minimax` function provided in the Maple computer algebra package.

1.4 Our goals

The methods – such as the bipartite method – that do not use multipliers are very helpful for small precision implementation (say, up to 16 bits), but larger precisions cannot be reached without requiring huge tables. Even the best improvements to the bipartite method cannot dismiss the fact that that method is an order-1 method: with tables with p address bits, it seems difficult to get more than around $2p$ bits of accuracy.

Hence, we focus on methods that require a few multiplications. Our main interest will be on *order-2 methods*. When using such methods, the coefficients of the polynomial approximations are, in general, not exactly representable with a small number of bits. Thus, they are truncated or rounded to the nearest, say k-bit, number. Choosing k results from a compromise between accuracy of approximation, and multiplier size and delay.

With an order-2 approximation, the truncation of the order-2 coefficient has a small effect only (unless k is very small), on the final accuracy. And yet, the truncation of the order-1 coefficient may have a strong influence on the accuracy of the approximation. The question that immediately springs in mind is *how much is the truncated best polynomial approximation to f close to the best approximation among the "truncated polynomials" ?* This is the very question that we try to address in this paper. We start from the minimax approximations to some functions, round their order-1 coefficient, and try to get better approximations than

[1] Although Chebyshev worked on both kinds of approximation, the minimax approximation should not be confused with the polynomial approximation that uses orthogonal Chebyshev polynomials. In practice, the minimax approximation is slightly better than the other one.

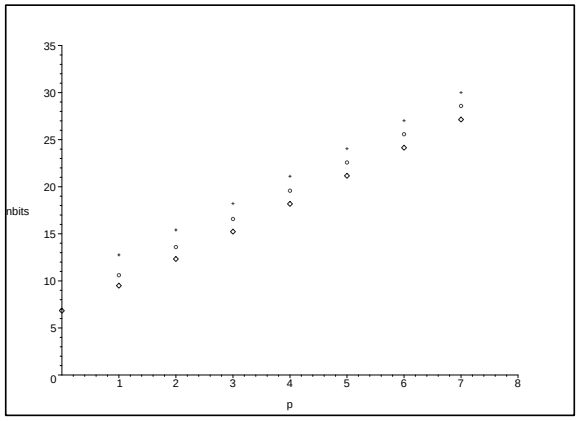

Figure 1. *Number of bits of accuracy of the degree-2 minimax piecewise approximations to* $\sin(x)$ *(circles),* $\exp(x)$ *(diamonds) and* $\sqrt{1+x}$ *(crosses), for* $x \in [0,1]$*. The interval is split into* 2^p *subintervals of equal size. For each subinterval a minimax approximation (called "subapproximation") is computed. We give here, as a function of* p*, the number of bits of accuracy of the less accurate subapproximation. Roughly speaking, this number of bits of accuracy grows linearly with* p*.*

the "truncated best one" by partially compensating (with the other coefficients) for the modification of the order-1 coefficient. Examples of such new approximations are given in Table 4. Similar "compensations" have already been done by Piñeiro, Bruguera and Muller [8]. From an existing polynomial approximation $a_0 + a_1 x + a_2 x^2$ to some function f, they round a_1 to the nearest k-bit number a_1^* and recompute a new polynomial approximation $a_0^* + a_1^* x + a_2^* x^2$ by noticing that

$$a_0^* + a_2^* X \approx f(\sqrt{X}) - a_1^* \sqrt{X}$$

where $X = x^2$, and computing a minimax computation of an approximation to $f(\sqrt{X}) - a_1^* \sqrt{X}$.

Here, we will show that there is no need to compute again a minimax approximation (a_0^* and a_2^* are easily deducible from a_0, a_2 and $a_1 - a_1^*$), and we will be able to predict how much accuracy is saved by such a compensation: around 3 bits.

2 Accurate "truncated" order-2 approximations

We aim at building degree-2 polynomial approximations to some regular enough function f, for which the coefficients of degree 1 are representable with a very small number of bits only. Let x be the input value. We assume that x is represented with n bits in fixed point, and is between 0 and 1. Let us denote $0.x_1x_2\ldots x_n$ this representation.

Fig. 2 shows the main blocks of an architecture implementing an order-2 approximation. The most p sig-

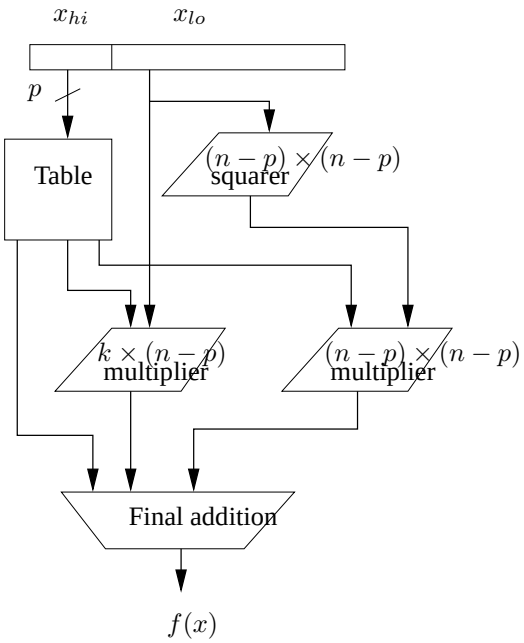

Figure 2. *Order-2 approximation (see for instance [8].*

nificant bits of x are used as address bits, to look up in a table a degree-2 approximation to $f(x)$ in the interval $[h, h+2^{-p}]$, where $h = 0.x_1x_2\ldots x_p$. Define $\ell = x - h = 0.000\cdots0x_{p+1}x_{p+2}\cdots x_n$. To design a suitable approximation, we will start from the degree-2 minimax approximation to f in $[h, h+2^{-p}]$, expressed as a function of ℓ:

$$P(\ell) = a_0 + a_1\ell + a_2\ell^2 \approx f(x).$$

Define $\epsilon_{\text{minimax}}$ as the error of this approximation, that is:

$$\epsilon_{\text{minimax}} = \max_{\ell\in[0,2^{-p}]} |P(\ell) - f(h+\ell)|.$$

We wish to compute (and then to store in a table) a slightly different polynomial approximation to f, for which the degree-1 coefficient has a binary representation with a small number, say k, of bits. Let us denote

$$P^*(\ell) = a_0^* + a_1^*\ell + a_2^*\ell^2$$

that new approximation. We wish $P^*(\ell)$ to be as close as possible to $P(\ell)$ for $\ell \in [0, 2^{-p}]$. This means

$$(a_1 - a_1^*)\,\ell \approx (a_0^* - a_0) + (a_2^* - a_2)\,\ell^2 \quad \text{for } \ell \in [0, 2^{-p}].$$

Our method consists in first choosing a_1^* as the k-bit number that is closest to a_1. By doing that, we now have to find an approximation

$$\delta_0 + \delta_2\ell^2$$

to $(a_1 - a_1^*)\ell$. The coefficients a_0^* and a_2^* will be obtained by adding δ_0 and δ_2 to a_0 and a_2, respectively. Define $L = \ell^2$. Our problem reduces to finding in the interval $[0, 2^{-2p}]$ (i.e., the interval where L lies) an order-1 approximation to $(a_1 - a_1^*)\sqrt{L}$. Such an approximation is obtained by multiplying by $(a_1 - a_1^*)$ an approximation to $\sqrt{L}$. Hence, in the next section, we get minimax approximations to the square root function.

2.1 Order-1 minimax approximations to the square-root function

Concerning degree-1 approximations to the square root, there is no need to run Remez' algorithm. Theorem 1 makes it possible to directly get minimax approximations.

Theorem 2 *The degree-1 minimax approximation to $\sqrt{L}$ in the interval $[0, 2^{-2p}]$ is*

$$2^{-p-3} + 2^p L,$$

and the error of this approximation is 2^{-p-3}.

Proof. From Theorem 1, the maximum distance between the linear approximation and the square root is reached at 3 points. The concavity of the square root function implies that two of these points are 0 and 2^{-2p} (see Figure 3). Let us call α the third point. Let us denote $A_0 + A_1L$ the linear approximation, and $\epsilon_{\surd}$ the error of approximation. We have

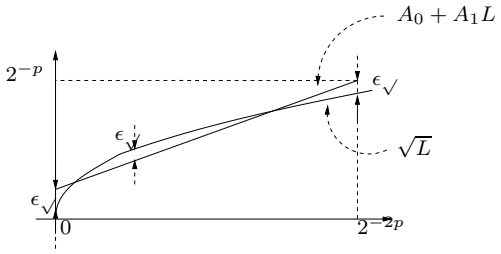

Figure 3. *Minimax order-1 approximation to the square-root in the interval $[0, 2^{-2p}]$.*

$$\begin{cases} A_0 & = & \epsilon_{\surd} \\ A_0 + A_1 2^{-2p} - \sqrt{2^{-2p}} & = & \epsilon_{\surd} \\ \sqrt{\alpha} - A_0 - A_1\alpha & = & \epsilon_{\surd} \end{cases} \tag{1}$$

Moreover, since function $\sqrt{L} - A_0 - A_1L$ reaches its maximum value at $L = \alpha$, the derivative of this function is zero at this point. Therefore

$$\frac{1}{2\sqrt{\alpha}} - A_1 = 0 \tag{2}$$

Elementary calculation from (1) and (2) gives the result.

2.2 Coefficients and error bounds

The previous two subsections allow us to get the coefficients of P^*. These coefficients a_0^*, a_1^* and a_2^* are

$$\begin{cases} a_1^* & = & a_1 \text{ rounded to } k \text{ bits} \\ a_0^* & = & a_0 + (a_1 - a_1^*)2^{-p-3} \\ a_2^* & = & a_2 + (a_1 - a_1^*)2^p \end{cases} \tag{3}$$

and the approximation error is upper-bounded by[2]

$$\begin{aligned} &\epsilon_{\text{method}} \\ &= \epsilon_{\text{minimax}} + |a_1 - a_1^*|\,\epsilon_{\surd} \\ &= \epsilon_{\text{minimax}} + |a_1 - a_1^*|\,2^{-p-3} \end{aligned} \tag{4}$$

Now, we can easily get a bound on the error committed without using our method if we just round a_1 in the initial

[2] Of course (5) is an upper bound, and to get a tighter error bound, it is much preferable to directly calculate

$$\max_{\ell\in[0,2^{-p}]} |P^*(\ell) - f(h+\ell)|.$$

approximation. The error will be $\epsilon_{\text{minimax}}$ plus the maximum value of $|a_1 - a_1^*|\,\ell$, that is

$$\begin{array}{c} \epsilon_{\text{round}} \\ = \epsilon_{\text{minimax}} + |a_1 - a_1^*|\,2^{-p} \end{array} \quad (5)$$

This shows that when $\epsilon_{\text{minimax}}$ is much smaller than $|a_1 - a_1^*|\,2^{-p}$ (which happens in all practical cases), our method is 8 times more accurate than the naive rounding of coefficient a_1. Our strategy saves three bits of accuracy. This clearly appears in Figures 1, 2 and 3. In these figures, we have compared, for some very common functions (sine, exponential and $\log(1+x)$), the errors of the standard degree-2 minimax approximation ("exact", i.e., not truncated coefficients), the "rounded" minimax approximation (the order-1 coefficient is rounded to k bits), and our method. We also put the error of the order-1 minimax approximation (with "exact" coefficients). Table 4 gives the obtained coefficients for the exponential function with $p = k = 4$.

3 Using these results

We now give some examples that show how can our approximation method be used. We also compare the obtained results with some examples presented in the literature.

3.1 Exponential function with $p = 6$ and $k = 8$

Consider the case of the exponential function with $p = 6$ and $k = 8$ (that is, the table will contain 64 elements, and a_1^* will be an 8-bit number). Table 2 shows that the accuracy of approximation of the polynomials generated by our method is 18 bits. Hence, the final accuracy of an implementation, due to the rounding of a_0^*, a_2^* (and possibly ℓ in the squaring) cannot be better than or equal to 18 bits. Let us try to achieve 17 bits. To do that, let us try to make the error on the computation of $a_0^* + a_2^*\ell^2$ less than 2^{-18}, as follows:

- a_0^* will be rounded to the nearest number exactly representable with 18 fractional bits, say $\hat{a_0^*}$. This will give
$$\left|\hat{a_0^*} - a_0^*\right| \leq 2^{-19}$$
- we have to make sure that the computed value of $a_2^*\ell^2$ is at a distance from the exact value that is less than 2^{-19}.

The first question that should be addressed is how many bits of a_2^* do we keep, and what is the required accuracy when computing ℓ^2. Define $\hat{a_2^*}$ as a_2 rounded to some k' fractional bits. The number ℓ is less than 2^{-6}. Let ϵ be the error on ℓ^2 (either due to the fact that we truncate ℓ before computing ℓ^2, or to the fact that we truncate ℓ^2 before multiplying it by $\hat{a_2^*}$). From (3), the largest value of a_2^* is less than 2.

We have

$$\begin{array}{l} \left|a_2^*\ell^2 - \hat{a_2^*}(\ell+\epsilon)^2\right| \approx \left|a_2^* - \hat{a_2^*}\right|\ell^2 + 2\hat{a_2^*}\epsilon\ell \\ \leq 2^{-12-k'-1} + 4\epsilon\ell. \end{array}$$

To make this value less than 2^{-19} it suffices to choose $k' = 7$ and $\epsilon \leq 2^{-16}$. Again, to get $\epsilon \leq 2^{-16}$, it suffices to keep 8 bits of ℓ. Therefore, for each of the $2^8 = 64$ subintervals, the number of bits that must be stored is:

- 18 for a_0^*;
- 8 for a_1^*;
- 8 for a_2^* (7 for the fractional part, and 1 for the integer part).

Hence, to get a final accuracy of 17 bits, our method will require a table of $(18 + 8 + 8) \times 2^8$ bits $= 1088$ bytes. To get a similar accuracy, the bipartite method would require around 12 Kbytes of table.

3.2 Sine function with $p = 8$ and $k = 10$

A very similar calculation, with $p = 8$, $k = 10$, the sine function in $[0, 1]$ and the figures given by Table 1 shows that we can achieve 21 bits of accuracy with 7 stored bits for a_2^* and 22 stored bits for a_0^*. This leads to a table of 1.184 Kbytes. We can compare this figure with the best known multipartite decomposition, suggested by De Dinechin and Tisserand [3], who achieve 16 bits of accuracy with a table of similar size. And yet, our design requires the additional delay[3] of a 10×17 bit multiplication. By the way, that delay can be reduced to the delay of 5 additions if a_1^* is stored booth-recoded (to do that, we need 15 bits instead of 10 to store a_1^*).

3.3 Getting seed-values for Newton-Raphson division

The original bipartite method was designed in order to generate seed values (initial approximations to the reciprocal of a number) for Newton-Raphson division. We can as well use our method to generate accurate reciprocal approximations at low cost.

For instance, for reciprocals of mantissas of floating-point numbers (this reduces to $f = 1/(1+x)$ for $x \in [0, 1)$), the choice $p = 3$ and $k = 4$ makes it possible to get an accuracy of more than 10 bits with an extremely small table (40 bytes) and very small multiplications (4 bits of a_1^* and 4 bits of a_2^* do suffice).

[3] The multiplication that computes ℓ^2 is done in parallel with the table lookup.

Table 1. *Accuracy of various degree-2 approximations (expressed in number of bits) to the sine function in* $[0, 1]$*, assuming various values of* p *(number of subintervals) and* k *(size of* a_1^**). We compare the errors of the standard degree-2 minimax approximation, called here "best possible" (no limitation to the size of* a_1*), the "rounded" minimax approximation (*a_1 *is rounded to* k *bits and the other coefficients remain unchanged), our method and the degree-1 minimax approximation.*

p	k	best possible degree 2	rounded	our method	best possible degree 1
4	3	19.58	8.00	11.00	12.28
	4		9.00	11.99	
	5		10.05	13.04	
	6		11.06	14.03	
	7		12.43	15.36	
6	6	25.58	13.00	16.00	16.26
	7		14.00	17.00	
	8		15.01	18.00	
	10		17.01	19.99	
	12		19.06	21.93	
8	8	31.58	17.00	20.00	20.25
	10		19.00	22.00	
	12		21.00	23.99	
	14		23.01	25.99	
10					24.25

Table 2. *Accuracy of various degree-2 approximations (expressed in number of bits) to the exponential function in* $[0, 1]$*, assuming various values of* p *and* k*.*

p	k	best possible degree 2	rounded	our method	best possible degree 1
4	4	18.18	7.10	10.10	10.60
	5		8.24	11.23	
	6		9.44	12.41	
5	4	21.16	8.09	11.09	14.57
	5		9.08	12.08	
	6		10.31	13.30	
8	8	30.14	15.00	18.00	18.56
	10		17.04	20.04	
	12		19.06	22.06	
10					22.55

Table 3. *Accuracy of various degree-2 approximations (expressed in number of bits) to* $\log(1+x)$ *in* $[0,1]$*, assuming various values of* p *and* k*.*

p	k	best possible degree 2	rounded	our method	best possible degree 1
4	4	18.71	9.06	12.05	12.08
	5		10.03	13.03	
	6		11.02	14.00	
6	6	24.61	13.02	16.02	16.02
	7		14.00	17.00	
	8		15.02	18.01	
8	8	30.59	17.00	20.00	20.00
	10		19.00	22.00	

Table 4. *Coefficients of the degree-2 approximation to the exponential function in* $[0,1]$ *that corresponds to* $p = 4$ *(i.e.,* 16 *subintervals) and* $k = 4$*.*

interval	degree 0	degree 1	degree 2
$[0, \frac{1}{16}]$	0.11111111111111111110	1.000	0.100000101000
$[\frac{1}{16}, \frac{1}{8}]$	1.00010000011000111010	1.001	−0.011011001110
$[\frac{1}{8}, \frac{3}{16}]$	1.00100010000110100001	1.001	0.101101010101
$[\frac{3}{16}, \frac{1}{4}]$	1.00110100101101001111	1.010	−0.000101011101
$[\frac{1}{4}, \frac{5}{16}]$	1.01001000110001110010	1.010	1.001100110000
$[\frac{5}{16}, \frac{3}{8}]$	1.01011101111001001100	1.011	0.100100010000
$[\frac{3}{8}, \frac{7}{16}]$	1.01110100011000110010	1.100	0.000001011001
$[\frac{7}{16}, \frac{1}{2}]$	1.10001100100110010000	1.100	1.100100100011
$[\frac{1}{2}, \frac{9}{16}]$	1.10100110000111101001	1.101	1.001110000110
$[\frac{9}{16}, \frac{5}{8}]$	1.11000001010011011011	1.110	0.111110011101
$[\frac{5}{8}, \frac{11}{16}]$	1.11011110010000011110	1.111	0.110110000011
$[\frac{11}{16}, \frac{3}{4}]$	1.11111010000101111111	10.00	0.110101011000
$[\frac{3}{4}, \frac{13}{16}]$	10.00011110001011110 11	10.00	10.111100111001
$[\frac{13}{16}, \frac{7}{8}]$	10.01000000111010010010	10.01	1.0011010010101
$[\frac{7}{8}, \frac{15}{16}]$	10.01100101111010001 01	10.10	−0.0110010100010
$[\frac{15}{16}, 1]$	10.10001101110100110 10	10.10	10.0010100011011

Conclusion

We have suggested a way of partially compensating for the loss of accuracy due to truncation or rounding of the order-1 coefficient of a polynomial approximation to some function. Our method can be used for designing hardware implementation of functions that require much smaller tables than the bipartite (and, more generally, than the order-1) methods, and that only need small arithmetic operators.

References

[1] Elisardo Antelo, Tomas Lang, and Javier D. Bruguera. Very-high radix circular CORDIC: Vectoring and unified Rotation/Vectoring. *IEEE Transactions on Computers*, 49(7):727–739, July 2000.

[2] P. W. Baker. Suggestion for a fast binary sine/cosine generator. *IEEE Transactions on Computers*, C-25(11), November 1976.

[3] F. de Dinechin and A. Tisserand. Some improvements on multipartite table methods. In Burgess and Ciminiera, editors, *Proc. of the 15th IEEE Symposium on Computer Arithmetic (Arith-15)*. IEEE Computer Society Press, 2001.

[4] J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. G. Thacher, and C. Witzgall. *Computer Approximations*. Wiley, New York, 1968.

[5] B. Wei J. Cao and J. Cheng. High-performance architectures for elementary function generation. In Burgess and Ciminiera, editors, *Proc. of the 15th IEEE Symposium on Computer Arithmetic (Arith-15)*. IEEE Computer Society Press, 2001.

[6] J.M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston, 1997.

[7] J.M. Muller. A few results on table-based methods. *Reliable Computing*, 5(3), 1999.

[8] J.A. Pineiro, J.D. Bruguera, and J.M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In Burgess and Ciminiera, editors, *Proc. of the 15th IEEE Symposium on Computer Arithmetic (Arith-15)*. IEEE Computer Society Press, 2001.

[9] E. Remez. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Académie des Sciences, Paris*, 198, 1934.

[10] D. Das Sarma and D. W. Matula. Faithful bipartite rom reciprocal tables. In S. Knowles and W. McAllister, editors, *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, July 1995. IEEE Computer Society Press, Los Alamitos, CA.

[11] M. Schulte and J. Stine. Symmetric bipartite tables for accurate function approximation. In In T. Lang, J.M. Muller, and N. Takagi, editors, *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[12] Michael J. Schulte and James E. Stine. Accurate function evaluation by symmetric table lookup and addition. In Thiele, Fortes, Vissers, Taylor, Noll, and Teich, editors, *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (Zurich, Switzerland)*, pages 144–153. IEEE Computer Society Press, 1997.

[13] M.J. Schulte and J.E. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, Aug. 1999.

[14] James E. Stine and Michael J. Schulte. The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing*, 21:167–177, 1999.

[15] J. Volder. The CORDIC computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, 1959. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.

[16] Jack E. Volder. The birth of CORDIC. *Journal of VLSI Signal Processing Systems*, 25(2):101–105, June 2000.

[17] J. Walther. A unified algorithm for elementary functions. In *Joint Computer Conference Proceedings*, 1971. Reprinted in E. E. Swartzlander, *Computer Arithmetic*, Vol. 1, IEEE Computer Society Press Tutorial, Los Alamitos, CA,1990.