Reliable Computing **5:** 279–288, 1999.

A Few Results on Table-Based Methods

JEAN-MICHEL MULLER
CNRS, Laboratoire LIP, Projet ARENAIRE, Ecole Normale Supérieure de Lyon, France,
e-mail: Jean-Michel.Muller@ens-lyon.fr

(Received: 28 September 1998; accepted: 21 December 1998)

Abstract. Table-based methods are frequently used to implement functions. We examine some methods introduced in the literature, and we introduce a generalization of the bipartite table method, named the **multipartite table method**.

1. Introduction

Throughout the paper, f is the function to be evaluated. We assume n-bit, fixed-point arguments, between $1/2$ and 1 (that is, they are mantissas of floating-point numbers).

Table-based methods have frequently been suggested and used to implement some arithmetic (reciprocal, square root) and transcendental functions. One can distinguish three different classes of methods:

- **compute-bound methods:** these methods use table-lookup in a small table to find parameters used afterward for a polynomial or rational evaluation. The main part of the evaluation of f consists in arithmetic computations;
- **table-bound methods:** The main part of the evaluation of f consists in looking up in a generally rather large table. The computational part of the function evaluation is rather small (e.g., a few additions);
- **in-between methods:** these methods use the combination of table lookup in a medium-size table and a significant yet reduced amount of computation (e.g. one or two multiplications, or several "small multiplications" that use rectangular—fast and/or small—multipliers).

Many methods currently used on general-purpose systems belong to the first class (e.g. Tang's methods [8]–[11]. The third class of methods has been widely studied since 1981 [2]. The use of small (e.g., rectangular) multipliers to fasten the computational part of the evaluation has been suggested by several authors (see for instance Wong and Goto's algorithms for double precision calculations [13], or Ercegovac et al.'s methods [1]).

In this paper, we examine some table-bound methods. Of course, the straightforward method, consisting in building a table with n address bits, cannot be used unless n is very small. The first useful table-bound methods have been introduced

in the last decade: they have become implementable thanks to progress in VLSI technology. Wong and Goto [12] have suggested the following method. We split the binary representation of the input-number x into four k-bit numbers, where $k = n/4$. That is, we write*:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.
Then $f(x)$ is approximated by:

$$\begin{aligned} f(x_1 + x_2 2^{-k}) &+ \frac{1}{2} 2^{-k} \{ f(x_1 + x_2 2^{-k} + x_3 2^{-k}) - f(x_1 + x_2 2^{-k} - x_3 2^{-k}) \} \\ &+ \frac{1}{2} 2^{-2k} \{ f(x_1 + x_2 2^{-k} + x_4 2^{-k}) - f(x_1 + x_2 2^{-k} - x_4 2^{-k}) \} \\ &+ 2^{-4k} \left\{ \frac{x_3^2}{2} f^{(2)}(x_1) - \frac{x_3^3}{6} f^{(3)}(x_1) \right\}. \end{aligned}$$

The approximation error due to the use of this approximation is about 2^{-5k}.

The **bipartite table method** was first suggested by Das Sarma and Matula [4] for quickly computing reciprocals. A slight improvement, the **symmetric bipartite table method** was introduced by Schulte and Stine [6]. Due to the importance of the bipartite table method (BTM), we will present it in detail in the next section. Compared to Wong and Goto's method, it requires larger tables. And yet, the amount of computation required by the BTM is reduced to one addition.

The problem of evaluating a function given by a converging series can be reduced to the evaluation of a partial product array (PPA). Schwarz [7] suggested to use multiplier structures to sum up PPAs. Hassler and Takagi [3] use PPAs to evaluate functions by table look-up and addition.

2. Order-1 Methods

The methods described in this section use an order-1 Taylor approximation of f. This leads to very simple computations (mere additions), but the size of the required tables may be quite large.

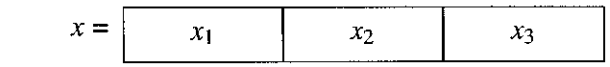
2.1. The Bipartite Table Method

This method was first suggested by DasSarma and Matula [4] for computing reciprocals. We split the binary representation of the input number x into 3 k-bit numbers, where $k = n/3$. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k}$$

* To make the paper easier to read and more consistent, we do not use Wong and Goto's notations here. We use the same notations as in the sequel of this paper.

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.



We then write the order-1 Taylor expansion of f at $x_1 + x_2 2^{-k}$. This gives:

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1 + x_2 2^{-k}) + \varepsilon_1 \tag{2.1}$$

with $\varepsilon_1 = \frac{1}{2} x_3^2 2^{-4k} f''(\xi_1)$, where $\xi_1 \in [x_1 + x_2 2^{-k}, x]$. Now, we approximate the value $f'(x_1 + x_2 2^{-k})$ by its order-0 Taylor expansion at x_1 (that is, by $f'(x1)$. This gives:

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1) + \varepsilon_1 + \varepsilon_2 \tag{2.2}$$

with $\varepsilon_2 = x_2 x_3 2^{-3k} f''(\xi_2)$, where $\xi_2 \in [x_1, x_1 + x_2 2^{-k}]$. This gives the **bipartite formula**:

$$f(x) = \alpha(x_1, x_2) + \beta(x_1, x_3) + \varepsilon \tag{2.3}$$

where

$$\begin{cases} \alpha(x_1, x_2) &= f(x_1 + x_2 2^{-k}), \\ \beta(x_1, x_3) &= x_3 2^{-2k} f'(x_1), \\ \varepsilon &\leq (\frac{1}{2} 2^{-4k} + 2^{-3k}) \max f'' \approx 2^{-3k} \max f''. \end{cases}$$

Hence, $f(x)$ can be approximated, with approximately n bits of accuracy (by this, we mean "with error $\approx 2^{-n}$") by the sum of two terms (α and β) that can be looked-up in $2n/3$-address bit tables, as illustrated by Figure 1. Moreover, whereas the first table (function α) must contain n-bit words, we can take into account the fact that approximately* $2k$ most significant bits of $\beta(x_1, x_3)$ are zero: there is no need to store them.

Assume we wish to compute the sine function, with 24-bit input numbers between 1/2 and 1, and error less than 2^{-24}. Functions f' and f'' are always less than 1. A straightforward use of what we have presented leads to choose $k = 9$, and to perform the summation (2.3) with 25-bit words. Hence the error ε of (2.3) is approximately 2^{-27}, and the error due to the truncation to 25 bits of α and β** is less than 2^{-25}. The first table has size 25×2^{17} (the first bit of x is a one: there is no need to use it as an address bit). The second table has size 7×2^{15}. All this requires 428 Kbytes of memory. It is worth noticing that one can get smaller tables by using symmetries and splitting n into sub-words of slightly different sizes. All this has been suggested and implemented by Schulte and Stine [5], [6].

* This depends on the values of f'.

** Assuming that the values stored in the tables are rounded to the nearest.

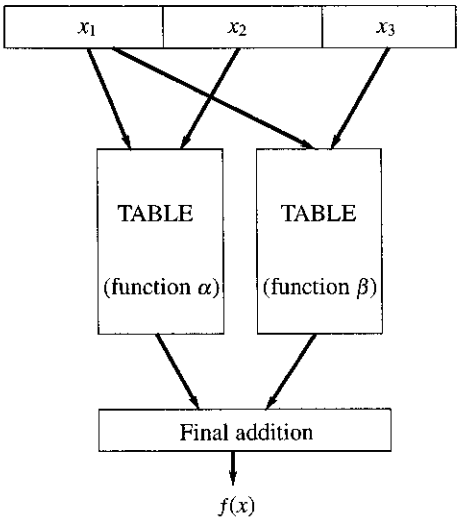


Figure 1. The bipartite table method.

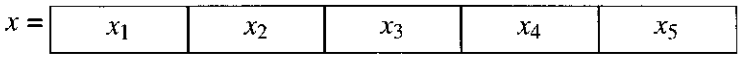
The BTM still leads to large tables in single precision, and thus it is far from being implementable in double precision. And yet, this leads to another idea: we should try to generalize the bipartite method, by splitting the input word into more than three parts. Let us first try a splitting into five parts, we will after that generalize to an arbitrary odd number of parts. It is worth noticing that Schulte and Stine also have proposed [5] a method based on a splitting into more than 3 parts. Their method is called the *Symmetric table addition method* (STAM). We will compare our method and the STAM method in Section 2.4.

2.2. The Tripartite Table Method

Now, we split the input n-bit fixed-point number x into five k-bit parts $x_1, x_2, \ldots, x_5$. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k} + x_5 2^{-4k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.



We use the order-1 Taylor expansion of f at $= x_1 + x_2 2^{-k} + x_3 2^{-2k}$:

$$\begin{aligned} f(x) &= f(x_1 + x_2 2^{-k} + x_3 2^{-2k}) \\ &\quad + (x_4 2^{-3k} + x_5 2^{-4k}) f'(x_1 + x_2 2^{-k} + x_3 2^{-2k}) \\ &\quad + \varepsilon_3 + \varepsilon_1 \end{aligned} \tag{2.4}$$

with $\varepsilon_3 = \frac{1}{2}(x_4 2^{-3k} + x_5 2^{-4k})^2 f''(\xi_3)$, with $\xi_3 \in [x_1 + x_2 2^{-k} + x_3 2^{-2k}, x]$, which gives $\varepsilon_3 \leq \frac{1}{2} 2^{-6k} \max f''$.

In (2.4), we expand the term $(x_4 2^{-3k} + x_5 2^{-4k}) f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ as follows:

- $x_4 2^{-3k} f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ is replaced by $x_4 2^{-3k} f'(x_1 + x_2 2^{-k})$. The error committed is $\varepsilon_4 = x_3 x_4 2^{-5k} f''(\xi_4)$, where $\xi_4 \in [x_1 + x_2 2^{-k}, x_1 + x_2 2^{-k} + x_3 2^{-2k}]$. We easily get $\varepsilon_4 \leq 2^{-5k} \max f''$.
- $x_5 2^{-4k} f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ is replaced by $x_5 2^{-4k} f'(x_1)$. The error committed is $\varepsilon_5 = (x_2 2^{-k} + x_3 2^{-2k}) x_5 2^{-4k} f''(\xi_5)$, where $\xi_5 \in [x_1, x_1 + x_2 2^{-k} + x_3 2^{-2k}]$. We get $\varepsilon_5 \leq 2^{-5k} \max f''$.

This gives the **tripartite formula**:

$$f(x) = \gamma(x_1, x_2, x_3) + \delta(x_1, x_2, x_4) + \theta(x_1, x_5) + \varepsilon, \tag{2.5}$$

where

$$\begin{aligned} \gamma(x_1, x_2, x_3) &= f(x_1 + x_2 2^{-k} + x_3 2^{-2k}), \\ \delta(x_1, x_2, x_4) &= x_4 2^{-3k} f'(x_1 + x_2 2^{-k}), \\ \theta(x_1, x_5) &= x_5 2^{-4k} f'(x_1), \\ \varepsilon &\leq \left(\frac{1}{2} 2^{-6k} + 2 \times 2^{-5k}\right) \max f'' \approx 2^{-5k+1} \max f''. \end{aligned}$$

Hence, $f(x)$ can be obtained by adding three terms, each of them being looked-up in a table with (at most) $3n/5$ address bits. This is illustrated by Figure 2.

2.3. Generalization: The Multipartite Table Method

The previous approach is straightforwardly generalized. We now assume that the n-bit input number x is split into $2p+1$ k-bit values $x_1, x_2, \ldots, x_{2p+1}$. That is,

$$x = \sum_{i=1}^{2p+1} x_i 2^{(i-1)k},$$

where the x_i's are multiples of 2^{-k} and satisfy $0 \leq x_i < 1$. As in the previous sections, we use the order-1 Taylor expansion:

$$\begin{aligned} f(x) = {} & f(x_1 + x_2 2^{-k} + \cdots + x_{p+1} 2^{-pk}) \\ & + (x_{p+2} 2^{(-p-1)k} + \cdots + x_{2p+1} 2^{-2pk}) f'(x_1 + x_2 2^{-k} + \cdots + x_{p+1} 2^{-pk}) \\ & + \varepsilon_{p+1} \end{aligned}$$

with $\varepsilon_{p+1} \leq \frac{1}{2} 2^{-2(p+1)k} \max f''$. We expand the term

$$(x_{p+2} 2^{(-p-1)k} + \cdots + x_{2p+1} 2^{-2pk}) f'(x_1 + x_2 2^{-k} + \cdots + x_{p+1} 2^{-pk}),$$

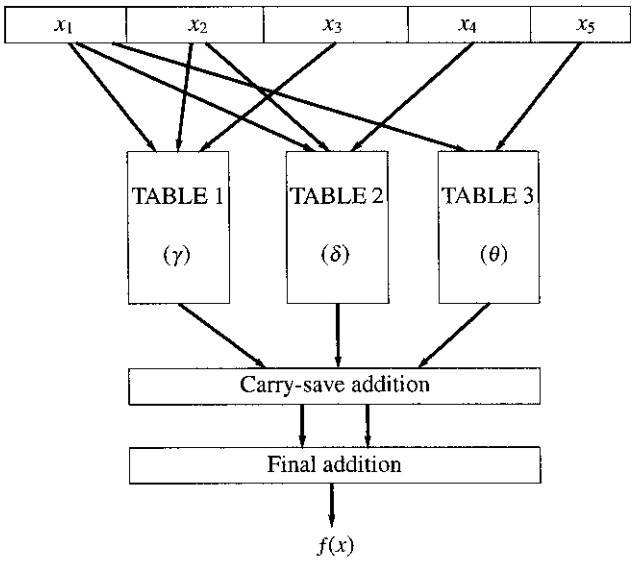


Figure 2. The tripartite table method.

and perform Taylor approximations to $f'(x_1 + x_2 2^{-k} + \cdots + x_{p+1}2^{-pk})$. We then get:

$$\begin{aligned} f(x) = {} & f(x_1 + x_2 2^{-k} + \cdots + x_{p+1}2^{-pk}) + \varepsilon_{p+1} \\ & + x_{p+2}2^{(-p-1)k} f'(x_1 + x_2 2^{-k} + \cdots + x_p 2^{(-p+1)k}) + \varepsilon_{p+2} \\ & + x_{p+3}2^{(-p-2)k} f'(x_1 + x_2 2^{-k} + \cdots + x_{p-1}2^{(-p+2)k}) + \varepsilon_{p+3} \\ & + x_{p+4}2^{(-p-3)k} f'(x_1 + x_2 2^{-k} + \cdots + x_{p-2}2^{(-p+3)k}) + \varepsilon_{p+4} \\ & \cdots \\ & + x_{2p+1}2^{-2pk} f'(x_1) + \varepsilon_{2p+1} \end{aligned}$$

where $\varepsilon_{p+2}, \varepsilon_{p+3}, \ldots, \varepsilon_{2p+1}$ are less than $2^{(-2p-1)k} \max f''$.

This gives the **multipartite (or $(p+1)$-partite) formula**:

$$\begin{aligned} f(x) = {} & \alpha_1(x_1, x_2, \ldots, x_{p+1}) \\ & + \alpha_2(x_1, x_2, \ldots, x_{p-1}, x_{p+3}) \\ & + \alpha_3(x_1, x_2, \ldots, x_{p-2}, x_{p+4}) \\ & + \alpha_4(x_1, x_2, \ldots, x_{p-3}, x_{p+5}) \\ & \cdots \\ & + \alpha_{p+1}(x_1, x_{2p+1}) \\ & + \varepsilon \end{aligned} \tag{2.6}$$

where

$$\begin{cases} \alpha_1(x_1, \ldots, x_{p+1}) = f(x_1 + x_2 2^{-k} + \cdots + x_{p+1} 2^{-pk}), \\ \alpha_i(x_1, \ldots, x_{p-i+2}, x_{p+i}) = x_{p+i} 2^{-p-i+1} f'(x_1 + x_2 2^{-k} + \cdots + x_{p-i+2} 2^{(-p+i-1)k}), \\ \varepsilon \leq \left(\frac{1}{2} 2^{(-2p-2)k} + p 2^{(-2p-1)k}\right) \max f'' \\ \approx p 2^{(-2p-1)k} \max f''. \end{cases}$$

Too large values of p are unrealistic: performing many additions to avoid a few multiplications is not reasonable.

Let us try to calculate the amount of memory that would be required for implementing the double-precision sine function ($n = 53$), and $2p + 1 = 7$. We would choose $k = 8$, and perform the final summation with 55-bit accuracy. Hence the total amount of memory would be $(55 + 23 + 15) \times 2^{31} + 7 \times 2^5$ bits, that is, around 20 Gbytes, which is far from being feasible: the order-1 methods do not seem to be applicable beyond single-precision.

2.4. Comparison with the STAM Method

The Symmetric table addition method (STAM) was suggested by Schulte and Stine [5]. They split the binary representation of the input number x into m sub-words (whose sizes may be different), $x_1, x_2, \ldots, x_m$ and they use the order-1 Taylor expansion of f at the midpoint of the interval of numbers whose binary representation starts with x_1 and x_2. They expand the order-1 part of the Taylor approximation so that each term is a function of x_1 and a sub-word x_i only. Hence they approximate the result by the sum of $m - 1$ values, each of them being looked-up in a table that receives two sub-words as an input address.

At a first glance, if we examine the multipartite method and the STAM assuming in both cases a splitting into m sub-words, the STAM method requires smaller tables: the STAM method use 2-sub-word tables, and the multipartite method uses $p+1$-sub word tables.

And yet, assuming that the first two words have size n_0 and n_1, the order-1 Taylor approximation used by the STAM method has an error whose order of magnitude is

$$\frac{1}{2} 2^{-2(n_0+n_1+1)} f''(x)$$

therefore, to reach an error equal to approximately 2^{-n}, we need

$$n_0 + n_1 \geq \frac{n-3}{2}.$$

Hence, the first two sub-words must represent approximately half the global input-word. As a consequence, the tables will have around $n/2$ address bits.

Table 1 gives table sizes for various order-1 methods, assuming that we implement a 24-bit sine function. This table shows that both methods lead to a similar

Table 1. Table sizes for various order-1 methods, assuming that we implement a 24-bit sine function.

method	memory size (Kbytes)	error bound
bipartite (straightforward)	428	0.625×2^{-24}
bipartite (Schulte and Stine)	244	
tripartite (ours)	74	1.38×2^{-24}
splitting into 4 terms (Schulte and Stine)	92	
splitting into 5 terms (Schulte and Stine)	74.5	

amount of memory. This is not surprising, since both methods are very similar. Now, let us try to get smaller tables by using approximations of order larger than 1.

3. Higher-Order Methods

In the previous section, we have used order-1 Taylor expansions only. We also have seen that these methods are not applicable for precisions significantly larger than single precision. Now, let us give an example of the use of an order-2 expansion. As in Section 2.2, we split the input n-bit fixed-point number x into five k-bit parts $x_1, x_2, \ldots, x_5$. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k} + x_5 2^{-4k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.

$$\begin{aligned}
\alpha_1(x_1, x_2) &= f(x_1 + x_2 2^{-k}) - 3f(x_1) - \frac{1}{2}(2^{-3k} + 2^{-4k}) x_2^2 f''(x_1), \\
\alpha_2(x_1, x_3) &= f(x_1 + x_3 2^{-2k}) - \frac{1}{2} 2^{-3k} x_3^2 f''(x_1) - \frac{1}{6} 2^{-4k} x_3^3 f'''(x_1), \\
\alpha_3(x_1, x_4) &= f(x_1 + x_4 2^{-3k}) - \frac{1}{2} 2^{-4k} x_4^2 f''(x_1), \\
\alpha_4(x_1, x_5) &= f(x_1 + x_5 2^{-4k}), \\
u &= x_2 + x_3, \\
v &= x_2 - x_3, \\
w &= x_2 + x_4, \\
z &= x_2 - x_4, \\
\beta_1(u, x_1) &= \frac{1}{12} 2^{-4k} u^3 f'''(x_1) + \frac{1}{2} 2^{-3k} u^2 f''(x_1), \\
\beta_2(v, x_1) &= -\frac{1}{12} 2^{-4k} v^3 f'''(x_1) - \frac{1}{2} 2^{-3k} v^2 f''(x_1), \\
\beta_3(w, x_1) &= \frac{1}{4} w^2 f''(x_1),
\end{aligned} \tag{3.1}$$

$$\beta_4(z, x_1) = -\frac{1}{4}z^2 f''(x_1).$$

Then

$$f(x) \approx \alpha_1(x_1, x_2) + \alpha_2(x_1, x_3) + \alpha_3(x_1, x_4) + \alpha_4(x_1, x_5) + \beta_1(u, x_1) + \beta_2(v, x_1) + \beta_3(w, x_1) + \beta_4(z, x_1)$$

with an error of the order of 2^{-5k}. Hence, with this method, we can use tables with $2n/5$ address bits. Four additions are used to generate u, v, w and z, and after the table-lookups, 8 terms are added with a carry-save addition tree. This method would require around 20 Kbytes of table for single-precision. It would require around 100 Mbytes for double precision, which is still a lot.

Conclusion

Various table-based methods have been suggested during the last decade. When single-precision implementation is at stake, table-bound methods seem to be a good candidate for implementing fast functions. Unless there is a technology breakthrough, these methods are not suitable for double precision.

References

1. Ercegovac, M. D., Lang, T., Muller, J. M., and Tisserand, A.: *Reciprocation, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers*, Technical Report RR97-47, LIP, École Normale Supérieure de Lyon, November 1997, available at `ftp://ftp.lip.ens-lyon.fr/pub/Rapports/RR/RR97/RR97-47.ps.Z`.
2. Farmwald, P. M.: High Bandwidth Evaluation of Elementary Functions, in: Trivedi, K. S. and Atkins, D. E. (eds), *Proceedings of the 5th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society Press, Los Alamitos, CA, 1981.
3. Hassler, H. and Takagi, N.: Function Evaluation by Table Look-Up and Addition, in: Knowles, S. and McAllister, W. (eds), *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, July 1995. IEEE Computer Society Press, Los Alamitos, CA.
4. Das Sarma, D. and Matula. D. W.: Faithful Bipartite Rom Reciprocal Tables, in: Knowles, S. and McAllister, W. H. (eds), *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, 1995, IEEE Computer Society Press, Los Alamitos, CA, pp. 17–28.
5. Schulte, M. and Stine, J.: Accurate Function Approximation by Symmetric Table Lookup and Addition, in: *Proceedings of ASAP'97*, IEEE Computer Society Press, Los Alamitos, CA, 1997.
6. Schulte, M. and Stine, J.: Symmetric Bipartite Tables for Accurate Function Approximation, in: Lang, T., Muller, J. M., and Takagi, N. (eds), *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society Press, Los Alamitos, CA, 1997.
7. Schwarz. E.: *High-Radix Algorithms for High-Order Arithmetic Operations*, PhD thesis, Dept. of Electrical Engineering, Stanford University, 1992.
8. Tang, P. T. P.: Table Lookup Algorithms for Elementary Functions and Their Error Analysis, in: Kornerup, P. and Matula, D. W. (eds), *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, Grenoble, France, June 1991, IEEE Computer Society Press, Los Alamitos, CA, pp. 232–236.
9. Tang, P. T. P.: Table-Driven Implementation of the Expm1 Function in IEEE Floating-Point Arithmetic, *ACM Transactions on Mathematical Software* **18** (2) (1992), pp. 211–222.
10. Tang, P. T. P.: Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic, *ACM Transactions on Mathematical Software* **15** (2) (1989), pp. 144–157.

11. Tang, P. T. P.: Table-Driven Implementation of the Logarithm Function in IEEE Floating-Point Arithmetic, *ACM Transactions on Mathematical Software* **16** (4) (1990), pp. 378–400.
12. Wong, W. F. and Goto, E.: Fast Evaluation of the Elementary Functions in Single Precision, *IEEE Transactions on Computers* **44** (3) (1995), pp. 453–457.
13. Wong, W. F. and Goto, E.: Fast Hardware-Based Algorithms for Elementary Function Computations Using Rectangular Multipliers, *IEEE Transactions on Computers* **43** (3) (1994), pp. 278–294.