Table-Lookup Algorithms for Elementary Functions and Their Error Analysis

Ping Tak Peter Tang
Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Ave.
Argonne, IL 60439-4801

Abstract

Table-lookup algorithms for calculating elementary functions offer superior speed and accuracy when compared with more traditional algorithms. We show that, with careful design, it is feasible to implement table-lookup algorithms in hardware. Furthermore, we present a uniform approach to carry out a tight error analysis for such implementations.

1 Introduction

Since the adoption of IEEE Standard 754 for floating-point arithmetic [7], there has been a revival of interest in implementing elementary functions to near-perfect accuracy (see [12], [5], and [8], for example). A common thread of the recent work in this area is table-lookup algorithms. From a mathematical and algorithmic point of view, table-lookup algorithms are straightforward generalizations of more traditional algorithms such as those in [2] and [4]. According to W. Kahan, J. C. P. Miller had already pointed out in late 1950s that such approaches would be desirable as soon as memory became inexpensive. More recently, work by Gal ([3]) and the new runtime library by IBM ([1] and [6]) are notable in promoting table-lookup algorithms. Although these recent efforts use rather large tables and implement the functions in software, we illustrate here that, with careful design, the table sizes can be made so small that these table-lookup algorithms become easily realizable in hardware. Despite the size reduction, the speed and accuracy benefits brought about by table-lookup algorithms are preserved. Furthermore, we show that these table-lookup algorithms lend themselves to a uniform error analysis that yields tight error bounds. Typically, a theoretical bound of 0.57 unit in the last place (ulp) can be obtained for an implementation whose maximum error observed over several million arguments is 0.55 ulp.

The rest of the paper is organized as follows. Section 2 presents the general idea behind table-lookup algorithms. Section 3 illustrates the idea by three specific examples. Section 4 presents a uniform approach for a tight error analysis and an illustrative example. Section 5 discusses the feasibility of hardware implementations of table-lookup algorithms. Section 6 makes some concluding remarks on the advantages of table-lookup algorithms over CORDIC ([9]) and ordinary (without table-lookup) polynomial algorithms.

2 Table-Lookup Algorithms

Let f be the function to be implemented and I be the domain of interest. A typical table-lookup algorithm contains a set of "breakpoints" $c_k, k = 1, 2, \dots, N$ in I and a table of N approximations T_k to $f(c_k)$. For any input argument $x \in I$, the algorithm calculates $f(x)$ in three steps.

Reduction: For this given x, the algorithm selects an appropriate breakpoint c_k. (In general, c_k is the breakpoint closest to x.) It then applies a "reduction transformation"

$$r = \mathcal{R}(x, c_k).$$

A typical case is $\mathcal{R}(x, c_k) = x - c_k$.

Approximation: The algorithm now calculates $f(r)$ using some approximation formula

$$p(r) \approx f(r).$$

Very often, p is a polynomial.

Reconstruction: Based on the reduction transformation $\mathcal{R}$, the values $f(c_k)$ and $f(r)$, and the nature of f, the algorithm calculates $f(x)$ by a reconstruction formula $\mathcal{S}$:

$$\begin{aligned} f(x) &= \mathcal{S}(f(c_k), f(r)) \\ &\approx \mathcal{S}(f(c_k), p(r)) \\ &\approx \mathcal{S}(T_k, p(r)). \end{aligned}$$

Although a traditional polynomial or rational-function-based algorithm (see [2] and [4]) can also be expressed in these three steps, there are two properties peculiar to a table-lookup algorithm. First, the reduction process in a table-lookup algorithm is much more flexible since, unlike a traditional algorithm, the choice of breakpoints is basically independent of the function f in question. In most situations, the breakpoints are chosen so that the reduced argument r can be computed efficiently on the particular machine in question. Second, in a table-lookup algorithm, the magnitude of the reduced argument r can be made as small as one wishes, limited only by the table size one can accommodate. We now illustrate these ideas by three realistic examples.

3 Algorithms for 2^x, $\log x$, and $\sin x$

The functions 2^x, $\log x$, and $\sin x$ are among the most commonly used elementary functions. We have purposely chosen different bases for the exponential and logarithm functions (base 2 and e, respectively) to illustrate the flexibility of table-lookup algorithms.

The algorithms here calculate the functions on "primary" domains. Transformations of arguments to such domains are standard and well known (see [2] and [4]).

3.1 2^x on [-1,1]

Reduction: Find the breakpoint $c_k = k/32, k = 0, 1, \ldots, 31$ such that

$$|x - (m + c_k)| \leq 1/64,$$

where $m = -1, 0$, or 1. Calculate r by $r = [x - (m + c_k)] \cdot (\log 2)$. Note that $|r| \leq \log 2/64$.

Approximation: Approximate $e^r - 1$ by a polynomial $p(r)$,

$$p(r) = r + p_1 r^2 + \cdots + p_n r^{n+1}.$$

The coefficients $p_1, p_2, \ldots, p_n$ can be determined by applying the Remes algorithm [14] to the function $e^r - 1$ on the interval $[-\log 2/64, \log 2/64]$.

Reconstruction: Reconstruct 2^x by the relationships

$$\begin{aligned} 2^x &= 2^{m+c_k} \cdot e^r \\ &= 2^m\{2^{c_k} + 2^{c_k}(e^r - 1)\} \\ &\approx 2^m\{2^{c_k} + 2^{c_k} \cdot p(r)\} \\ &\approx 2^m\{T_k + T_k \cdot p(r)\}. \end{aligned}$$

$T_k \approx 2^{c_k}$, $k = 0, 1, \ldots, 31$, are the tabulated values.

3.2 $\log x$ on [1,2]

Reduction: If $x < e^{1/16}$, compute $\log(x)$ by a simple polynomial approximation and exit. This polynomial is designed to approximate $\log(x)$ on $[1, e^{1/16}]$ (see [13] for details). Otherwise, find the breakpoint $c_k = 1 + k/64, k = 0, 1, \ldots, 64$ such that

$$|x - c_k| \leq 1/128.$$

Calculate r by $r = 2(x - c_k)/(x + c_k)$. Note that $|r| \leq 1/128$.

Approximation: Approximate $\log(x/c_k)$ by an odd polynomial $p(r)$:

$$p(r) = r + p_1 r^3 + p_2 r^5 + \cdots + p_n r^{2n+1}.$$

The coefficients $p_1, p_2, \ldots, p_n$ can be determined by applying the Remes algorithm to the function $\log([1 + r/2]/[1 - r/2])$ on the interval $[0, 1/128]$ using odd polynomials. Note that

$$\begin{aligned} \log\left(\frac{x}{c_k}\right) &= \log\left(\frac{1+\frac{r}{2}}{1-\frac{r}{2}}\right) \\ &= 2\left(\left(\frac{r}{2}\right) + \frac{1}{3}\left(\frac{r}{2}\right)^3 + \frac{1}{5}\left(\frac{r}{2}\right)^5 + \cdots\right). \end{aligned}$$

Reconstruction: Reconstruct $\log x$ by the relationships

$$\begin{aligned} \log(x) &= \log(c_k) + \log(x/c_k) \\ &\approx \log(c_k) + p(r) \\ &\approx T_k + p(r). \end{aligned}$$

$T_k \approx \log(c_k)$, $k = 0, 1, \ldots, 64$, are the tabulated values.

3.3 $\sin x$ on $[0, \pi/4]$

Reduction: If $|x| < 1/16$, calculate $\sin x$ by a simple polynomial approximation. Otherwise, find the breakpoint c_{jk} of the form

$$c_{jk} = 2^{-j}(1+k/8) \qquad j = 1, 2, 3, 4; \quad k = 0, 1, \ldots, 7$$

that is closest to x. Calculate r by $r = x - c_{jk}$. Note that $|r| \leq 1/32$.

Approximation: Approximate $\sin r - r$ and $\cos r - 1$ by polynomials p and q, respectively:

$$\begin{aligned} p(r) &= p_1 r^3 + p_2 r^5 + \cdots + p_n r^{2n+1}, \\ q(r) &= q_1 r^2 + q_2 r^4 + \cdots + q_m r^{2m}. \end{aligned}$$

The coefficients $p_1, p_2, \ldots, p_n; q_1, q_2, \ldots, q_m$ can be determined by applying the Remes algorithm to the functions $\sin r - r$ and $\cos r - 1$ on the interval $[0, 1/32]$ using odd and even polynomials, respectively.

Reconstruction: Reconstruct $\sin(x)$ by the relationships

$$\begin{aligned} \sin(x) &= \sin(c_{jk} + r) \\ &= \sin(c_{jk}) \cos r + \cos(c_{jk}) \sin r \\ &\approx \sin(c_{jk}) + r \cos(c_{jk}) \\ &\quad + (\sin(c_{jk}) q(r) + \cos(c_{jk}) p(r)) \\ &\approx S_{jk} + r C_{jk} + (S_{jk} q(r) + C_{jk} p(r)). \end{aligned}$$

Table 1: A Realistic Set of Algorithms for IEEE Double Precision

Function	No. of Table Entries	No. of Coefficients
2^x	32	$n = 5$
$\log x$	64	$n = 2$
$\sin x$	64	$n = 3, m = 3$

To conclude this section, we tabulate in Table 1 the table size and coefficient requirements for a realistic set of algorithms tailored to IEEE double precision. Clearly, this is one particular choice out of the many possible combinations of breakpoints and approximating polynomials. Take 2^x, for example: one can reduce the table size to 16 by increasing n to 6 (hence the price of an add and multiply). Optimal choice is an engineering issue whose answer depends on the particular project in question.

4 Error Analysis

Recall the three steps of calculating f at x:

Reduction: $r = \mathcal{R}(x, c_k)$.

Approximation: $p(r) \approx f(r)$

Reconstruction: $f(x) = \mathcal{S}(f(c_k), f(r)) \approx \mathcal{S}(T_k, p(r))$

Because of inexact computations, we obtain $\hat{r}$ instead of r, $\hat{p}$ instead of p, and $\hat{\mathcal{S}}$ instead of $\mathcal{S}$. Hence, the computed result is

$$\hat{\mathcal{S}}(T_k, \hat{p}(\hat{r})).$$

The goal of the error analysis is to estimate accurately the difference

$$|\mathcal{S}(f(c_k), f(r)) - \hat{\mathcal{S}}(T_k, \hat{p}(\hat{r}))|.$$

We apply the triangular inequality. Thus,

$$\begin{aligned} &|\mathcal{S}(f(c_k), f(r)) - \hat{\mathcal{S}}(T_k, \hat{p}(\hat{r}))| \\ \leq\ & |\mathcal{S}(f(c_k), f(r)) - \mathcal{S}(f(c_k), f(\hat{r}))| + \\ & |\mathcal{S}(f(c_k), f(\hat{r})) - \mathcal{S}(f(c_k), p(\hat{r}))| + \\ & |\mathcal{S}(f(c_k), p(\hat{r})) - \hat{\mathcal{S}}(T_k, \hat{p}(\hat{r}))| \\ \leq\ & E_1 + E_2 + E_3. \end{aligned}$$

In most situations,

$$\begin{aligned} E_1 &\leq \text{constant} \cdot |f(r) - f(\hat{r})| \\ &\approx \text{constant} \cdot |f'(r)| \cdot |r - \hat{r}| \end{aligned}$$

and

$$E_2 \leq \text{constant} \cdot \max_t |f(t) - p(t)|.$$

E_1 can be easily estimated because the reduction process $\mathcal{R}$ is usually so simple that $|r - \hat{r}|$ can be estimated tightly. E_2 can also be easily estimated since the numerical value

$$\max_t |f(t) - p(t)|$$

is obtained when the polynomial is sought, usually by the use of the Remes algorithm. The maximum is taken over the domain of approximation.

Let us make a short digression for those interested in our implementation of the Remes algorithm. The version of Remes algorithm we use is the one-point exchange algorithm (see [10] for example) as opposed to a multiple-exchange algorithm ([14]). Moreover, the exchange scheme is based on the simplex algorithm which is a generalization of that based on sign alternations (see [11] for details). These specifics allow the algorithm to handle non-Haar systems as well as approximation of discontinuous functions. Moreover, the domain does not need to be discretized, because

$$E_p = \max_t |f(t) - p(t)|$$

is calculated by searching the zeros of $f'(t) - p'(t)$ by numerical root finder. Consequently, the numerical value of E_p we obtain is accurate to many digits. This accuracy is definitely sufficient for the purpose of error analysis, because most of the time we need only one or two digits of E_p (see below).

We return to the discussion on errors. The rounding error

$$E_3 = |\mathcal{S}(f(c_k), p(\hat{r})) - \hat{\mathcal{S}}(T_k, \hat{p}(\hat{r}))|$$

in calculating the polynomial and reconstruction is usually the most difficult to estimate tightly. With the use of table-lookup algorithms, however, the analysis is greatly simplified. The reason is that the magnitude of the reduced argument r (or $\hat{r}$) is typically so small that rounding errors associated with r^k, $k \geq 2$, are practically zero. The simplicity of the analysis in [12] and [13] illustrates the situation.

To illustrate the ideas here, we carry out the analysis of 2^x for a typical IEEE double-precision implementation. Let ϵ denote 1 ulp of 1, i.e., $\epsilon = 2^{-52}$. Clearly, the subtraction $s := x - (m + c_k)$ is exact. Hence the errors in the reduction step are those caused by the multiplication by $\log 2$ and the fact that the value "log 2" used is only a 53-bit approximation:

$$\begin{aligned} r &= s \log 2, \quad \text{and} \\ \hat{r} &= s(\log 2 + \delta_1) + \delta_2, \end{aligned}$$

where δ_1 is the error in the approximation to $\log 2$, and δ_2 is the error caused by the finite-precision product. Now,

$$\log 2 = .69314718 \ldots \in 2^{-1} \cdot [1, 2).$$

Therefore, rounding (as opposed to truncating) $\log 2$ to working precision (53 bits in our case) gives an error no bigger than $\frac{1}{2}\frac{1}{2}\epsilon = 2^{-2}\epsilon$. Hence $|\delta_1| \leq 2^{-2}\epsilon$. Next,

$$|s \cdot (\log 2 + \delta_1)| \leq \log 2/64 + \delta_1/64.$$

Since $\log 2/64+\delta_1/64 \in 2^{-7}\cdot[1,2)$, rounding the product $s(\log 2/64 + \delta_1)$ to working precision gives an error no bigger than $2^{-8}\epsilon$. Hence, $|\delta_2| \leq 2^{-8}\epsilon$. Therefore,

$$|r - \hat{r}| \leq |s\delta_1| + |\delta_2| \leq 2^{-7}\epsilon.$$

Hence,

$$|E_1| \leq |\frac{d}{dt}e^t||r - \hat{r}| \leq 1.02 \times 2^{-7}\epsilon.$$

Next, the best approximating polynomial p obtained by a Remes algorithm gives

$$|(e^t - 1) - p(t)| \leq 2^{-63} = 2^{-11}\epsilon, \quad |t| \leq \log 2/64.$$

Thus,

$$|E_2| \leq 2^{-11}\epsilon.$$

Finally, we estimate the errors in computing $p(\hat{r})$ and the final reconstruction. Since 2^{c_k} is not representable in 53 bits, we use 2 variables T_1 and T_2 where T_1 is 2^{c_k} rounded to 53 bits and T_2 is a correction term that makes $T_1 + T_2 = 2^{c_k}$ for all practical purposes. (Note that a T_2 having 6 significant bits will be sufficient.) The reconstruction is

$$2^m(T_1 + (T_1 * p + T_2)).$$

Scaling by 2^m is exact. The last add contributes no more than 0.5 ulp of error. Estimating the error in $T_1 * p + T_2$ is simple: Since $|\hat{r}^2| < 2^{-12}$, the only significant error in calculating p is the last add. Thus the computed result is

$$\begin{aligned}&(2^{c_k} + \delta_1)(p + \delta_2) + T_2 + \delta_3 \\ &= \quad 2^{c_k}p + T_2 + \delta_1 p + \delta_2 2^{c_k} + \delta_3,\end{aligned}$$

where $|p| \leq 2^{-6}$, $|\delta_1| \leq 2^{-1}\epsilon$, $|\delta_2| \leq 2^{-7}\epsilon$, and $|\delta_3| \leq 2^{-6}\epsilon$.

The rounding error is bounded by

$$\begin{aligned}|E_3| &\leq \frac{1}{2}\,\text{ulp} + (2^{-7} + 2^{-6} + 2^{-6})\epsilon \cdot 2^m \\ &\leq \frac{1}{2}\,\text{ulp} + (2^{-7} + 2^{-5})\,\text{ulp} \\ &\leq 0.54\,\text{ulp}.\end{aligned}$$

Consequently,

$$|E_1| + |E_2| + |E_3| \leq 0.556\,\text{ulp}.$$

Note that in the case $c_k = 0$ (in particular, $|x| \leq 1/64$), the bound given here is pesimistic because no error will be committed in $T_1 * p + T_2$, for $T_1 = 1$ and $T_2 = 0$.

5 Feasibility of Hardware Implementations

Table-lookup algorithms have been implemented successfully in software ([6] and [12], for example). With today's VLSI technology and the algorithms' flexibility, hardware implementations also seem extremely feasible. In what follows, we reexamine the algorithm for 2^x in Section 3.1 from the point of view of hardware implementation.

Table Size: The table required here has 32 double-precision values. Assuming 8 bytes of storage per entry (although one can device more compact storage schemes), this table is only 2 kbits. Such a table is easily accommodated by today's standard.

Reduction and Table Lookup: Consider the reduction for $|x| \geq 1/16$ where x is an IEEE double-precision number. Thus

$$x = (-1)^s \cdot 2^k \cdot (1.b_1 b_2 \ldots b_{52})_{\text{base2}},$$

where $s \in \{0,1\}, k \in \{-1,-2\}$. Now, mathematically,

$$m + c_k = \text{round-to-integer}(32x)/32,$$

and both m and c_k are determined by the seven bits of information s, $\text{lsb}(k)$, and $\{b_1, b_2, \ldots, b_5\}$. Clearly, simple multiplexers can be constructed to deliver both $m + c_k$ and T_k.

Approximation and Reconstruction: Most hardware performs IEEE sums and products via an internal data format that has a longer mantissa and a wider exponent field than the working precision in question (say double precision or double-extended precision) in question. The longer mantissa typically contains the guard, round, and sticky bits that ensure correct IEEE rounding; and the internal exponent field typically contains two extra bits designed to accommodate products of working-precision values of extreme magnitudes, say, two smallest denormalized numbers. Rounding and exception handling are usually performed separately during conversion of internal formats back to the working precision.

It is not hard to see that the approximation and reconstruction steps in Section 3.1 can be carried out solely in the internal format. First, provided the internal exponent field has two extra bits, the recurrence

$$\begin{aligned}p(r) &= r + p_1 r^2 + \cdots + p_5 r^6 \\ &= r + r \cdot r \cdot (p_1 + r \cdot (p_2 + \cdots + r \cdot p_5)\cdots)\end{aligned}$$

and the reconstruction

$$2^m(T_k + T_k \cdot p(r))$$

where $|r| \leq \log 2/64$ will not overflow or underflow. (Note that the coefficients $p_j \approx 1/j!$ are moderate in magnitude.) Second, note that there is no need to round any of the intermediate results to working precision. In fact, keeping the extra guard and round bits would enhance the accuracy. Performing these two steps without the need of rounding or error checking will clearly save a good amount of time compared to full IEEE operations everywhere.

6 Concluding Remarks

Table-lookup algorithms offer several advantages over traditional polynomial/rational-funtion algorithms and CORDIC algorithms. In comparison, a table-lookup algorithm is generally

1. faster because it requires less work in the approximation steps,
2. more accurate because the rounding error made in the approximation step is tiny, and
3. amenable to tight error analysis.

CORDIC algorithms have been attractive from a hardware point of view because they require only shifts and adds (see [9] for example). The price one has to pay for this simplicity is a relatively large number of iterations of pseudo-division (reduction) and pseudo-multiplication (reconstruction). Advances in VLSI technology have now made it possible to realize basic operations such as primitive floating-point adds and multiplies (that is, without rounding or exception handling) on the order of a few clocks on modern hardware. In such hardware environments, table-lookup algorithms are extremely feasible alternatives to CORDIC.

Acknowledgement

This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] R. C. Agarwal, J. W. Cooley, F. G. Gustavson, J. B. Shearer, G. Slishman, and B. Tuckerman, New scalar and vector elementary functions for the IBM System/370, *IBM Journal of Research and Development,* 30, no. 2, March 1986, pp. 126–144.

[2] W. Cody and W. Waite, *Software Manual for the Elementary Functions,* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[3] S. Gal, Computing elementary functions: A new approach for achieving high accuracy and good performance, in *Accurate Scientific Computations,* Lecture Notes in Computer Science, Vol. 235, Springer, New York, 1985, pp. 1–16.

[4] J. F. Hart et al., *Computer Approximations,* John Wiley and Sons, New York, 1968.

[5] D. Hough, Elementary functions based upon IEEE arithmetic, *Mini/Micro West Conference Record,* Electronic Conventions, Inc., Los Angeles, 1983.

[6] IBM Elementary Math Library, *Programming RPQ P81005, Program number 5799-BTB, Program Reference and Operations Manual,* SH20-2230-1, August 1984.

[7] IEEE standard for binary floating-point arithmetic, *ANSI/IEEE Standard 754-1985,* Institute of Electrical and Electronic Engineers, New York, 1985.

[8] P. W. Markstein, Computation of elementary functions on the IBM RISC System/6000 processor, *IBM Journal of Research and Development,* 34, no. 1, January 1990, pp. 111–119.

[9] R. Nave, Implementation of transcendental functions on a numeric processor, *Microprocessing and Microprogramming,* 11, 1983, pp. 221–225.

[10] M. J. D. Powell, *Approximation Theory and Methods,* Cambridge University Press, Cambridge, 1981.

[11] P. T. P. Tang, A fast algorithm for linear complex Chebyshev approximations, *Mathematics of Computation,* 51, no. 184, October 1988, pp. 721–739.

[12] P. T. P. Tang, Table-driven implementation of the exponential function in IEEE floating-point arithmetic, *ACM Transactions on Mathematical Software,* 15, no. 2, June 1989, pp. 144–157.

[13] P. T. P Tang, Table-driven implementation of the logarithm function in IEEE floating-point arithmetic, *ACM Transactions on Mathematical Software,* 16, no. 2, December 1990, pp. 378-400.

[14] L. Veidinger, On the numerical determination of the best approximation in the Chebyshev sense, *Numerische Mathematik,* 2, 1960, pp. 99–105.