CONF-9106103--1

# 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

CONF-9106103--1

DE91 006047



**Abstract**

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

## 1 Introduction

Since the adoption of IEEE Standard 754 for floating-point arithmetic [4], there has been a revival of interest in implementing elementary functions to near perfect accuracy (see [6], [3], and [5], for example). A common thread of the recent works in this area is table-lookup algorithms. Although these recent works 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 also show that these table-lookup algorithms render themselves to a uniform error analysis that yield tight error bounds. It is quite typical that a theoretical bound of 0.57 units in the last place (ulp) can be obtained for an implementation of 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.

*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.

Section 4 presents a uniform approach for tight error analysis and an illustrative example. Section 5 makes some concluding remarks on the advantages of table-lookup algorithms over CORDIC 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 selects a set of "breakpoints" $c_j, j = 1, 2, \ldots, N$, in $I$ and tabulates the $N$ values of $f(c_j)$. 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" $\mathcal{R}$ to obtain a reduced argument

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

$\mathcal{R}$ is always simple; 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 formula $\mathcal{R}$ and the values $f(c_k)$ and $f(r)$, the algorithm calculates $f(x)$ by a reconstruction formula $\mathcal{S}$

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

The formula $\mathcal{S}$ is highly dependent on the function $f$.

Although a traditional polynomial or rational-function based algorithm ([1], [2]) 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 in 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 [1] and [2]).

## 3.1 $2^x$ on [-1,1]

**Reduction:** If $|x| < 1/16$, calculate $2^x$ by a simple polynomial approximation. Otherwise, find the breakpoint $c_k = k/32, k = 0, 1, \ldots, 31$ such that

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

where $m = -1$ or 0. Calculate $r$ by $r = [x - (m + c_k)] \cdot (\log 2)$.

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

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

**Reconstruction:** Reconstruct $2^x$ by the relationships

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

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

**Reduction:** If $x < e^{1/16}$, compute $\log(x)$ by a simple polynomial approximation. 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 $r = 2(x - c_k)/(x + c_k)$.

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

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

Note that

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

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 | $n = 3, m = 3$ |

**Reconstruction:** Reconstruct $\log x$ by the relationships

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

### 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 $r = x - r$.

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

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

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

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

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.

## 4 Error Analysis

Recall the three steps of calculating $f$ at $x$:

**Reduction:** $r = \mathcal{R}(x)$ (We omit the "$c_k$" for simplicity's sake.)

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

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

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

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

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

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

We apply the triangular inequality. Thus,

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

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 Remez algorithm. (The maximum is taken over the domain of approximation.) The rounding error

$$E_3 = |\mathcal{S}(p(\hat{r})) - \hat{\mathcal{S}}(\hat{p}(\hat{r}))|$$

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 [6] and [7] illustrates the situation.

To illustrate the ideas here, we carry out the analysis of $2^x$ for a typical IEEE double-precision implementation. Let $\epsilon$ denote 1 ulp of 1, i.e., $\epsilon = 2^{-52}$. Clearly, the subtraction

$x - (m + c_k)$ is exact. Hence the errors in the reduction step are the caused by the multiplication by $\log 2$ and that the "$\log 2$" 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 $|\delta_1| \leq 2^{-2}\epsilon$ and $|\delta_2| \leq 2^{-8}\epsilon$. Thus,

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

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 Remez 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 contribute no more than $\frac{1}{2}$ ulp of error. 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

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

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

Thus 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,

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

# 5 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 rounding error made in the approximation step is tiny, and
3. amenable to tight error analysis.

Since the table size required can be made moderate and that basic operations such as primitive floating-point adds and multiplys (that is, without exception handlings) can be easily realized in the order of a few clocks on modern hardware, hardware implementations of table-lookup algorithms is extremely feasible.

# References


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

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

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

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

[5] 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.

[6] P. T. P. Tang, Table-driven implementation of the exponential function in IEEE floating-point arithmetic, *ACM Transactions on Mathematical Software*, 16, no. 2, June 1989, pp. 144–157.

[7] P. T. P Tang, Table-driven implementation of the logarithm function in IEEE floating-point arithmetic, Preprint MCS-P55-0289, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., February 1989 (to appear in *ACM Transactions on Mathematical Software*).

# END

# DATE FILMED

05 / 09 / 91