Practically Fast Multiple-Precision Evaluation of LOG(X)

TATEAKI. SASAKI* and YASUMASA KANADA**

A new algorithm for multiple-precision evaluation of log (x) is presented. The algorithm is based on the well-known q-expansion formulas for elliptic theta functions and the famous arithmetic-geometric mean of Gauss. The algorithm is a generalization of the Salamin-Brent algorithm based on the arithmetic-geometric mean. The efficiency of the new algorithm is shown by numerical experiments.

1. Introduction

This paper describes a new algorithm for multiple-precision evaluation of $\log(x)$ with a real argument x. If the relative error of a number is of $O(2^{-p})$ with p a positive integer, we say the number is of precision p. The algorithm to be described in this paper requires $O(M(p)\log(p))+O(M(p)\sqrt{p})$ single-precision operations for evaluating $\log(x)$ to precision p, where $M(p)$ is the number of single-precision operations necessary for multiplying two numbers of precision p. The time complexity of the algorithm can be reduced to $O(M(p)\log(p))$ by a simple argument reduction. The algorithm is based on the well-known q-expansion formulas for elliptic theta functions [8, p. 229] and the famous arithmetic-geometric mean of Gauss [7, p. 352].

The problem of evaluating elementary functions to any given precision has been investigated by many authors. Among others, Brent [3] presented many algorithms having the time complexity $O(M(p)\log(p))$ for $\exp(x)$, $\log(x)$, $\tan(x)$, $\operatorname{atan}(x)$, etc. In his method, $\log(x)$ is evaluated by first solving a nonlinear equation $T(m)=x$ by Newton's method to get $m=T^{-1}(x)$ and then calculating $U(m)$. Functions T and U are calculated by quadratically converging iterative algorithms, which require about $3\cdot\log_2(p)$ and $2\cdot\log_2(p)$ square root evaluations, respectively. Numerical experiments show that these square root evaluations require relatively large overhead unless p is quite large and that the $O(M(p)p)$ Taylor series method which sums many terms of the Taylor series for $\log(1+x)$ turns out to be faster than the Brent's algorithm when p is less than several thousands.

A much better algorithm of time complexity $O(M(p)\log(p))$, which was proposed by Salamin [see, 1] and by Brent [4], adjusts the argument x so that $x>4\cdot 2^{p/2}$ and applies the arithmetic-geometric mean iteration. This algorithm requires about $2\cdot\log_2(p)$ square root evaluations, and it is the fastest at present for large p. The algorithm we propose in this paper is a generalization of the Salamin-Brent algorithm, and it improves their algorithm for p in an intermediate range.

Let us next comment on the Schwab-Borchardt algorithm [2, 12]. This algorithm uses a simple iteration relation which is a kind of arithmetic-geometric mean iteration. Gauss suggested this relation to Pfaff in his letter in 1800, and Pfaff promptly determined the limit of the iteration. Pfaff's result was, however, not published and the algorithm was rediscovered by Schwab in ~1812 and by Borchardt in 1880. The Schwab-Borchardt algorithm was seldom used because the error reduces by a factor of only 4 after each iteration. In 1961, Thacher [13] improved the error reduction factor up to 16, and in 1972, Carlson [6] improved the algorithm drastically so that the error becomes $O(2^{-n^2-n})$ after the nth iteration. This algorithm has, therefore, the time complexity $O(M(p)\sqrt{p})$. In this algorithm, however, about $\sqrt{p}$ square root evaluations are necessary, which requires quite large overhead in actual computation.

In our algorithm, only about $\log_2(p)$ square root evaluations are necessary and the $\sqrt{p}$ dependence of the time complexity is due to the evaluation of a power series of the form $\sum x^{n^2}$. An actual implementation of our algorithm revealed that the evaluation of this power series was quite fast for precision in an intermediate range, such as $p \lesssim 10^3 \sim 10^4$. For very large p, we can reduce the computing time of the power series by a simple argument reduction, and the algorithm reaches the Salamin-Brent algorithm.

2. Elliptic Theta Functions and the Arithmetic-Geometric Mean

We first summarize the necessary formulas from the theory of elliptic functions. The first kind complete elliptic integrals K and K' for the modulus k are defined by [8, Ch. 9]

$$K \equiv K(k)=\int_0^{\pi/2}\frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}, \tag{1}$$

$$K' \equiv K(k'),\quad k'=\sqrt{1-k^2}. \tag{2}$$

We use the well-known q-expansion formulas for the elliptic theta functions $\theta_0(u)$ and $\theta_3(u)$ [8, Ch. 10]. We need only the values at $u=0$:

*The Institute of Physical and Chemical Research Wako-shi, Saitama 351, Japan.

**Computer Center, The University of Tokyo Bunkyo-ku, Tokyo 113, Japan.

$$\theta_0^0 \equiv \theta_0(0) = 1 + 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}, \tag{3}$$

$$\theta_3^0 \equiv \theta_3(0) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}. \tag{4}$$

If $|q| \ll 1$ these series converge very fast. The complete elliptic integrals K and K' are related to q by the formula [8, Ch. 18]

$$q = \exp(-\pi K'/K) \quad \text{or} \quad \pi K'/K = \log(1/q). \tag{5}$$

Furthermore, k (or k') and K can be expressed by θ_0^0 and θ_3^0 as

$$k = \sqrt{1-(\theta_0^0/\theta_3^0)^4} \quad (\text{or } k' = (\theta_0^0/\theta_3^0)^2), \tag{6}$$

$$K = \pi(\theta_3^0)^2/2. \tag{7}$$

Another formula we need is the famous arithmetic-geometric mean of Gauss. Let $a_0 = 1$ and $b_0 = k$, $0 < k < 1$, and calculate the nth terms a_n and b_n by the formula

$$a_n = (a_{n-1} + b_{n-1})/2, \quad b_n = \sqrt{a_{n-1}b_{n-1}}, \quad n \geqq 1. \tag{8}$$

Then, the series $\{a_n\}$ and $\{b_n\}$ converge quadratically to a common limit. The limit is called the arithmetic-geometric mean and given by [7, pp. 352–355 or 5]

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = \pi/2K'. \tag{9}$$

3. Algorithm

Suppose we are given q such that $0 < q \ll 1$. Then, using (3) and (4), we can evaluate θ_0^0 and θ_3^0 to precision $p+g$, where g is a guard precision. Using (6) and (7), we can evaluate k and K to precision $p+g$. We cannot directly evaluate K' from θ_0^0 and θ_3^0, but we can evaluate K' by the arithmetic-geometric mean iteration. Then, according to the formula (5), the ratio $\pi K'$ to K gives $\log(1/q)$. Note that, since $q^{(n+1)^2} = q^{n^2}q^{2n+1}$, we can evaluate θ_0^0 and θ_3^0 by about $2\sqrt{p}$ multiplications and $2\sqrt{p}$ additions of multiple-precision numbers. From these arguments, it is clear that the following algorithm evaluates $\log(1/q)$ to precision p:

Algorithm for log(1/*q*), $0 < q \ll 1$.

```
    $SIGN \leftarrow -1$;
    $T0 \leftarrow T3 \leftarrow QNN \leftarrow Q \leftarrow q$;
    $Q2 \leftarrow Q \cdot Q$; $Q2N1 \leftarrow Q2 \cdot Q$;
    while $(QNN \leftarrow QNN \cdot Q2N1) > 2^{-p}$ do
        begin $T0 \leftarrow T0 + SIGN \cdot QNN$;
              $T3 \leftarrow T3 + QNN$;
              $SIGN \leftarrow -SIGN$;
              $Q2N1 \leftarrow Q2N1 \cdot Q2$;
        end;
    $T0 \leftarrow 1 - 2 \cdot T0$; $T3 \leftarrow 1 + 2 \cdot T3$;
    $K \leftarrow (1 - (T0/T3)^4)^{1/2}$;
    $A \leftarrow 1$; $B \leftarrow K$;
    while $(A - B) > 2^{-p/2}$ do
        begin $Q \leftarrow (A+B)/2$;
              $B \leftarrow (AB)^{1/2}$;
              $A \leftarrow Q$;
        end;
    $A \leftarrow (A+B)/2$;
    return $\pi/(A \cdot T3^2)$.
```

The above algorithm is for argument q such that $0 < q \ll 1$. As we will explain in Sec. 4, a reasonable range of q is $0.01 \lesssim q \lesssim 0.1$. Given an argument x for logarithm, we can reduce the argument by dividing (or multiplying) an integral power of 2. In our current program, the argument q is determined as

$$0.005 < q \equiv x/2^j \leqq 0.01, \quad j \text{ is an integer.}$$

Then, we can calculate $\log(x)$ by the formula

$$\log(x) = \log(q) + j \cdot \log(2),$$

with a precomputed value of $\log(2)$.

4. Efficiency of the Algorithm

Let us briefly discuss the time complexity of our algorithm. Since the above algorithm is for the argument q such that $0 < q \ll 1$, the first step of evaluation of $\log(x)$, $x > 0$, is to reduce the argument so that x becomes much smaller than 1. This argument reduction will make the q-expansion series for θ_0^0 and θ_3^0 converge fast. If, however, x becomes very small then k also becomes small and the arithmetic-geometric mean iteration converges slowly. Therefore, a reasonable range into which the argument is reduced is $0.01 \lesssim x \lesssim 0.1$. This argument reduction can be done in $O(M(p))$ operations.

The best known method of evaluating the square root of x to high precision is to use Newton's iteration such as

$$y_{n+1} = \frac{1}{2}(y_n + x/y_n). \tag{10}$$

Using this iteration and controlling the precision of y_n so that only the significant digits of y_n are calculated, we can evaluate the square root in $O(M(p))$ operations [see 4 in details].

The arithmetic-geometric mean iteration converges quadratically:

$$a_{n+1} - b_{n+1} = \frac{\{(a_n+b_n)/2\}^2 - a_nb_n}{(a_n+b_n)/2 + \sqrt{a_nb_n}} \simeq \frac{(a_n-b_n)^2}{8a_{n+1}}.$$

Hence, the evaluation of the arithmetic-geometric mean to precision p requires $O(\log_2(p))$ iterations or $O(M(p)\log_2(p))$ operations.

With the argument reduction mentioned above, the evaluation of θ_0^0 and θ_3^0 requires $O(M(p)\sqrt{p})$ operations. Therefore, the time complexity of our algorithm is $O(M(p)\log(p)) + O(M(p)\sqrt{p})$.

Let us next consider the case where the argument q is reduced to an extremely small value. If $0 < q \simeq 0$, formulas (3) and (4) give

$$\theta_0^0 = 1 - 2q + O(q^4),$$
$$\theta_3^0 = 1 + 2q + O(q^4).$$

Then, formulas (6) and (7) tell us that

$$k=4\sqrt{q}(1-4q+0(q^2)),$$
$$K=(\pi/2)(1+4q+0(q^2)).$$

Therefore, if we reduce the argument q so that $4\sqrt{q}<2^{-p/2}$ and calculate the arithmetic-geometric mean a_∞ with the initial values $a_0=1$ and $b_0=k=4\sqrt{q}$, then we have

$$\log(1/q)=\pi K'/K=\frac{\pi}{a_\infty}(1+O(2^{-p})), \qquad (11)$$

or

$$K'=\frac{\pi}{2a_\infty}=(1+O(2^{-p}))\log(1/q). \qquad (12)$$

The relation (12) is nothing but the one on which the Salamin-Brent algorithm for $\log(x)$ is based. Therefore, our algorithm is a generalization of the Salamin-Brent algorithm. It is easy to see that, if we choose the value of b_0 as small as $2^{-p/2}$, about $2\cdot\log_2(p)$ iterations are necessary to calculate the arithmetic-geometric mean with the accuracy $O(2^{-p})$. That is, the time complexity of our algorithm becomes $O(M(p)\log(p))$ by a simple argument reduction.

5. Numerical Tests and Discussions

In this section, we define the precision of a number N by the number of significant decimal digits of N and denote it by P. The relation between p and P is $P=p/\log_2(10)$.

We have implemented our algorithm described in Sec. 3 on our arbitrary precision real arithmetic system [11] written in LISP. The reason of using a LISP-based system is that its facility of detailed precision handling allows us to write an efficient program [9]. We have also programmed four other algorithms for $\log(x)$ and compared them with our algorithm. Table 1 shows the results, where the elliptic method uses our algorithm with $q\sim0.01$, Taylor's method 1 sums terms of the Taylor series for $\log(1+x)$ after reducing the argument into the range $|x|<0.05$, Taylor's method 2 uses the Taylor series for $\log(1+x)/(1-x)$ with $|x|<0.05$, Newton's method solves the nonlinear equation $x=\exp(y)$ by Newton's iteration method to get $y=\log(x)$, and the arithmetic-geometric mean method uses the Salamin-Brent algorithm. The $\exp(x)$ in Newton's method is evaluated by first reducing the argument into the range $0\leqq x<2^{-\sqrt{p}}$ and then summing terms of the Taylor series for $\exp(x)$. The time complexity is $O(M(p)\log(p))$ for the arithmetic-geometric mean method, $O(M(P)\sqrt{P})$ for the elliptic method and Newton's method, and $O(M(P)P)$ for Taylor's methods 1 and 2.

It should be commented that we have programmed routines for $\sqrt{x}$, $\exp(x)$ and $\log(x)$ to be as efficient as possible. For example, the nth term of the Taylor series for $\exp(x)$, $\log(1+x)$, $\log(1+x)/(1-x)$, θ_0^0, or θ_3^0 is calculated only up to the $(P+G)$th decimal place with G a number of guard digits; the evaluation of the

Table 1 Comparison of five algorithms for log (x). The numbers listed are times (in milli-seconds) for evaluating log(2) to the Pth decimal place. For the details of the algorithms, see the text. The computation was made on a FACOM M-200 computer by using a LISP-based "bigfloat" system. The times listed here do not contain garbage collection times, because the garbage collection times are peculiar to LISP-based systems and they depend on the memory size used (in our experiment, the garbage collection time averaged about 5%). The constants π and log(2) which are necessary in the elliptic method were precomputed to the 10100th decimal place.

P	elliptic method	Taylor's method 1	Taylor's method 2	Newton's method	a-g mean method
20	58	21	15	116	74
40	76	35	23	182	115
60	82	53	30	212	142
80	107	72	40	307	162
100	114	94	51	350	196
150	167	160	84	589	251
200	194	255	130	732	297
400	431	997	485	2,085	687
600	879	2,594	1,223	5,513	1,399
800	1,254	5,430	2,544	8,804	2,133
1000	1,712	9,892	4,533	13,670	3,279
2000	6,259				11,240
4000	24,750				47,280
6000	63,160				114,100
8000	101,400				211,500
10000	196,900				334,100

square root $\sqrt{a_m b_m}$ in the mth step of the arithmetic-geometric mean iteration is carried out by using $a_{m+1}=(a_m+b_m)/2$ as the initial value y_0 for Newton's iteration (10) so far as $a_m\sim b_m$, reducing the number of iterations. (Note that, in the Salamin-Brent algorithm, $a_0=1$ and $b_0=O(10^{-P/2})$ and $a_m\gg b_m$ for $m\lesssim\log_2(P)$. Hence, steps of the first half of the arithmetic-geometric mean iteration in the Salamin-Brent algorithm cannot be made efficient by this technique.) However, we have used $O(P^2)$ algorithms for multiplying and dividing two P-digit numbers.

We can see that the methods using Taylor series are quite efficient for $P\lesssim200$, although they are inefficient for $P\gtrsim1000$. Our algorithm using elliptic theta functions is the most efficient for $P\gtrsim400$, although it takes considerable overhead for $P\lesssim200$. Furthermore, our algorithm is faster than the Salamin-Brent algorithm even for relatively large P although the asymptotic time complexity of our algorithm is worse than that of the Salamin-Brent algorithm. The algorithm using Newton's method is the worst of the five, although its time complexity is asymptotically better than that of Taylor series methods.

In order to investigate our algorithm further, we measured times for computing $\sqrt{2}$, the arithmetic-geometric mean in both our and the Salamin-Brent algorithms, and elliptic theta functions in our algorithm. Table 2 shows the results. We can see that our square-root routine is quite efficient making the calculation of the arithmetic-geometric mean fast. Nevertheless, the

Table 2 Timing data (in milli-seconds) for evaluating $\sqrt{2}$ by our square-root routine, and the arithmetic-geometric mean and θ_0^0 & θ_3^0 which are necessary for calculating log(2) by our and Salamin-Brent algorithms. Garbage collection times are not included.

P	$\sqrt{2}$	a-g mean our method	θ_0^0 & θ_3^0	a-g mean S-B method
20	7	45	5	69
40	9	61	7	109
60	10	65	9	137
80	12	87	12	155
100	12	92	13	189
150	16	137	19	242
200	17	155	26	286
400	31	345	61	666
600	58	726	110	1,363
800	75	1,015	173	2,079
1000	94	1,369	248	3,202
2000		5,104	833	10,980
4000		20,590	2,976	46,310
6000		54,130	6,420	112,000
8000		85,750	11,120	207,700
10000		172,700	17,100	328,300

arithmetic-geometric mean requires much more computing time than the elliptic theta functions. The computing time of the arithmetic-geometric mean in the Salamin-Brent algorithm is about two or a little larger than that in our algorithm, as was expected.

Readers may wonder why the computing times of elliptic theta functions are so small. The reason is a careful control of the precision. In our algorithm, the argument reduction is so made that the reduced argument becomes a number of as few figures as possible when the original argument is a number of few figures. For example, the argument $x=2$ is reduced to the argument $q=2/256=0.0078125$. Furthermore, after calculating each term of a power series, we truncate each number concerned by discarding figures in high precision places which do not contribute to the significant digits of the answer. Therefore, although we are calculating the answer with P significant digits, we are mostly handling numbers whose figures are not as many as P.

According to the above discussion, our log (x) routine and routines using Taylor series and Newton's method will require more computing times when the argument x is composed of many figures. In order to investigate this, we have calculated log (π) by these four algorithms. Table 3 shows the results. We see that the computing times increased considerably compared with those in Table 1. However, our algorithm is still the fastest among the five algorithms we have compared when $P \gtrsim 400$.

Remember that our algorithm has a freedom in reducing the argument q. The q may be any value so far

Table 3 Calculation of log(π) by four algorithms. The times (in milli-seconds) do not contain garbage collection times. For the details of the algorithms, see the text. The timing data by the a-g mean method are almost the same as those in Table 1.

P	elliptic method	Taylor's method 1	Taylor's method 2	Newton's method
20	57	28	18	99
40	75	53	28	156
60	82	80	40	233
80	107	111	53	264
100	115	146	69	306
150	172	257	113	518
200	205	407	178	643
400	498	1,592	677	1,848
600	1,071	4,153	1,742	4,914
800	1,653	8,701	3,624	7,892
1000	2,411	15,840	6,564	19,120

as $0<q\ll 1$. Since our algorithm includes the Salamin-Brent algorithm as a limiting case, we can always make our algorithm faster than the Salamin-Brent algorithm by optimumly setting the value of q. The above experiments indicate that such optimized algorithm will be the most efficient at present for $P \gtrsim 400 \sim 1000$.

Acknowledgements

The authors thank Prof. R. Brent for his useful comment for improving the program and information.

References

1. Beeler, M., Gosper, R. W. and Schroeppel, R. Hakmem, *Memo* No. 239, M.I.T. Artificial Intelligence Lab. (1972), 70–71.
2. Borchardt, C. W. Gesammelte Werke, Berlin, 455–462 (1888).
3. Brent, R. P. Fast multiple-precision evaluation of elementary functions, *J. Assoc. Comput. Mach.* **23** (1976), 242–251.
4. Brent, R. P. Analytic Computational Complexity, Academic Press, New York-San Francisco-London (1976), 151–176.
5. Carlson, B. C. Algorithms involving arithmetic and geometric means, *Amer. Math. Monthly* **78** (1971), 496–505.
6. Carlson, B. C. An algorithm for computing logarithms and arctangents, *Math. Comput.* **26** (1972), 543–549.
7. Gauss, C. F. Carl Friedrich Gauss Werke, Bd. 3, Göttingen (1876).
8. Hancock, H. Theory of Elliptic Functions, Dover, New York (1961).
9. Kanada, Y. and Sasaki, T. LISP-based big-float system is not slow, SIGSAM Bulletin 15 (1981), 13–19.
10. Salamin, E. Computation of π using arithmetic-geometric mean, *Math. Comput.* **30** (1976), 565–570.
11. Sasaki, T. An arbitrary precision real arithmetic package in REDUCE, *Lecture Notes in Comput. Sci.* 72, Springer-Verlag (1979), 358-368.
12. Schwab, J. Elémens de Géométrie, Vol. 1, Nancy (1813), 103-107.
13. Thacher, H. C. Iterated square root expansions for the inverse cosine and inverse hyperbolic cosine, *Math. Comput.* **15** (1961), 399–403.

(Received January 28, 1982: revised May 6, 1982)