1989

Efficient Multiple-Precision Evaluation of Elementary Functions

David M. Smith
Loyola Marymount University, dsmith@lmu.edu

MATHEMATICS OF COMPUTATION
VOLUME 52, NUMBER 185
JANUARY 1989, PAGES 131-134

Efficient Multiple-Precision Evaluation of Elementary Functions

By David M. Smith

Abstract. Let $M(t)$ denote the time required to multiply two t-digit numbers using base b arithmetic. Methods are presented for computing the elementary functions in $O(t^{1/3}M(t))$ time.

1. Introduction. In [2] Brent shows that the elementary functions can be computed with t digits of precision using base b arithmetic in $O(M(t)\log t)$ operations. $M(t)$ represents the time required to perform one t-digit multiplication. These are the fastest known methods asymptotically, but because the algorithms are complicated, for precisions of no more than a few thousand digits there are more efficient algorithms.

For commonly used precisions the best methods currently in use run in $O(t^{1/2}M(t))$ time [1], [3]. At these precisions, $M(t) = O(t^2)$, although faster methods exist for high precision [4]. This paper presents similar algorithms for which the running time has been improved to $O(t^{1/3}M(t))$. Because this improvement is fairly simple, the resulting algorithms are faster than those in [3] even at low precision.

2. Exponential and Related Functions. Function computations can often be speeded up by using various identities to reduce the size of the argument prior to summing a series, and then reversing the reduction at the end. The exponential identity

$$\exp(x) = \exp(x/2^k)^{2^k}$$

can be used as follows. Compute $y = x/2^k$ using a few divide by integer operations, then sum the series for $\exp(y)$, then do k squarings to recover $\exp(x)$. Brent uses this technique in [3] to obtain an algorithm with speed $O(t^{1/2}M(t))$.

Since the power series for $\exp(x)$ consists of terms which are closely related, the next term can be obtained from the previous term by one division by a small integer and one $O(M(t))$ operation to get the next power of x. The operations with integers and the addition of the terms are all $O(t)$, so reducing the number of multiplications is important. The direct sum

$$\exp(x) \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

requires $(n-1)$ multiplications, $(n-1)$ divisions by an integer, and n additions.

Received June 25, 1987; revised February 18, 1988.
1980 *Mathematics Subject Classification* (1985 *Revision*). Primary 65D15.

The sum can be rearranged as j concurrent sums

$$\begin{array}{llll} & 1 & +x^j/j! & +x^{2j}/(2j)!+\cdots \\ +x & [1 & +x^j/(j+1)! & +\cdots \\ +x^2 & [1/2! & +x^j/(j+2)! & +\cdots \\ +x^3 & [1/3! & +x^j/(j+3)! & +\cdots \\ \vdots & & & \vdots \\ +x^{j-1} & [1/(j-1)! & +x^j/(2j-1)! & +\cdots . \end{array}$$

To add the next term to each of these sums $S_0, S_1, \ldots, S_{j-1}$ requires one multiplication to get the next power of x^j, j divisions by an integer, and j additions. The original x^j term can be obtained in $O(\log j)$ multiplications using a binary exponentiation method [4]. Then the polynomial $S_{j-1}x^{j-1}+\cdots+S_1x+S_0$ is evaluated as $(\cdots(S_{j-1}x+S_{j-2})x+\cdots+S_1)x+S_0$, which takes $(j-1)$ multiplications and $(j-1)$ additions. For high precision this means the number of multiplications needed to compute $\exp(x)$ is about $n/j+j$ and the number of $O(t)$ operations is the same as for the direct sum. Because the j sums must be stored separately, this algorithm uses more space than the $O(t^{1/2}M(t))$ algorithm. Although the actual order of the operations is different here, the idea behind the arrangement above is similar to one given by Paterson and Stockmeyer [5].

To estimate the time for this algorithm, assume that the argument is about 1 in magnitude, base b arithmetic with t digits is used, and k halvings are done before the j sums are computed. The original argument x is assumed to lie in some fixed, bounded interval, and the number of terms n needed from a Taylor series is assumed to be a single-precision integer. The value of n is determined by the equation

$$\frac{(2^{-k})^n}{n!} = b^{-t}.$$

Using Stirling's approximation for $n!$ provides an approximation for the number of terms in the series which must be taken:

$$n \approx \frac{t\log b}{\log t + k\log 2}.$$

Including the k squarings needed to reverse the argument reduction, the total work, W, is estimated by the number of multiplications:

$$W \approx \frac{t\log b}{j(\log t + k\log 2)} + j + k.$$

Choosing j and k to minimize W gives

$$j = t^{1/3}(\log b/\log 2)^{1/3},$$
$$k = t^{1/3}(\log b/\log 2)^{1/3} - \log t/\log 2.$$

Letting j and k be the nearest integers to these values gives an algorithm with $O(t^{1/3})$ multiplications, and it follows that $\exp(x)$ can be computed in $O(t^{1/3}M(t))$ time.

Logarithms can be computed in $O(t^{1/3}M(t))$ time using Newton iteration and the exponential function. Starting with an approximation generated in single or

double precision, the precision is doubled at each iteration until the desired multiple-precision accuracy is obtained. Since only the last iteration is done at full precision, computing the logarithm takes only slightly longer than the exponential function.

Power functions and hyperbolic functions can be computed from formulas involving exponential and/or logarithm functions, so they are also obtained in $O(t^{1/3}M(t))$ time.

3. Trigonometric Functions. For $\sin(x)$ the argument can first be reduced to lie between 0 and $\pi/4$ using various identities. Then this value is further reduced by dividing by 3^k, and then the Taylor series is added as j sums in a manner similar to $\exp(x)$. After summing the series, $\sin(x)$ is recovered by k iterations of the formula

$$\sin(3a) = \sin(a)(3 - 4\sin^2(a)).$$

This requires two full multiplications for each of the k steps, the reduced argument is about 3^{-k}, and the sine series has only half as many terms as the exponential series. The total number of $O(M(t))$ operations done in computing $\sin(x)$ is about

$$W \approx \frac{t \log b}{2j(\log t + k \log 3)} + j + 2k.$$

Minimizing W gives

$$j = t^{1/3}(\log b/\log 3)^{1/3},$$
$$k = \tfrac{1}{2}t^{1/3}(\log b/\log 3)^{1/3} - \log t/\log 3,$$

so the sine is computed in $O(t^{1/3}M(t))$ time. Because the sum has only $n/2$ terms, and reversing the argument reduction takes longer than for the exponential, the algorithm does less argument reduction than for $\exp(x)$.

The other trigonometric functions can be computed from $\sin(x)$ and identities. Inverse trigonometric functions can be done using Newton iteration and $\sin(x)$. This gives all the trigonometric functions in $O(t^{1/3}M(t))$ time.

4. Results Using Fast Multiplication. If a multiplication algorithm can be used which is much faster that $O(t^2)$, then the time taken for all the $O(t)$ operations becomes large enough to change the best values of j and k. There are $O(n)$ additions and integer divisions, with $n = O(t/k)$, so the time for $\exp(x)$ could then be estimated by

$$T \approx \left(\frac{t \log b}{j(\log t + k \log 2)} + j + k\right) M(t) + \frac{2t^2}{k}$$

and a similar expression would apply to $\sin(x)$. If $M(t) = o(t^{4/3})$ with j and k still $O(t^{1/3})$ as above, then the time spent on multiplications is $o(t^{5/3})$, while the additions and integer divisions take $O(t^{5/3})$ time.

Minimizing T gives different choices for j and k when $M(t) = o(t^{4/3})$. In this case the best values are $j = O(\sqrt[4]{M(t)})$ and $k = O(t/\sqrt{M(t)})$, and the algorithm runs in $O(t\sqrt{M(t)})$ time. So if a very fast multiplication algorithm is used, fewer concurrent sums are needed and more argument reduction is done.

5. Conclusion. The formulas for j and k above are approximations which would be modified slightly in a program for computing the elementary functions using multiple-precision arithmetic.

The $O(t)$ operations cannot be ignored completely at low precision, and some guard digits are needed during the computation so that the final result can be rounded correctly to t digits. During the summing of the series many of the operations can be done at less than full precision.

These factors mean that a program which implements these $O(t^{1/3}M(t))$ algorithms efficiently will use constants in the formulas for j and k which have been chosen to take these details into account.

Tests comparing such a program with Brent's MP package [3] have been made. Using a large base for the arithmetic, the $O(t^{1/3}M(t))$ versions are 10–20% faster for $t = 10$ and increase to 2–3 times as fast for $t = 250$. These algorithms are now the fastest known methods using multiple-precision arithmetic with low to moderate precision for computing the elementary functions.

Mathematics Department
Loyola Marymount University
Los Angeles, California 90045

1. R. P. BRENT, "The complexity of multiple-precision arithmetic," in *Complexity of Computational Problem Solving* (R. S. Anderssen and R. P. Brent, eds.), Univ. of Queensland Press, Brisbane, 1976, pp. 126–165.

2. R. P. BRENT, "Fast multiple-precision evaluation of elementary functions," *J. ACM*, v. 23, 1976, pp. 242–251.

3. R. P. BRENT, "A Fortran multiple-precision arithmetic package," *ACM Trans. Math. Software*, v. 4, 1978, pp. 57–70.

4. D. E. KNUTH, *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, Mass., 1981.

5. M. S. PATERSON & L. J. STOCKMEYER, "On the number of nonscalar multiplications necessary to evaluate polynomials," *SIAM J. Comput.*, v. 2, 1973, pp. 60–66.