Comp. & Maths. with Appls., Vol. 5, pp. 163–167
Pergamon Press Ltd., 1979. Printed in Great Britain

EVALUATION OF FUNCTIONS ON MICROCOMPUTERS: RATIONAL APPROXIMATION OF *k*th ROOTS

M. ANDREWS,† B. EISENBERG, S. F. MCCORMICK and G. D. TAYLOR
Department of Mathematics, Colorado State University, Fort Collins, CO 80523, U.S.A.

Communicated by R. B. Kelman

(*Received January* 1978; *and in revised form August* 1978)

Abstract—This paper describes the implementation of rational approximation algorithms for evaluation of *k*th roots in short wordlength machines. The emphasis is on maintaining full machine precision in computers that use fixed point, truncated binary arithmetic with at most 16 bits of wordlength. Included is a table of coefficients for evaluation of *k*th roots on a 16 bit machine with $3 \le k \le 11$.

1. INTRODUCTION

Keeping pace with external processes while maintaining full machine precision during computation is a necessary and demanding task for microprocessor software used in real-time applications. To be effective, numerical algorithms used in this environment must be carefully designed and implemented. With real-time applications in mind, this paper describes several ingredients that must be considered for rational approximation of *k*th roots ($k \ge 3$) on small scale machines. (For purposes of discussion, the terms small scale machines and microprocessors shall refer to binary computers using fixed point, truncated arithmetic with at most 16 bits of wordlength.) For motivation and a general comparison of small and large scale machines and their uses, see [1].

It should be noted that techniques based on rational polynomial approximation are not the best choice for every possible small scale machine or application. Of course, this can be said of any algorithm used to compute *k*th roots, simply because special machine architectures and application requirements, e.g., accuracy, may advantageously support other techniques such as bit counting[2], local Taylor expansion, Newton's method[3, 4], table look-up, or hybrid combinations of these techniques[5]. However, for most small scale machines presently in use and for applications which require general root finders, a rational polynomial approximation is probably the method of choice. As we shall see, such approximations achieving near machine accuracy require relatively few arithmetic operations and data storage locations, and the general structure of the algorithm may be used to reduce program length over a wide range of desired values of *k*. For focus, the discussion concentrates on the values $3 \le k \le 11$.

For large computers, the choice of appropriate degrees and coefficients for rational approximation is generally a matter of considering tables of approximation error. For small computers, as one would expect, the severe truncation effects of short wordlength computation disrupt the theoretical values to the extent of requiring special care. Thus, computation of the actual approximation error in the microcomputer environment is essential to the proper selection of the degrees of the rational polynomial approximants as well as the correct selection of coefficients. This is the subject of Section 3. We first discuss some ingredients in Section 2 that need to be considered for careful implementation of a *k*th root algorithm.

2. IMPLEMENTATION

In this section, short descriptions are given for mechanisms useful for computing *k*th roots in small scale computers.

1. *Argument reduction*

(a) *Interval.* The choice of the primary interval is an important part in designing a general

†Computer Science Department.

This work was supported by the National Science Foundation under grant NSF MCS 76-12457.

root finding routine. Considerations analogous to those for square roots[1] might mislead one into choosing the interval $[1/2^k, 1)$ for computing kth roots since no multiply is required in the argument restoration phase. However, this would require the degrees of the polynomials in the rational approximation to increase with increasing k in order to obtain the desired accuracy over $[1/2^k, 1)$. Thus, for a general routine, $[1/2, 1)$ is more suitable for accurate rational polynomial approximation. Moreover, the interval $[1/2, 1)$ consists of left justified numbers so that greater precision is maintained in this mode.

(b) *Restoration.* Assume that the input value is represented by $\hat{x} = x \cdot 2^m$ where $x \in [1/2, 1)$. Let $m = m_0 \cdot k + q$, where m_0, q are integers and $0 \le q < k$ (note that the signs of m and m_0 are the same). Define $\phi(q, k) = 2^{(q/k)-1}$. Then

$$\hat{x}^{1/k} = x^{1/k} \cdot 2^{m_0+1} \cdot \phi(q, k). \tag{1}$$

(Since $\phi(q, k) \in [1/2, 1)$, the product $x^{1/k} \cdot \phi(q, k)$ can be formed without concern for overflow.) Restoration of the reduced argument now involves storage and multiplication by the quantities $\{\phi(q, k): 0 \le q \le k - 1\}$. This is somewhat of a disadvantage, but it is compensated by the low degree approximations needed for kth roots over $[1/2, 1)$.

2. *Sign bit*

The bit set aside in each arithmetic word to indicate sign is not needed in the kth root routine since the sign of a legitimate result is the same sign as the input value of x, and it is possible to ensure that all intermediate results are positive quantities. Thus, to reduce error, it is important to perform arithmetic operations internal to the kth root routine in the full wordlength of the machine. The commonly used sign bit position should therefore be employed as a significant bit of each operand and unsigned arithmetic should be performed.

3. *Carry bit*

Microcomputers do not exhibit guard bits attached to each word, but in general have a carry or overflow bit resident on the CPU chip that is set to one when and only when the proceeding arithmetic operation resulted in overflow. Such a capability is assumed in this paper.

4. *A transformation*

Since $x \in [1/2, 1)$, then $x^{1/k} \in [1/2, 1)$. Thus its leading bit must be 1 and precision can be extended to approximating $2x^{1/k} - 1$ instead of $x^{1/k}$.

5. *Rational approximation*

(a) *Parameter choice.* The selection of the appropriate form and specific coefficients required for the desired accuracy of approximation must be done carefully. Theoretical results for real number systems are useful only as guidelines for these choices. In the next section, we present these guidelines and provide examples for 16-bit machines with the values $k = 3, 4, 5, \ldots, 11$, assuming full machine precision is desired. In any event, for a general root finding subroutine, the degrees of the rational fit should be the same for all k so as to minimize program length. The apparent relative insensitivity of the rational approximation error to variations in k is an important feature in this regard.

(b) *Evaluation.* The order of operations in evaluating the chosen rational approximation is critical to maintaining accuracy and avoiding overflow. The 16-bit example given in section 3 [see eqn (3)] assumes that the polynomials in the numerator and denominator are evaluated first by Horner's method followed by a divide. This sequence facilitates the avoidance of overflow as discussed in part (c). Other methods such as Product, Classical Orthogonal, Newton or PAN are not competitive since they require more arithmetic operations (cf. [6, p. 67]).

(c) *Avoiding overflow.* Protection against overflow in the rational polynomial evaluation routine is first a matter of the proper choice of coefficients once the evaluation procedure is selected. Here we assume that Horner's method is used to evaluate the polynomials, so that suitable coefficients for the numerator and denominator may be chosen *a priori* by constraining their respective absolute sums to be smaller than unity.This constraint may be relaxed when all

but the constant term is positive as shown in Section 3. This has the tendency to reduce accuracy since this restriction lessens information carried in the significant bits in the evaluation. However, the other mechanisms described in this section overcome this loss so that roots to full machine precision are obtained.

The reader should be cautioned that intermediate evaluation of the polynomials (see equation (3) below) can result in overflow. However, this can occur only in evaluating expressions of the form $a_2x + a_1$ and it is always intermediate to forming $(a_2x + a_1)x$, which is theoretically less than one. Thus, a temporary fix is necessary to account for overflow by testing the carry bit and, if necessary, correcting after the post multiplication by x simply by adding x to the result. This is equivalent in principle to forming $(a_2x - (1 - a_1))x + x$.

The reader will note in the following section that the true value of the numerator in the rational approximant when $k = 2$ or 3 is always larger than unity for the coefficients given in Table 1. Specifically, the last add in the expression $(a_2x + a_1)x + a_0$ will cause an overflow for such values of k. This overflow should be ignored! That is, the correct expression for the numerator is $(a_2x + a_1)x - (1 - a_0)$, but since this would require a subtraction or an addition of a negative number, we have chosen instead to use the overflow feature as an artificial means of forming $[(a_2x + a_1)x + a_0] - 1$. Note that the coefficients in Table 1 are expressed as integers. Actually, considered as real numbers, they must each be divided by 2^{16}. Thus, for example, with $k = 3$ we have $a_2 = (0.1000001001000010)_2$.

Finally, for some values of x very near unity, the *machine* evaluation of the rational approximation of $2x^{1/k} - 1$ can in fact exceed unity. For such values of x, output from the kth root routine should be the largest machine representable number less than unity.This can be done either *a priori* by testing for such values of x on input or *a posteriori* by comparing the numerator and denominator before the divide (or checking the carry bit afterward).

Table 1. Coefficients for evaluation of equation (3) (See text for full details.)

k	a_2	a_1	a_0	b_2	b_1	b_0	Minimum accuracy	Average accuracy
3	33346	32156	61283	8340	41343	11566	15.555	17.735
4	29695	35522	63947	10128	42848	10651	15.263	18.012
5	26087	36190	140	10757	41889	9771	15.033	17.665
6	25349	38847	1341	11923	43858	9756	15.049	17.456
7	21732	35851	1997	11260	29775	8544	15.299	17.965
8	20735	36258	2587	11570	39696	8313	15.072	17.881
9	20036	36523	3021	11866	39625	8088	15.193	17.943
10	21137	40609	3791	13099	43578	8859	15.167	18.017
11	20245	41053	4238	13014	43601	8921	15.091	17.586

6. *Recommended sequence of steps*

Here we assume k, x and m are given, where $\hat{x} = x \cdot 2^m$, the left bit of x is used for the sign, and $1/2 \leq |x| < 1$. All shift operations are arithmetic; that is, zeros are entered into the vacated bit positions. The returned result is represented by y and m_0, where $y \cdot 2^{m_0} \sim \hat{x}^{1/k}$.

(a) Access the sign bit in x and error trap if k is even and x is negative. Store the sign bit as s and single left shift x.

(b) Compute the integers m_0 and q so that $m = m_0 \cdot k + q$, $0 \leq q < k$.

(c) Select the appropriate coefficients for the given k from Table 1.

(d) Evaluate the rational form based on the coefficients for approximating $2 \cdot x^{1/k} - 1$, calling the result r.

(e) Evaluate $y = t + r \cdot t$, where t is the stored value of $\phi(q, k)$. Test for overflow after the addition and, in the event it occurs, replace y by $(y/2) + (1/2)$ and increment m_0 by one. In either event, complete this step with a single right shift of y.

(f) Convert the most significant bit of y to agree with s. (Note that the right shift in step e guarantees that the leading bit of y is zero.) Output y and m_0.

7. *Rounding*

For evaluation of the rational approximation according to the implementation suggested in

the next section, it is important that only *truncated* arithmetic operations be performed. Since the coefficients were chosen assuming truncated arithmetic, it would actually be less accurate to perform rounding after each operation. On the other hand, in other parts of the kth root routine, namely, the argument reduction and restoration phases and step (e) above, it is recommended that rounding be used provided the application will permit the added computation.

3. SELECTION OF RATIONAL APPROXIMATIONS

Standard techniques of uniform rational approximation theory may be modified to select the appropriate rational approximation for $2x^{1/k} - 1$ on

$$X = [1/2, 1) \cap \left\{x : x = \frac{j}{2^{16}},\ j \text{ an integer}\right\}$$

for $k = 3, 4, \ldots, 11$. Each case is treated separately. For example, in the construction of Table 1, we first approximate $2x^{1/k} - 1$ on a subset T of X using the differential correction algorithm [7] applied to various different classes of rational functions $R_n^m(T)$, m, n nonnegative with $n + m = l$, $l = 2, 3, 4, 5$, where

$$R_n^m(T) = \left\{r(X) \equiv \left(\sum_{i=0}^{m} a_i x^i\right) / \left(\sum_{i=0}^{n} b_i x^i\right) : \sum_{i=0}^{n} b_i x^i > 0,\ x \in T\right\}. \tag{2}$$

This is done to estimate the error of approximation as a function of the particular classes $R_n^m(T)$. (For a FORTRAN listing of the differential correction algorithm and additional references concerning this algorithm see [7, 9].) Based upon these results, a class R_n^m is then selected as a candidate for approximating $2x^{1/k} - 1$. For instance, this leads to the choice R_2^2 that yields a "theoretical" error of 8.7×10^{-7} for $k = 3$ in the worst case based upon calculation on a CDC CYBER 172. We remark that R_2^2 gave full machine precision results with the least number of primitive operations ($+$, $-$, $*$, $/$) using a CDC CYBER 172. See Table 2 for a

Table 2. Comparison of maximum approximation error for R_2^2 based on precise and actual 16-bit truncated arithmetic evaluation of the rational approximants

k	3	4	5	6	7	8	9	10	11
Theoretical error	8.7E−7	7.2E−7	6.1E−7	5.2E−7	4.5E−7	4.0E−7	3.6E−7	3.2E−7	2.9E−7
Actual error	2.1E−5	2.5E−5	3.0E−5	2.9E−5	2.5E−5	2.7E−5	2.7E−5	2.7E−5	2.9E−5

summary of these computational results. Next, the theoretically best approximation is perturbed to provide a sufficiently good approximation for the desired application. The need for this step arises from the severe effects of truncation (or rounding) in a short wordlength environment. To accomplish this, we applied a modified version of the differential correction algorithm for computing a "best" approximation from $R_2^2(T)$ to $2x^{1/k} - 1$, k fixed, $k = 3, 4, \ldots, 11$. Basically, this version is simply the standard algorithm modified to do pertinent calculations in a sixteen bit truncated mode to seek coefficients that are best relative to the mode (fixed or floating point) of arithmetic in which the approximation would be evaluated. Running this algorithm (at most two iterations were required) for fixed k and $R_2^2(T)$ resulted in

$$R_k(x) = \frac{(a_2 x + a_1)x + a_0}{(b_2 x + b_1)x + b_0} \tag{3}$$

for approximating $2x^{1/k} - 1$. Since this function is unique up to a scalar multiple of both the numerator and denominator, one can scale the coefficients a_0, a_1, a_2, b_0, b_1 and b_2, if necessary, so that for all $x \in X$, both the numerator and denominator are strictly bounded below by unity. Observe that if one expands $r_k(x)$ as a partial fraction, an additional multiply can be saved. Yet, caution is advised since this approach may introduce additional scaling difficulties. See Table 1 for a listing of the coefficients corresponding to different values of k.

As noted earlier, due to the small number of machine representable numbers in the interval [1/2, 1), the error of approximation should actually be calculated at each of these values. One efficient procedure for accomplishing this is to simulate the particular mode of computation under consideration for the evaluation of $r_k(x)$ on a high speed large scale computer and to compare the evaluation so obtained to the large scale machine value of $2x^{1/k} - 1$. Observe that if

$$|r_k(x) - (2x^{1/k} - 1)| \leq 2^{-s} \tag{4}$$

then

$$\left|\left(\frac{r_k(x)}{2} + \frac{1}{2}\right) - x^{1/k}\right| \leq 2^{-s-1}, \tag{5}$$

producing an extra bit of precision as claimed.

In the event that the desired accuracy of approximation of $2x^{1/k} - 1$ by $r_k(x)$ is not achieved, we suggest the following procedures for generating an improved approximation. First, one should examine the error curve corresponding to the current approximation carefully. If there appears to be hope for improvement within the class of approximants being considered, e.g. the error curve has constant sign, points of maximum modulus all have the same sign, or the maximum modulus is attained only a few times, then start a local search (directional or otherwise) in a neighborhood of the current approximation for an improved approximation. Another strategy here would be to replace T by a larger subset of X and find a good approximation on this larger subset. If these strategies fail, then one should enlarge the class of approximants. With regard to this last possibility, it should be noted that proceeding to a larger class of approximants does not *a priori* guarantee increased accuracy. The larger class may give better theoretical accuracy in precise arithmetic; however, even in such cases where theoretically increased accuracy occurs, the truncation effects of the extra arithmetic operations needed to evaluate these theoretically better approximations can be deleterious and may fail to improve the approximation. Thus, additional care is needed for this option.

In the specific cases of this paper, we achieved an acceptable error of approximation for all values of k considered. As noted, we simulated 16-bit truncated arithmetic for the evaluation of each $r_k(x)$ on a CDC CYBER 172. Evaluating $r_k(x)$ in this mode and approximating $x^{1/k}$ with $(r_k(x)/2) + (1/2)$ (where this last expression was also evaluated using 16 bit truncated arithmetic yielded the results shown in Table 2, where the accuracy is measured as

$$-\log_2\left|x^{1/k} - \left(\frac{r_k(x)}{2} + \frac{1}{2}\right)\right|.$$

REFERENCES

1. M. Andrews, S. F. McCormick and G. D. Taylor, Evaluation of Functions on Computers: Square Root. *Comput. Math. Applic.* 4(4), 359–367 (1979).
2. T. C. Chen, Efficient Arithmetic Apparatus and Methods. *U.S. Pat.* 3,631,230, December, (1971).
3. D. L. Phillips, Generalized Logarithmic Error and Newton's Methods for the mth Root. *Math. Comput.* 24, 383–389 (1970).
4. G. D. Taylor, Optimal Starting Approximations for Newton's Method. *J. Approx. Theory*, 3, 156–163 (1970).
5. P. W. Baker, More Efficient Radix-2 Algorithms for Some Elementary Functions. *IEEE Trans. Computers*, C24(11), 1049–1054 (1975).
6. J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. R. Rice, H. C. Thacher and C. Witzgall, *Computer Approximations*, SIAM Series in Applied Mathematics, Wiley, New York, 67–69 (1968).
7. I. Barrodale, M. J. D. Powell and F. D. K. Roberts, The Differential Correction Algorithm for Rational l_∞ Approximation. *SIAM J. Numer. Anal.* 9, 493–504 (1972).
8. C. M. Lee and F. D. K. Roberts, A Comparison of Algorithms for Rational l_∞ Approximation. *Math. Comput.* 27, 111–121 (1973).
9. E. H. Kaufman, Jr and G. D. Taylor, Uniform Rational Approximations of Functions of Several Variables. *Internat. J. numer. Methods Engrg.* 9, 297–323 (1975).