BKM: a new hardware algorithm for complex elementary functions

Jean-Michel Muller, Jean-Claude Bajard, Sylvanus Kla

BKM: A New Hardware Algorithm for Complex Elementary Functions

Jean-Claude Bajard, Sylvanus Kla, and Jean-Michel Muller, *Member, IEEE*

***Abstract*— A new algorithm for computing the complex logarithm and exponential functions is proposed. This algorithm is based on shift-and-add elementary steps, and it generalizes some algorithms by Briggs and De Lugish, as well as the Cordic algorithm. It can easily be used to compute the classical real elementary functions (sin, cos, arctan, ln, exp). This algorithm is more suitable for computations in a redundant number system than the Cordic algorithm, since there is no scaling factor when computing trigonometric functions.**

***Index Terms*— Computer arithmetic, elementary functions, cordic, redundant number systems.**

I. INTRODUCTION

THE point at stake in this paper is the search for algorithms that rapidly compute elementary functions. Many methods have been proposed in the literature: approximation by polynomials or rational functions [3], [5], [6], [10], use of Newton's method, approximation by continued products, continued fractions, E-Method [8], [9] and shift-and-add methods. The algorithms presented in this paper belong to the class of shift-and-add methods. These methods are based on simple elementary steps: additions, and shifts (i.e., multiplications by a power of the radix of the number system used), and they go back to the 17th century: Briggs, a contemporary of Neper, invented an algorithm that made it possible to build the first tables of logarithms. For instance, to compute the logarithm of x in radix-2 arithmetic, numerous methods (including that of Briggs, adapted to this radix) [4], [12], [13], [15] broadly consist of finding a sequence $d_k = -1, 0, 1$, such that $x\prod_{k=1}^{n}(1 + d_k 2^{-k}) \approx 1$. Then $\ln(x) \approx -\sum_{k=1}^{n} \ln(1 + d_k 2^{-k})$. Another method belonging to the shift-and-add class is the CORDIC algorithm, introduced in 1959 by J. Volder [17] and then generalized by J. Walther [18]. CORDIC consists of the following iteration:

$$\begin{cases} x_{n+1} &= x_n - m d_n y_n 2^{-\sigma(n)} \\ y_{n+1} &= y_n + d_n x_n 2^{-\sigma(n)} \\ z_{n+1} &= z_n - d_n e_{\sigma(n)}, \end{cases} \tag{1}$$

m equals 0, 1 or -1, and d_n is equal to 1 or -1. The results and the values of d_n, m and $\sigma(n)$ are presented in Tables I and II.

Manuscript received September 17, 1993; revised February 23, 1994.

J.-C. Bajard is with Lab. LIM, Université de Provence, Marseille, 3 Place Victor Hugo, 13331 Marseille Cedex 03, France.

S. Kla is with INSET, Dept. FS, BP 1093 Yamussukro, Ivory Coast.

J.-M. Muller is with CNRS, Lab. LIP, Ecole Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France; e-mail: jmmuller@lip.ens-lyon.fr.

IEEE Log Number 9203096.

CORDIC allows computation of many functions [18]. For instance, e^x is obtained as $\cosh x + \sinh x$, while $\ln x$ is obtained as $2\tanh^{-1}\left|\frac{1-x}{1+x}\right|$. CORDIC has been implemented in many pocket calculators and floating-point coprocessors. Its major drawback arises when performing the iterations using a *redundant* (e.g., carry save or signed-digit) number system. Such systems are advantageous for quickly-performed arithmetic, since they make it possible to perform carry-free additions [2]. In these systems, d_n is difficult to evaluate. For instance, assume that we are in the *rotation mode* of CORDIC (see Table I), and that numbers are represented by p digits, in radix 2 with digits -1, 0 or 1. d_n is equal to the sign of the most significant nonzero digit of z_n: to find its value, we must examine some number of digits which may be close to p. Thus, the advantage of the redundant representation (a constant time elementary step) would be lost. An alternative is to accept $d_n = 0$: this makes it possible to examine a few digits of z_n only. Unfortunately, with such a method the scale factors K and K' are not constants. K is equal to $\prod_{n=0}^{\infty}\sqrt{1 + d_n^2 2^{-2n}}$, it is a constant if the d_i's are all equal to ± 1, but it is no longer a constant if the d_i's are allowed to be zero. Many solutions have been suggested to overcome this problem. Broadly speaking, they lead to a repetition of iterations in time [16], [1], or in space [7]. In order to avoid this wasteful repetition, we need to work out a new algorithm.

In the following, assume that we are using a radix-2 classical or signed-digit number system. Extension to binary carry-save representation is straightforward. Let us consider the basic step of CORDIC in trigonometric mode (i.e., iteration (1) with $m = 1$). If we define the complex number L_n as $L_n = x_n + iy_n$, we obtain $L_{n+1} = L_n(1 + id_n 2^{-n})$, this relation is close to the basic step of Briggs' algorithm. This remark brings us to a generalization of that algorithm: we could perform multiplications by terms of the form $(1 + d_n 2^{-n})$, where the d_n's are *complex numbers*, chosen such that a multiplication by d_n can be reduced to a few additions. In this paper, we study the following iteration, called BKM:

$$\begin{cases} L_{n+1} = L_n(1 + d_n 2^{-n}) \\ E_{n+1} = E_n - \ln(1 + d_n 2^{-n}) \end{cases} \tag{2}$$

with $d_n \in \{-1, 0, 1, -i, i, 1-i, 1+i, -1-i, -1+i\}$. We define $\ln z$ as the number t such that $e^t = z$, and whose imaginary part is between $-\pi$ and π.

- If we are able to find a sequence d_n such that L_n goes to 1, then we will obtain $E_n \to E_1 + \ln(L_1)$. We call this iteration mode the *L-mode* of the BKM algorithm.

TABLE I
DIFFERENT FUNCTIONS COMPUTABLE USING CORDIC

	$d_n = sign(z_n)$ (rotation mode)	$d_n = -sign(y_n)$ (vectoring mode)	scale factor
$m = 1$	$x_n \to K(x_0 \cos z_0 - y_0 \sin z_0)$ $y_n \to K(y_0 \cos z_0 + x_0 \sin z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 + y_0^2}$ $y_n \to 0$ $z_n \to z_0 - \arctan(y_0/x_0)$	$K = \prod_{n=0}^{\infty} \sqrt{1 + 2^{-2n}}$ $\simeq 1.646760$
$m = 0$	$x_n \to x_0$ $y_n \to y_0 + x_0 z_0$ $z_n \to 0$	$x_n \to x_0$ $y_n \to 0$ $z_n \to z_0 - y_0/x_0$	–
$m = -1$	$x_n \to K'(x_1 \cosh z_1 + y_1 \sinh z_1)$ $y_n \to K'(y_1 \cosh z_1 + x_1 \sinh z_1)$ $z_n \to 0$	$x_n \to K'\sqrt{x_1^2 - y_1^2}$ $y_n \to 0$ $z_n \to z_1 - \tanh^{-1}(y_1/x_1)$	$K' = \prod_{n=1}^{\infty} \sqrt{1 - 2^{-2\sigma(n)}}$ $\simeq 0.828159$

TABLE II
VALUES OF $\sigma(n)$ AND e_n

$m = 1$ (circular mode)	$\sigma(n) = n$	$e_n = \arctan 2^{-n}$
$m = -1$ (hyperbolic mode)	$\sigma(n) = n - k$ where k is the largest integer such that $3^{k+1} + 2k - 1 \leq 2n$	$e_n = \tanh^{-1} 2^{-n}$
$m = 0$ (linear mode)	$\sigma(n) = n$	$e_n = 2^{-n}$

- If we are able to find a sequence d_n such that E_n goes to 0, then we will obtain $L_n \to L_1 e^{E_1}$. We call this iteration the *E-mode* of BKM.

So, in the next sections, we will focus on the problem of finding sequences d_n such that L_n goes to 1 or such that E_n goes to 0. From this study, we will deduce convenient algorithms for computing the complex logarithm and exponential functions.

II. COMPUTATION OF THE EXPONENTIAL FUNCTION (E-MODE)

As pointed out at the end of the previous section, for computing e^{E_1} using BKM, one needs to find a sequence $d_n, d_n = -1, 0, 1, -i, i, i-1, i+1, -i-1, -i+1$, such that $\lim_{n\to\infty} E_n = 0$. Let us define d_n^x and d_n^y as the real and imaginary parts of d_n (they belong to $\{-1, 0, 1\}$) and E_n^x and E_n^y as the real and imaginary parts of E_n. We easily find:

$$\begin{cases} E_{n+1}^x = E_n^x - \frac{1}{2}\ln\left[1 + d_n^x 2^{-n+1} + \left(d_n^{x\,2} + d_n^{y\,2}\right)2^{-2n}\right] \\ E_{n+1}^y = E_n^y - d_n^y \arctan\left(\frac{2^{-n}}{1 + d_n^x 2^{-n}}\right) \end{cases} \tag{3}$$

In this section, we give an algorithm which computes the sequence d_n for any E_1 belonging to a rectangular set $R_1 = [-s_1^x, r_1^x] + i[-r_1^y, r_1^y]$. The proof of our algorithm is based on the construction of a sequence $R_n = [-s_n^x, r_n^x] + i[-r_n^y, r_n^y]$ of rectangular sets, whose length goes to zero as n goes to infinity, and such that for any $E_n \in R_n$, d_n is such that $E_{n+1} \in R_{n+1}$. d_n^x is chosen by examining a few digits of E_n^x and d_n^y is chosen by examining a few digits of E_n^y. These properties allow a simple and fast implementation of the choice of d_n.

A. Choice of d_n^x

The diagram presented in Fig. 1 shows the different parameters involved in determining d_n^x. This figure is close to the *Robertson Diagrams* that appear in many division algorithms [14]. In the following, we call such a diagram a *Robertson diagram*. The diagram is constructed as follows.

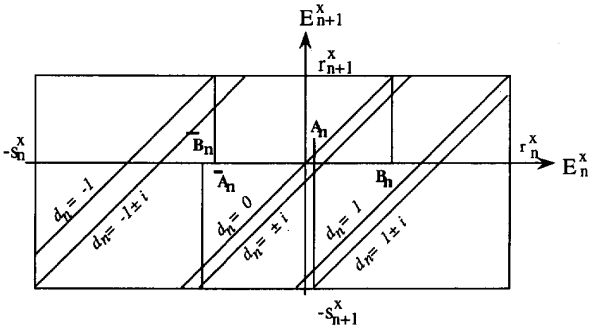


Fig. 1. The Robertson Diagram for E_n^x.

1) we assume that E_n^x belongs to the interval $[-s_n^x, r_n^x]$, which is the real part of R_n.
2) E_{n+1}^x is equal to $E_n^x - \frac{1}{2}\ln\left[1 + d_n^x 2^{-n+1} + \left(d_n^{x\,2} + d_n^{y\,2}\right)2^{-2n}\right]$, so the value of E_{n+1}^x vs. E_n^x is given by various straight lines parameterized by d_n^x and d_n^y.
3) d_n^x must be such that for any possible value of d_n^y, $E_{n+1}^x \in \left[-s_{n+1}^x, r_{n+1}^x\right]$.

r_{n+1}^x must be the largest value of E_{n+1}^x corresponding to the straight line $d_n = 1$ (see Fig 1). That is to say, r_n^x must satisfy: $r_{n+1}^x = r_n^x - \ln(1 + 2^{-n})$. Since the length of R_n goes to zero as n goes to infinity, we deduce: $r_n^x = \sum_{k=n}^{\infty} \ln\left(1 + 2^{-k}\right)$. Similarly, the lowest possible value for E_n^x must correspond to the value obtained with $d_n = -1 \pm i$. This gives: $s_n^x = -\frac{1}{2}\sum_{k=n}^{\infty} \ln\left(1 - 2^{-k+1} + 2^{-2k+1}\right)$.

The terms $\bar{A}_n$, A_n, $\bar{B}_n$ and B_n appearing in the diagram shown in Fig. 1 are equal to:

$$\begin{aligned} \bar{A}_n &= r_{n+1}^x + \ln(1 - 2^{-n}) \\ \bar{B}_n &= -s_{n+1}^x + \frac{1}{2}\ln(1 + 2^{-2n}) \\ A_n &= -s_{n+1}^x + \frac{1}{2}\ln(1 + 2^{-n+1} + 2^{-2n+1}) \\ B_n &= r_{n+1}^x. \end{aligned}$$

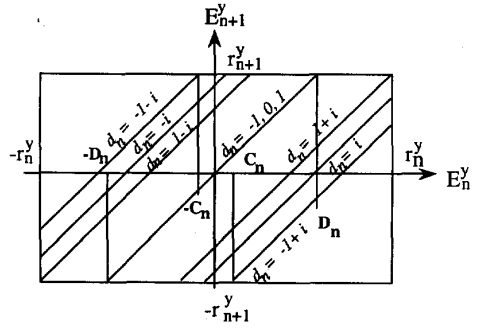


Fig. 2. The Robertson Diagram for E_n^y.

One can prove that $\bar{B}_n < \bar{A}_n$, and that $A_n < B_n$. From this, for any $E_n^x \in [-s_n^x, r_n^x]$, the following choices will give a value of E_{n+1}^x between $-s_{n+1}^x$ and r_{n+1}^x:

$$\begin{cases} \text{if} & E_n^x & < -\bar{B}_n & \text{then} & d_n^x = -1 \\ \text{if} & -\bar{B}_n & \le E_n^x < \bar{A}_n & \text{then} & d_n^x = -1 \text{ or } 0 \\ \text{if} & \bar{A}_n & \le E_n^x < A_n & \text{then} & d_n^x = 0 \\ \text{if} & A_n & \le E_n^x \le B_n & \text{then} & d_n^x = 0 \text{ or } 1 \\ \text{if} & B_n & < E_n^x & \text{then} & d_n^x = 1. \end{cases} \quad (4)$$

B. Choice of d_n^y

We use the relation $E_{n+1}^y = E_n^y - d_n^y \arctan\left(\frac{2^{-n}}{1+d_n^x 2^{-n}}\right)$. Fig. 2 shows the Robertson diagram associated to the choice of d_n^y. We want our choice to be independent of the choice of d_n^x. From this, we deduce: $r_n^y = \sum_{k=n}^{\infty} \arctan\left(\frac{2^{-k}}{1+2^{-k}}\right)$. The terms C_n and D_n appearing in the diagram are: $C_n = -r_{n+1}^y + \arctan\left(\frac{2^{-n}}{1-2^{-n}}\right)$ and $D_n = r_{n+1}^y$. One can prove that $C_n < D_n$. Thus, for any $E_n^y \in [-r_n^y, r_n^y]$, the following choices will give a value of E_{n+1}^y between $-r_{n+1}^y$ and $+r_{n+1}^y$:

$$\begin{cases} \text{if} & E_n^y < -D_n & \text{then} & d_n^y = -1 \\ \text{if} & -D_n \le E_n^y < -C_n & \text{then} & d_n^y = -1 \text{ or } 0 \\ \text{if} & -C_n \le E_n^y < C_n & \text{then} & d_n^y = 0 \\ \text{if} & C_n \le E_n^y \le D_n & \text{then} & d_n^y = 0 \text{ or } 1 \\ \text{if} & D_n < E_n^y & \text{then} & d_n^y = 1. \end{cases} \quad (5)$$

The convergence domain R_1 of the algorithm is:

$$-0.8298023738\cdots = -s_1^x \le E_1^x \le r_1^x = 0.8688766517\cdots$$
$$-0.749780302\cdots = -r_1^y \le E_1^y \le r_1^y = 0.749780302\cdots$$

C. The Algorithm

Relations (4) and (5) make it possible to find a sequence d_n such that, for $E_1 \in R_1$, $\lim_{n\to\infty} E_n = 0$. Now, let us try to simplify the choice of d_n: (4) and (5) involve comparisons that may require the examination of all the digits of the variables, we want to replace these comparisons by the examination of a small number of digits. The parameters $\bar{A} = -1/2$, $A = 1/4$, $C = 3/4$, $p_1 = 3$ and $p_2 = 4$ satisfy, for every n:

$$\begin{cases} 2^n\bar{B}_n & \le & \bar{A} - 2^{-p_1} & < & \bar{A} & \le & 2^n\bar{A}_n \\ 2^nA_n & \le & A & < & A + 2^{-p_1} & \le & 2^nB_n \\ 2^nC_n & \le & C & < & C + 2^{-p_2} & \le & 2^nD_n. \end{cases} \quad (6)$$

Therefore, if we denote $\tilde{E}_n^x$ the number obtained by truncating $2^nE_n^x$ after its p_1th fractional digit, and $\tilde{E}_n^y$ the number obtained by truncating $2^nE_n^y$ after its p_2^{th} fractional digit, we obtain, from (4), (5) and (6):

- if $\tilde{E}_n^x \le \bar{A} - 2^{-p_1}$ then $E_n^x \le \bar{A}_n$ therefore $d_n^x = -1$ is a valid choice
- if $\bar{A} \le \tilde{E}_n^x \le A$ then $\bar{B}_n \le E_n^x \le B_n$ therefore $d_n^x = 0$ is a valid choice[1]
- if $A + 2^{-p_1} \le \tilde{E}_n^x$ then $A_n \le E_n^x$ therefore $d_n^x = 1$ is a valid choice
- if $\tilde{E}_n^y \le -C - 2^{-p_2}$ then $E_n^y \le -C_n$ therefore $d_n^y = -1$ is a valid choice
- if $-C \le \tilde{E}_n^y \le C$ then $-D_n \le E_n^y \le D_n$ therefore $d_n^y = 0$ is a valid choice
- if $C + 2^{-p_2} \le \tilde{E}_n^y$ then $C_n \le E_n^y$ therefore $d_n^y = 1$ is a valid choice

From this, we deduce the E-mode of the BKM algorithm.

BKM Algorithm—E-mode

- Start with $E_1 \in R_1 = [-0.82980\cdots, +0.86887\cdots] + i.[-0.74978\cdots, +0.74978\cdots]$
- Iterate:

$$\begin{cases} L_{n+1} = L_n(1 + d_n 2^{-n}) \\ E_{n+1} = E_n - \ln(1 + d_n 2^{-n}) \end{cases}$$

with $d_n = d_n^x + id_n^y$, chosen as follows:

— define $\tilde{E}_n^x$ as the number obtained by truncating the real part of 2^nE_n after its 3rd fractional digit, and $\tilde{E}_n^y$ as the number obtained by truncating the imaginary part of 2^nE_n after its 4th fractional digit.

—

$$\begin{cases} \text{if} & \tilde{E}_n^x \le -\frac{5}{8} & \text{then} & d_n^x = -1 \\ \text{if} & -\frac{1}{2} \le \tilde{E}_n^x \le \frac{1}{4} & \text{then} & d_n^x = 0 \\ \text{if} & \frac{3}{8} \le \tilde{E}_n^x & \text{then} & d_n^x = 1 \\ \text{if} & \tilde{E}_n^y \le -\frac{13}{16} & \text{then} & d_n^y = -1 \\ \text{if} & -\frac{3}{4} \le \tilde{E}_n^y \le \frac{3}{4} & \text{then} & d_n^y = 0 \\ \text{if} & \frac{13}{16} \le \tilde{E}_n^y & \text{then} & d_n^y = 1. \end{cases}$$

- Result: $\lim_{n\to\infty} L_n = L_1 e^{E_1}$

In practice, instead of computing E_{n+1}=E_n $-$ $\ln(1 + d_n 2^{-n})$ and examining the first digits of $\alpha_n = 2^nE_n$, one would directly compute the sequence $\alpha_{n+1} = 2\alpha_n - 2^{n+1}\ln(1 + d_n 2^{-n})$.

D. Number of Iterations

Now, let us roughly estimate the number of iterations required to obtain a given accuracy. We want to compute $L_1 e^{E_1}$. The sequence d_i defined by the algorithm satisfies $L_1 e^{E_1} = L_1 \prod_{i=1}^{\infty}(1 + d_i 2^{-i})$. After n iterations of the E-mode, we have computed $L_1 \prod_{i=1}^{n}(1 + d_i 2^{-i})$. The relative error made by approximating $L_1 e^{E_1}$ by this value is $\left|1 - \frac{1}{\prod_{i=n+1}^{\infty}(1+d_i 2^{-i})}\right|$, which is bounded by a term equivalent to 2^{-n}. Therefore, after n iterations of the E-mode, we obtain a relative error approximately equal to 2^{-n}.

[1] Since $\bar{A}$, A, and $\tilde{E}_n^x$ have at most p_1 fractional digits, if $\tilde{E}_n^x > \bar{A} - 2^{-p_1}$, then $\tilde{E}_n^x \ge \bar{A}$.

E. Number of Constants Stored

This algorithm requires the storage of the constants $\ln(1 + d_i^x 2^{-i+1} + (d_i^{x2} + d_i^{y2})2^{-2i})$, $d_i^x, d_i^y = -1, 0, 1$ and $\arctan\left(\frac{2^{-i}}{1+d_i^x 2^{-i}}\right)$, $d_i^x = -1, 0, 1$. So, we need to store 9 terms for each value of i. Therefore, to obtain approximately n accuracy binary digits, we need to store $9n$ constants. This result can be improved by observing that if $i > n/2$, then $\ln(1+d_i^x 2^{-i+1}+(d_i^{x2}+d_i^{y2})2^{-2i})$ and $\arctan\left(\frac{2^{-i}}{1+d_i^x 2^{-i}}\right)$ can be replaced by $d_i^x 2^{-i+1}$ and 2^{-i} with accuracy 2^{-n}. Thus, we only need to store $\frac{9n}{2}$ constants.

III. Computation of the Logarithm Function (L-Mode)

As shown in the introduction, computing the logarithm of a complex number L_1 using BKM requires the calculation of a sequence d_n, $d_n = -1, 0, 1, -i, i, i-1, i+1, -i-1, -i+1$, such that $\lim_{n\to\infty} L_n = 1$, with $L_{n+1} = L_n(1 + d_n 2^{-n})$.

A. A Straightforward Strategy

In the following, we use the norm $\|.\|$ defined as $\|a+ib\| = \max\{|a|, |b|\}$. Let us define a sequence ϵ_n as $\epsilon_n = 2^n(L_n - 1)$. We deduce:

$$\epsilon_{n+1} = 2(\epsilon_n + d_n) + d_n\epsilon_n 2^{-n+1}. \tag{7}$$

If we find a sequence d_n such that the sequence ϵ_n is bounded, then L_n will go to 1. An intuitive solution is to choose $d_n \simeq -\epsilon_n$. So, in this section, we consider the following straightforward strategy which consists of building a sequence $\|\epsilon_n\| \leq 3/2$ as follows:

- at step i, we examine the value $\tilde{\epsilon}_i$ obtained by truncating the real and imaginary parts of ϵ_i after their p^{th} fractional digits, where p is a very small integer.
- d_i is obtained by rounding the real and imaginary parts of $-\tilde{\epsilon}_i$ to the nearest integer. Since p is small, this operation is easily performed. If $\|\epsilon_i\| \leq 3/2$, this choice will give $d_i \in D$.

If this algorithm actually gives $\|\epsilon_n\| \leq 3/2$ for any n, then L_n will go to 1. From $\|\tilde{\epsilon}_i - \epsilon_i\| \leq 2^{-p}$ and $\|d_i + \tilde{\epsilon}_i\| \leq 1/2$, using (7), we deduce:

$$\|\epsilon_{n+1}\| \leq 1 + 2^{1-p} + 2^{-n+1}\|d_n\epsilon_n\|. \tag{8}$$

The norm $\|\cdot\|$ satisfies $\|zz'\| \leq 2\|z\|.\|z'\|$, therefore:

$$\|\epsilon_{n+1}\| \leq 1 + 2^{1-p} + 2^{-n+2}\|\epsilon_n\|. \tag{9}$$

If $n \geq 4$, $p \geq 4$, and if $\|\epsilon_n\| \leq 3/2$, then, using (9), one can prove that for any $k \geq n$, $\|\epsilon_k\| \leq 3/2$. Therefore, if we start the iteration (7) at step 4, from ϵ_4 satisfying $\|\epsilon_4\| \leq 3/2$, then the strategy presented above will give a sequence d_n which fulfills $\lim_{n\to\infty} L_n = 1$. This strategy allows computation of the logarithm in a very tiny domain: we can use it to compute the logarithm of L_4 such that $\|\epsilon_4\| = \|16(L_4 - 1)\| \leq 3/2$. So the convergence domain of this algorithm is $L_4 \in \left[1 - \frac{3}{32}, 1 + \frac{3}{32}\right] + i.\left[-\frac{3}{32}, +\frac{3}{32}\right]$.

B. Computation of the Logarithm in a Larger Domain

As in Section III-A, we study the sequence $\epsilon_{k+1} = 2(\epsilon_k + d_k) + d_k\epsilon_k 2^{-k+1}$, where ϵ_k is defined as $2^k(L_k - 1)$. Our purpose is to start the evaluation at step $k = 1$, with ϵ_1 belonging to a domain T that will be given later, and to obtain, after a few steps, say n steps ($n \geq 3$), a value ϵ_{n+1} satisfying $\epsilon_{n+1} \leq 3/2$. After this, the strategy presented in the previous section can be used. Our goal is to obtain a convergence domain $L_1 \in \frac{1}{2}T + 1$ larger than the previous one. The following algorithm was found through simulations, before being proved.

BKM Algorithm—L-Mode:

- Start with L_1 belonging to the trapezoid T delimited by the straight lines $x = 1/2$, $x = 1.3$, $y = x/2$, $y = -x/2$. T is the domain where the convergence is proven, but experimental tests show that the actual convergence domain of the algorithm is larger.
- Iterate: $\begin{cases} L_{n+1} = L_n(1 + d_n 2^{-n}) \\ E_{n+1} = E_n - \ln(1 + d_n 2^{-n}) \end{cases}$
  with $d_n = d_n^x + i d_n^y$, chosen as follows:
  - define ϵ_n^x and ϵ_n^y as the real and imaginary parts of $\epsilon_n = 2^n(L_n - 1)$, and $\tilde{\epsilon}_n^x$ and $\tilde{\epsilon}_n^y$ as the values obtained by truncating these numbers after their 4th fractional digits.
  - At step 1, we have the equation found at the bottom of the page.
  - At step n, $n \geq 2$

$$\begin{cases} \text{if} & \tilde{\epsilon}_n^x \leq -1/2 & \text{then} & d_n^x = 1 \\ \text{if} & -1/2 < \tilde{\epsilon}_n^x < 1/2 & \text{then} & d_n^x = 0 \\ \text{if} & 1/2 \leq \tilde{\epsilon}_n^x & \text{then} & d_n^x = -1 \end{cases}$$

$$\begin{cases} \text{if} & \tilde{\epsilon}_n^y \leq -1/2 & \text{then} & d_n^y = 1 \\ \text{if} & -1/2 < \tilde{\epsilon}_n^y < 1/2 & \text{then} & d_n^y = 0 \\ \text{if} & 1/2 \leq \tilde{\epsilon}_n^y & \text{then} & d_n^y = -1 \end{cases}$$

- result: $\lim_{n\to\infty} E_n = E_1 + \ln(L_1)$.

In a practical implementation, instead of computing L_n and examining the first digits of $\epsilon_n = 2^n(L_n - 1)$, one would directly compute the sequence ϵ_n using (7).

Proof of the Algorithm: Our goal is to show that if $L_1 \in T$, then there exists $n \geq 4$ such that $\|\epsilon_n\| \leq 3/2$. Thus the proof of section 3.1 will hold. In order to prove

$$\begin{cases} \text{if} & \tilde{\epsilon}_1^x \leq -7/16 & \text{and} & 6/16 \leq \tilde{\epsilon}_1^y & \text{then} & d_1 = 1 - i \\ \text{if} & \tilde{\epsilon}_1^x \leq -7/16 & \text{and} & \tilde{\epsilon}_1^y \leq -6/16 & \text{then} & d_1 = 1 + i \\ \text{if} & -6/16 \leq \tilde{\epsilon}_1^x & \text{and} & 8/16 \leq \tilde{\epsilon}_1^y & \text{then} & d_1 = -i \\ \text{if} & -6/16 \leq \tilde{\epsilon}_1^x & \text{and} & \tilde{\epsilon}_1^y \leq -9/16 & \text{then} & d_1 = i \\ \text{if} & \tilde{\epsilon}_1^x \leq -7/16 & \text{and} & -5/16 \leq \tilde{\epsilon}_1^y \leq 5/16 & \text{then} & d_1 = 1 \\ \text{if} & -6/16 \leq \tilde{\epsilon}_1^x & \text{and} & -1/2 \leq \tilde{\epsilon}_1^y \leq 1/2 & \text{then} & d_1 = 0 \end{cases}$$

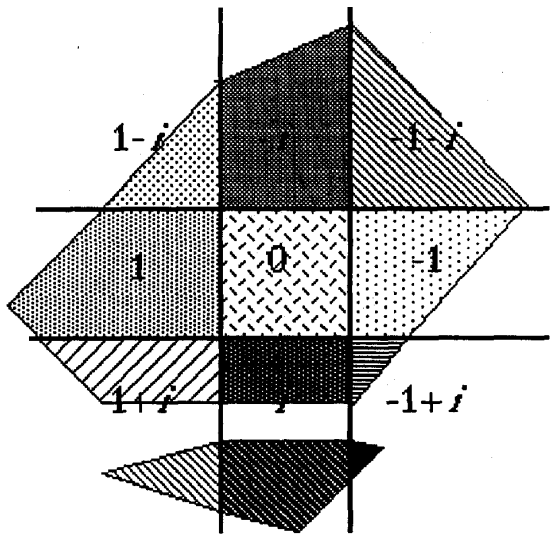


Fig. 3. β_k is splitinto convex polygons following the d-areas.

this, we build a sequence β_k of *bounding sets*, such that for any $L_1 \in T$, $\epsilon_k \in \beta_k$. Our problem is reduced to show that there exists $n \geq 4$ such that β_n is included in the square $\|z\| \leq 3/2$. Let us explain how the sequence β_k is computed.

The first bounding set β_1 is equal to $2(T-1)$. At step k, β_k is an aggregate of convex polygons, represented by their vertices. A step of the algorithm can be represented by a splitting of the complex plane into 9 convex *d-areas*. The d-area associated with $\delta \in \{-1, 0, 1, -i, i, i-1, i+1, -i-1, -i+1\}$ is the domain $D(\delta)$ such that if $\epsilon_k \in D(\delta)$, then the algorithm gives $d_k = \delta$. For instance, if $k \geq 2$, then $D(-1-i)$ is the set of the complex numbers whose real and imaginary parts are greater than $1/2$. In $D(\delta)$, the transformation $\epsilon_{k+1} = 2(\epsilon_k + \delta) + \delta\epsilon_k 2^{-k+1}$ is a *similarity*, i.e., the combination of a rotation and the multiplication by a real factor. A similarity transforms a convex polygon into another convex polygon.

Each convex polygon of β_k is split into sub-convex polygons, obtained by intersecting it with the d-areas. Fig. 3 presents the bounding step at step k, the various d-areas (for $k \geq 2$), and the splitting of the polygons of β_k.

Broadly speaking, β_{k+1} is obtained by computing the transformation of each sub-convex polygon obtained after the splitting (the image of a polygon is obtained by computing the image of its vertices). However, we have to take into account the fact that the choice of d_k is based on the examination of $\tilde{\epsilon}_k^x$ and $\tilde{\epsilon}_k^y$, which are obtained by truncating the real and imaginary parts of ϵ_k after their 4th fractional digits. For instance, if $\tilde{\epsilon}_k = \tilde{\epsilon}_k^x + i\tilde{\epsilon}_k^y$ belongs to $D(-1)$, this does not prove that ϵ_k actually belongs to $D(-1)$. Therefore, to each sub-convex polygon, a "ribbon" of length 2^{-4} is added, so that if $\tilde{\epsilon}_k$ belongs to the "old" subconvex polygon, then ϵ_k belongs to the "new" sub-convex polygon. After this, for each "new" sub-polygon, we compute its image by the similarity defined by the value of d_k associated with the polygon (Fig. 4). This gives the new bounding set β_{k+1}.

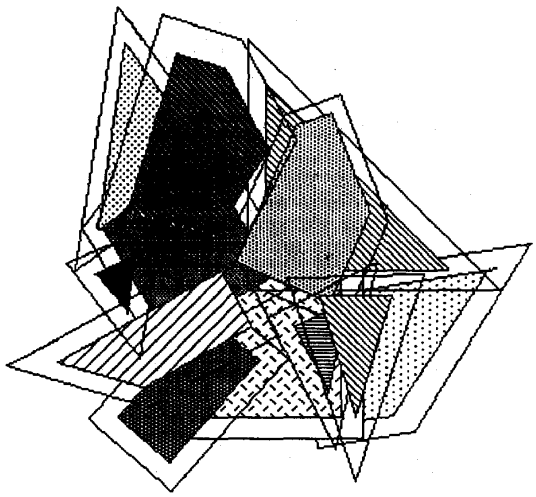


Fig. 4. The iteration is applied to each of the vertices of the convex polygons.

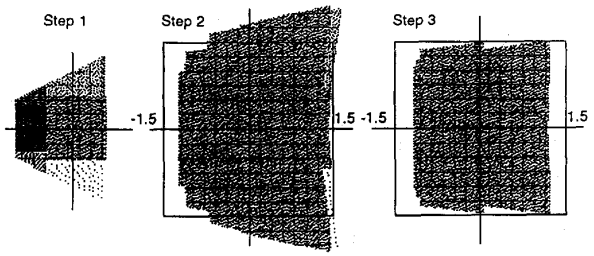


Fig. 5. The initial domain β_1 and the bounding sets β_4 andβ_6.

The proof by induction that for any $L_1 \in T$, $\epsilon_k \in \beta_k$ is straightforward. If we find $n \geq 4$ such that all the vertices of the sub-convex polygons of β_n are in the square $\|z\| \leq 3/2$, then the algorithm is proven. The adequate value of n is 6: this leads to a number of vertices which is much too large to be examined by a paper-and-pencil method. We have used a program for computing all the vertices of β_6, this program is written in ML, and uses exact rational arithmetic. Fig. 5 shows the bounding sets β_1, β_4, and β_6. Using this program, we have verified that all the vertices of β_6 are included in the square $\|z\| \leq 3/2$.

C. Number of Iterations

Let us estimate the number of iterations required to obtain a given accuracy. The sequence d_i satisfies $\ln(L_1) = -\sum_{k=1}^{\infty} \ln(1 + d_k 2^{-k})$. After n iterations of the L-mode, we have computed $E_1 - \sum_{k=1}^{n} \ln(1 + d_k 2^{-k})$. The absolute error made by approximating $E_1 + \ln(L_1)$ by this value is $\sum_{k=n+1}^{\infty} \ln(1 + d_k 2^{-k})$, which is bounded by a term equivalent to $2^{-n}\sqrt{2}$. From this we deduce that, to obtain an absolute error equal to 2^{-n}, one needs to perform $n + 1$ iterations.

IV. Range Reduction

The algorithms for computing elementary functions generally converge in some bounded domain. For computing $f(x)$ with an arbitrary value x, one usually needs to find a value

x^* belonging to the convergence domain of the algorithm that computes f, such that $f(x)$ can be deduced from $f(x^*)$. This operation is called *range reduction.*

A. Range Reduction for the Complex Exponential Function

We assume that, if I is an interval containing zero, whose length is greater than ρ, we can compute, from any real x, an integer k such that $x - k\rho \in I$. This can be done by performing a few steps of a SRT-like division algorithm. Assume that we want to compute e^{x+iy}. BKM allows the evaluation of the exponential function in $R_1 = [-0.82980237\cdots, +0.86887665\cdots] + i.[-0.749780302\cdots, +0.749780302\ldots]$. The range reduction can be performed as follows:

1) Compute k_y such that $y - k_y.\frac{\pi}{4}$ belongs to $[-r_1^y, r_1^y]$. Define y^* as $y - k_y.\frac{\pi}{4}$.
2) We have: $e^{x+iy} = e^{i.(k_y \bmod 8)\frac{\pi}{4}} e^{x+iy^*}$. The multiplication by $e^{i.(k_y \bmod 8)\frac{\pi}{4}}$ looks difficult to reduce to a small amount of additions and shifts. Fortunately, this problem is easily overcame. As an example, let us consider the case $k_y \bmod 8 = 1$. The term $e^{\frac{i\pi}{4}}$ is equal to $\frac{\sqrt{2}}{2}(1+i)$. A multiplication by this term is avoided by adding $-\frac{1}{2}\ln(2) = \ln\frac{\sqrt{2}}{2}$ to x, which gives a value x', and then obtaining: $e^{x+iy} = (1+i)e^{x'+iy^*}$. A multiplication by $1+i$ is easily reduced to two additions. A similar trick can be used for the other possible values of $k_y \bmod 8$. So, if we define K_p and γ_p as follows:

$$\begin{cases} K_0 = 1 & \text{and} & \gamma_0 = 0 \\ K_1 = 1+i & \text{and} & \gamma_1 = -\frac{1}{2}\ln(2) \\ K_2 = i & \text{and} & \gamma_2 = 0 \\ K_3 = -1+i & \text{and} & \gamma_3 = -\frac{1}{2}\ln(2) \\ K_4 = -1 & \text{and} & \gamma_4 = 0 \\ K_5 = -1-i & \text{and} & \gamma_5 = -\frac{1}{2}\ln(2) \\ K_6 = -i & \text{and} & \gamma_6 = 0 \\ K_7 = 1-i & \text{and} & \gamma_7 = -\frac{1}{2}\ln(2) \end{cases}$$

then, with $p = k_y \bmod 8$ and $x' = x + \gamma_p$, we get $e^{x+iy} = K_p e^{x'+iy^*}$.

3) Compute k_x such that $x' - 2k_x\ln(2)$ belong to $[-s_1^x, r_1^x]$. Then we obtain $e^{x+iy} = 2^{2k}K_p e^{x^*+iy^*}$.
The exponential of $x^* + iy^*$ is computed using the E-mode of BKM, and the multiplication by $2^{2k}K_p$ is reduced to two additions and a shift.

B. Range Reduction for the Complex Logarithm Function

Let us define the cone C_0 as the set of the numbers $x+iy$ such that $|y| \le x/2$ (see Fig. 6), and $C_1, C_2, C_3, C_4, C_5, C_6$, and C_7 as $C_k = C_0 e^{\frac{ik\pi}{4}}$. For each nonzero element z of C_0 there exists an integer n such that $2^n C_0$ belongs to the convergence domain T of the L-mode. Since $C_0 = \lambda C_0$ for any nonnegative real λ, we easily find:

$$\begin{array}{ll} C_1 = (1+i)C_0 = \frac{C_0}{1-i} & C_5 = (-1-i)C_0 = \frac{C_0}{-1+i} \\ C_2 = iC_0 = \frac{C_0}{-i} & C_6 = -iC_0 = \frac{C_0}{i} \\ C_3 = (-1+i)C_0 = \frac{C_0}{-1-i} & C_7 = (1-i)C_0 = \frac{C_0}{1+i}. \\ C_4 = -C_0 & \end{array}$$

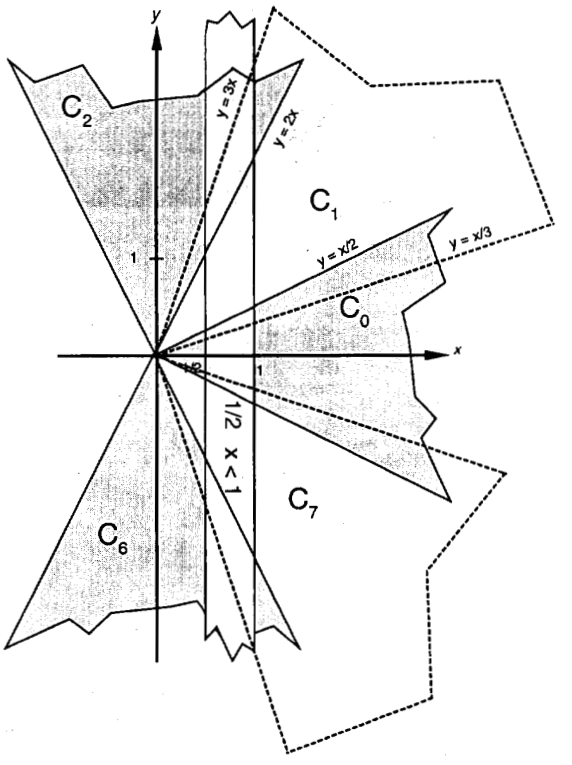


Fig. 6. Second step of the reduction.

Assume that we want to compute the logarithm of $z_{\text{init}} = x_{\text{init}} + iy_{\text{init}}$. In order to do this, from z_{init} we want to obtain $z_{\text{Bkm}} = 2^k(d^x + id^y)z_{\text{init}}$, with $d^x, d^y = -1, 0, 1$ and d^x, d^y not simultaneously equal to zero, such that z_{Bkm} belongs to T. After this, $\ln(z_{\text{Bkm}})$ is computed using the L-mode, then $k\ln(2) + \ln(d^x + id^y)$ is subtracted from the result (we just need to store the 8 possible values of $\ln(d^x + id^y)$). This reduction is performed in three steps:

Prescaling: We find k_1 such that $z = x + iy = \pm 2^{k_1} z_{\text{init}}$ satisfies $\frac{1}{2} \le x < 1$.

Search for a Cone C_p Containing z: Now, let us find p such that $z \in C_p$. Since $x > 1/2$, the only possible values of p are 0, 1, 2, 6, 7. From the definition of the cones C_p, we easily find (see Fig. 6):

$$\begin{array}{lll} \text{If} & |y| \le x/2 & \text{then} \quad z \in C_0 \\ \text{If} & -3x \le y \le -x/3 & \text{then} \quad z \in C_7 \\ \text{If} & x/3 \le y \le 3x & \text{then} \quad z \in C_1 \\ \text{If} & y \le -2x & \text{then} \quad z \in C_6 \\ \text{If} & y \ge 2x & \text{then} \quad z \in C_2. \end{array}$$

By taking into account the overlapping of the cones C_p, we can replace these comparisons by comparisons involving only a few digits of x and y. If we define $\tilde{x}$ and $\tilde{y}$ as the numbers obtained by truncating x and y after their 5^{th} fractional digits (as previously, we assume a binary number system), we easily deduce: $|x - \tilde{x}| \le 1/32$ and $|y - \tilde{y}| \le 1/32$. Therefore we have the following.

1) If $|\tilde{y}| < \frac{\tilde{x}}{2} - \frac{1}{32}$ then $|y| \le \frac{x}{2}$, therefore, $z \in C_0$.

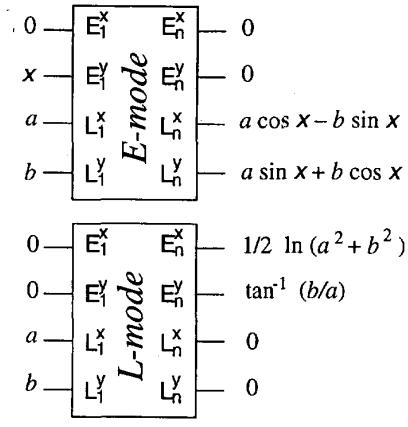


Fig. 7. 2–D rotations and arctangents.

2) If $\frac{\tilde{x}}{2} - \frac{1}{32} \leq \tilde{y} \leq 2\tilde{x} + \frac{1}{16}$ then:
   a) $y \geq \frac{x}{2} - \frac{1}{32} - \frac{1}{64}$. Since $x \geq \frac{1}{2}$, $(\frac{x}{2} - \frac{x}{3}) = \frac{x}{6} \geq \frac{1}{12}$. Therefore, $y \geq \frac{x}{3}$.
   b) $y \leq 2x + \frac{1}{16} + \frac{1}{32}$. Since $x \geq \frac{1}{2}$, $3x + (2x - 3x) + \frac{1}{16} + \frac{1}{32}$ is less than $3x + \frac{1}{16} + \frac{1}{32} - \frac{1}{2}$ therefore, $y \leq 3x - \frac{13}{32} < 3x$.

   therefore $z \in C_1$
3) If $-2\tilde{x} - \frac{1}{16} \leq \tilde{y} \leq -\frac{\tilde{x}}{2} + \frac{1}{32}$, therefore, in a similar fashion, $z \in C_7$.
4) If $\tilde{y} > 2\tilde{x} + \frac{1}{16}$ then $y \geq 2x$, therefore $z \in C_2$
5) $\tilde{y} < -2\tilde{x} - \frac{1}{16}$ then, in a similar fashion, $z \in C_6$.

Postscaling: We have found a number $z = x + iy$, an integer k_1, a sign $s = \pm 1$ and a cone $C_p, p = 0, 1, 2, 6, 7$ satisfying $z = s.2^{k_1} z_{\text{init}}$, $\frac{1}{2} \leq x \leq 1$ and $z \in C_p$. Now, define ρ_k and ℓ_k as:

$$\begin{aligned} \rho_0 &= 1 & \rho_6 &= 1 + i \\ \rho_1 &= 1 - i & \rho_7 &= i \\ \rho_2 &= -i & \ell_k &= \ln \rho_k, k = 0, 1, 2, 6, 7. \end{aligned}$$

The number $z' = x' + iy' = \rho_p z$ belongs to C_0. If $p \neq 0$, z' does not necessarily belong to the convergence domain T of the L-mode (which is obtained by intersecting C_0 with the domain $\frac{1}{2} \leq x \leq 1.3$). Define an integer k_2 such that $\frac{1}{2} \leq 2^{k_2} x' \leq 1.3$. Then $z_{\text{Bkm}} = 2^{k_2} z' \in T$. Define ℓ as ℓ_p if $s = 1$ and $\ell_p + i\pi$ if $s = -1$. The logarithm L_{Bkm} of z_{Bkm} is obtained using the L-mode, then $\ell + (k_1 + k_2).\ln(2)$ is subtracted to L_{Bkm}, in order to obtain the logarithm of z_{init}. The imaginary part of the final result will be between -2π and 2π, so a "correction step"—consisting in adding $\pm\pi$ or $\pm 2\pi$—may be performed if a particular range is required.

V. Computation of Elementary Functions

As shown in the previous sections, BKM makes it possible to compute the following functions:

- in E-mode, $L_1 e^{E_1}$, where E_1 is a complex number belonging to R_1,
- in L-mode, $E_1 + \ln(L_1)$, where L_1 belongs to the trapezoid T.

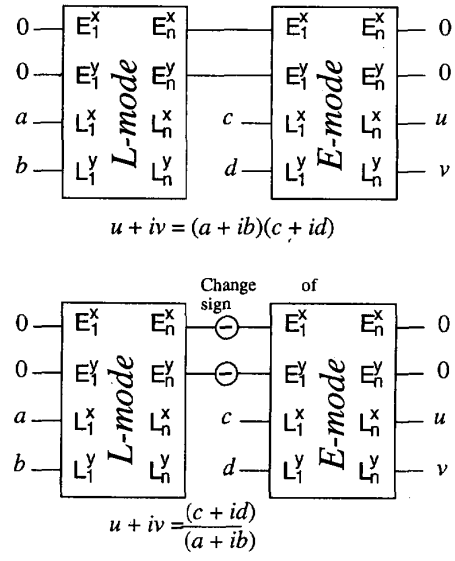


Fig. 8. Complex multiplications and divisions.

Therefore, using BKM, one can compute the following functions of real variables.

A. Functions Computable Using One Mode of BKM

1) *Real sine and cosine Functions*: In the E-mode of BKM, one can compute the exponential of $E_1 = 0 + i\theta$, and obtain $L_n = \cos\theta + i \sin\theta \pm 2^{-n}$.
2) *Real Exponential Function*: If E_1 is a real number belonging to $[-0.8298023738, +0.8688766517]$, the E-mode will give a value L_n equal to $L_1 e^{E_1} \pm 2^{-n}$.
3) *Real Logarithm:* If L_1 is a real number belonging to [0.5, 1.3], the E-mode will give a value E_n equal to $E_1 + \ln(L_1) \pm 2^{-n}$. Furthermore, in this case, the BKM iteration is reduced to Brigg's algorithm, and it is possible to show that the algorithm gives a correct result if and only if $L_1 \in [\prod_{n=1}^{\infty}(1 + 2^{-n})^{-1}, \prod_{n=2}^{\infty}(1 - 2^{-n})^{-1}] \simeq [0.419422, 1.73137]$.
4) *2-D Rotations:* As pointed out in [11], performing rotations is useful for Fast Fourier Transformation, Digital Filtering, and Matrix Computations. The 2-D vector $(c\ d)^t$ obtained by rotating $(a\ b)^t$ of an angle θ satisfies: $c + id = (a + ib)e^{i\theta}$ therefore, $(c\ d)^t$ is computed using the E-mode, with $L_1 = a + ib$ and $E_1 = i\theta$ (see Fig. 7).
5) *Real* **arctan** *Function:* From the relation:

$$\ln(x + iy) = \begin{cases} \frac{1}{2}\ln(x^2 + y^2) + i \arctan \frac{y}{x} \bmod (2i\pi) & \text{if } x > 0 \\ \frac{1}{2}\ln(x^2 + y^2) + i(\pi + \arctan\frac{y}{x}) \bmod (2i\pi) & \text{if } x < 0 \end{cases}$$

one can easily deduce that, if $x + iy$ belongs to the convergence domain of the L-mode of BKM, then $\arctan y/x$ is the limit value of the imaginary part of E_n, assuming that the L-mode is used with $E_1 = 0$ and $L_1 = x + iy$ (see Fig. 7).

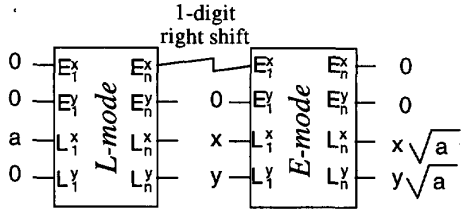


Fig. 9. Computation of $x\sqrt{a}$ and $y\sqrt{a}$ in parallel.

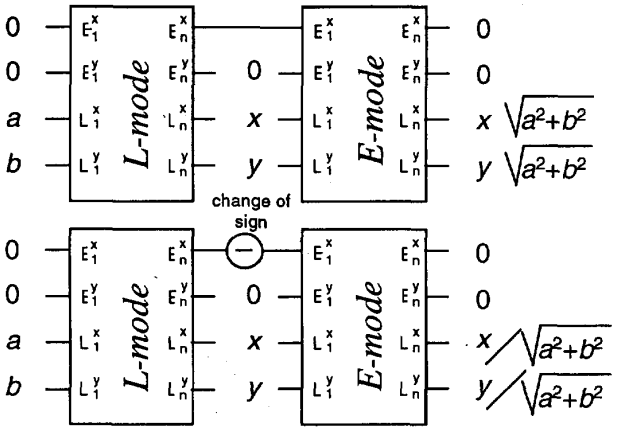


Fig. 10. Computation of lengths and normalization.

B. Functions Computable Using Two Consecutive Modes of BKM

Using two BKM operations, one can compute many functions. Some of these functions are the following.

1) *Complex Multiplication and Division:* The product zt is evaluated as $z.e^{\ln t}$, while z/t is evaluated as $z.e^{-\ln t}$ (see Fig. 8). One can compute $(ab)e^z$ or $(\frac{a}{b})e^z$, where a, b and z are complex numbers, using the same operator, by choosing E_1^x equal to the real part of z, and E_1^y equal to the imaginary part of z.
2) *Computation of $x\sqrt{a}$ and $y\sqrt{a}$ in Parallel* (x, y and a are **real** numbers): we use the relation $\sqrt{a} = e^{\frac{1}{2}\ln(a)}$, see Fig. 9. One can also compute $\frac{x}{\sqrt{a}}$ and $\frac{y}{\sqrt{a}}$.
3) *Computation of Lengths and Normalization of 2-D Vectors*: As shown previously, the L-mode allows the computation of $F = \frac{1}{2}\ln(a^2+b^2) = \ln\sqrt{a^2+b^2}$, where a and b are real numbers. Using the E-mode, we can compute e^F or e^{-F} (See Fig. 10).

VI. Comparison with CORDIC

In order to obtain p significant bits, CORDIC and BKM roughly need p iterations. BKM requires more hardware than CORDIC: BKM needs the storage of $\frac{9p}{2}$ constants, while CORDIC needs the storage of p constants to compute trigonometric and hyperbolic functions. Since these constants are represented by p digits, both algorithms need a $O(p^2)$ area for storage of the constants. Both algorithms need a shifter able to perform an n-position shift at step n. A barrel shifter makes it possible to perform a n-position shift (for any $n \leq p$) in constant time, and lies in an area $O(n^2)$. Since the area complexity of most adders is better than $O(n^2)$, the area complexity of CORDIC and BKM is $O(n^2)$. The computations performed during a BKM iteration are:

- For the variable:

$$E_n \begin{cases} \alpha_{n+1}^x = 2\alpha_n^x - 2^n \ln\left[1 + d_n^x 2^{-n+1} + \left(d_n^{x\,2} + d_n^{y\,2}\right)2^{-2n}\right] \\ \alpha_{n+1}^y = 2\alpha_n^y - 2^{n+1} d_n^y \arctan\left(\frac{2^{-n}}{1+d_n^x 2^{-n}}\right) \end{cases}$$

  if, instead of computing E_n and examining the first digits of $\alpha_n = 2^n E_n$, we directly compute α_n.
- For the variable:

$$L_n \begin{cases} \epsilon_{n+1}^x = 2(\epsilon_n^x + d_n^x) + (d_n^x \epsilon_n^x - d_n^y \epsilon_n^y)2^{-n+1} \\ \epsilon_{n+1}^y = 2(\epsilon_n^y + d_n^y) + (d_n^y \epsilon_n^x + d_n^x \epsilon_n^y)2^{-n+1} \end{cases}$$

  if, instead of computing L_n and examining the first digits of $\epsilon_n = 2^n(L_n - 1)$, we directly compute ϵ_n.

So BKM looks more complicated than CORDIC. As a matter of fact, in order to compare CORDIC and BKM, we have to assume that we use a redundant number system. Using such a system, the time complexities of both algorithms are $O(p)$. As pointed out in many papers dealing with CORDIC, efficient use of CORDIC with such a number system requires a doubling of the iterations in space [7] or in time [16], [1]. Doubling the CORDIC iterations in time gives:

$$\begin{cases} x_{n+1} = x_n - d_n y_n 2^{-n} - d_n^2 x_n 2^{-2n-2} \\ y_{n+1} = y_n + d_n x_n 2^{-n} - d_n^2 y_n 2^{-2n-2} \\ z_{n+1} = z_n - 2d_n \arctan 2^{-n-1}. \end{cases} \tag{10}$$

This iteration is at least as complex as the BKM iteration (at step n one needs to perform an n-position shift and a $2n$ − 2-position shift: this requires a larger shifter, or several consecutive shifts). Doubling the iterations in space requires more control: in the *branching* CORDIC method [7], one needs to compare at each step the values given by two CORDIC modules. Furthermore, doubling iterations makes it possible to obtain a constant scale factor, but this scale factor remains different from 1, therefore, for computing many functions, one needs to perform a multiplication after the CORDIC iterations. So, although both methods have the same time and space complexities, BKM looks more interesting when using a redundant number system.

VII. Conclusion

We have proposed a new algorithm for the computation of many elementary functions (complex exponential and logarithm functions, complex multiplication, complex functions $(ab)e^z$ and $(\frac{a}{b})e^z$, real functions sin, cos, $\arctan\frac{y}{x}$, $\ln(x^2+y^2)$, $x\sqrt{a}, x\sqrt{a^2+b^2}, x/\sqrt{a^2+b^2}$, and 2-D rotations). This algorithm matches the CORDIC algorithm, since it allows the use of a redundant number system without any scale factor problem and allows the computation of more functions.

Acknowledgment

We thank C. Paulin for her precious knowledge of *ML* and her careful reading of the manuscript.

REFERENCES

[1] T. Asada, N. Takagi, and S. Yajima, "Redundant cordic methods with a constant scale factor," *IEEE Trans. Comput.*, vol. 40, no. 9, pp. 989–995, Sept. 1991.

[2] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Elect. Comput.*, vol. 10, pp. 389–400, 1961. Reprinted in *Computer Arithmetic*, vol. 2, E.E. Swartzlander, Ed. CA: IEEE Computer Society Press Tutorial, 1990.

[3] W. P. Burleson, "Polynomial evaluation in VLSI using distributed arithmetic," *IEEE Trans. Circuits and Syst.*, vol. 37, no. 10, 1990.

[4] T. C. Chen, "Automatic computation of logarithms, exponentials, ratios and square roots," *IBM J. Res. and Dev.*, vol. 16, pp. 380–388, 1972.

[5] W. Cody and W. Waite, *Software Manual for the Elementary Functions*. . Englewood Cliffs, NJ: Prentice-Hall, 1980.

[6] J. Duprat and J. M. Muller, "Hardwired polynomial evaluation," *J. Parallel and Distrib. Comput.*, Special Issue on *Parallelism in Computer Arithmetic*, vol. 5, 1988.

[7] ——, "The cordic algorithm: New results for fast VLSI implementation," *IEEE Trans. Comput.*, vol. 42, no. 2, pp. 168–178, Feb. 1993.

[8] M. D. Ercegovac, "A general method for evaluation of functions and computation in a digital computer," Ph.D. Thesis, Dept. of Computer Sci., Univ. of Illinois, Urbana-Champaign, 1975.

[9] M. D. Ercegovac, "A general hardware-oriented method for evaluation of functions and computations in a digital computer," *IEEE Trans. Comput.*, vol. C-26, no. 7, pp. 667–680, 1977.

[10] J. F. Hart, *Computer Approximations*. New York: Wiley, 1968.

[11] Y. H. Hu, "Cordic-based VLSI architectures for digital signal processing," *IEEE Signal Processing Mag.*, 1992.

[12] B. De Lugish, "A class of algorithms for automatic evaluation of functions and computations in a digital computer," Ph.D. Thesis, Dept. of Comput. Sci., Univ. of Illinois, Urbana, 1970.

[13] J. M. Muller, "Discrete basis and computation of elementary functions," *IEEE Trans. Comput.*, vol. C-34, no. 9, pp. 857–862, Sept., 1985.

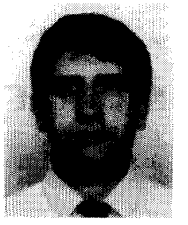
[14] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Elec. Comput.*, vol. EC-7, pp. 218–222, 1958. Reprinted in *Computer Arithmetic*, vol. 1, E.E. Swartzlander, Ed. CA: IEEE Computer Society Press Tutorial, 1990.

[15] W. H. Specker, "A class of algorithms for $\ln(x)$, $\exp(x)$, $\sin(x)$, $\cos(x)$, $\tan^{-1}(x)$ and $\cot^{-1}(x)$," *IEEE Trans. Elect. Comput.*, vol. EC-14, 1965. Reprinted in *Computer Arithmetic*, vol. 1, E.E. Swartzlander, Ed. CA: IEEE Computer Society Press Tutorial, 1990.

[16] N. Takagi, T. Asada, and S. Yajima, "A hardware algorithm for computing sine and cosine using redundant binary representation (in Japanese)," *Trans. IECE Japan*, vol. J69-D, no. 6, pp. 841–847, June 1986. English translation available in *Systems and Computers* in Japan, vol. 18 no. 8, pp. 1–9, Aug. 1987.

[17] J. Volder, "The cordic computing technique," *IRE Trans. Elect. Comput.*, 1959. Reprinted in *Computer Arithmetic*, vol. 1, E.E. Swartzlander, Ed. CA: IEEE Computer Society Press Tutorial, 1990.

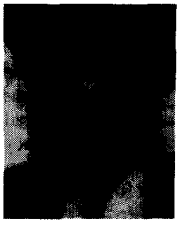
[18] J. Walther, "A unified algorithm for elementary functions," in *Joint Comput. Conf. Proc.*, 1971. Reprinted in *Computer Arithmetic*, vol. 1, E.E. Swartzlander, Ed. CA: IEEE Computer Society Press Tutorial, 1990.



Jean-Claude Bajard was born in Saint-Etienne, France, in 1957. He received the Ph.D. degree in computer science in 1993 from the Ecole Normale Supérieure de Lyon.

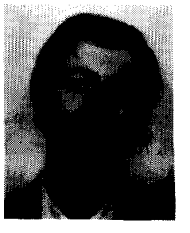
He taught mathematics in high school from 1979 to 1990, and served as an assistant professor at Ecole Normale Supérieure de Lyon in 1993. He joined the Lab. LIM, Université de Provence, Marseille, France in 1993.

Dr. Bajard's research interests include computer arithmetic and VLSI design.



Sylvanus Kla was born in Ivory Coast in 1963. He received the Engineer degree in civil engineering from Ecole Nationale Supérieure des Travaux Publics, Yamoussoukro, Ivory Coast, in 1988, the M.S. degree in applied mathematics from Grenoble University, France, in 1989, and the Ph.D. degree from Lyon University, France, in 1993.

His research interests are in computer arithmetic and computer architecture.



Jean-Michel Muller (M'87) was born in Grenoble, France, in 1961. He received the Engineer degree in applied mathematics and computer science in 1983, and the Ph.D. in computer science in 1985, both from the Institut National Polytechnique de Grenoble, France.

He was posted from 1986 to 1989 with Tim3–Imag Laboratory, Grenoble. He is currently with CNRS, Lab. LIP, Ecole Normale Supérieure de Lyon, Lyon. He teaches computer arithmetic in the Ecole Normale Supérieure de Lyon. His research interests include computer arithmetic and computer architecture.

Dr. Muller served as General Chairman of the 10th Symposium on Computer Arithmetic (Grenoble, France, June, 1991).