position of A and B is involved; the other $n-1$ bits are arbitrary. Hence, $S_1 = 2 \cdot 4^{n-1}$. For $i > 1$, the leftmost 2^{i-2} bits must generate $ESG = 100$ (equal), and the other 2^{i-2} bits of the 2^{i-1} involved bits must generate $ESG = 010$ or 001 (smaller than or greater than), since otherwise the completion signal would be generated either before or after the ith level. The 2^{i-2} bits generating $ESG = 100$ correspond to $2^{2^{i-2}}$ configurations. The other 2^{i-2} bits correspond to $(4^{2^{i-2}} - 2^{2^{i-2}})$ configurations since the total number of configurations is $4^{2^{i-2}}$, among which $2^{2^{i-2}}$ correspond to $ESG = 100$. Thus, for $i > 1$:

$$S_i = 2^{2^{i-2}}(4^{2^{i-2}} - 2^{2^{i-2}})4^{n-2^{i-1}} = 4^n(2^{-2^{i-2}} - 2^{-2^{i-1}}).$$

The average delay through P- and I-modules (in unit d)

$$\begin{aligned} D &= \left\{S_1 + \sum_{i=2}^{k+1} iS_i\right\} \Big/ 4^n \\ &= \left\{2 * 4^{n-1} + \sum_{i=2}^{k+1} i4^n(2^{-2^{i-2}} - 2^{-2^{i-1}})\right\} \Big/ 4^n \\ &= 2^{-1} + \sum_{i=2}^{k+1} i(2^{-2^{i-2}} - 2^{-2^{i-1}}) \\ &< 2^{-1} + \sum_{i=2}^{\infty} i(2^{-2^{i-2}} - 2^{-2^{i-1}}) \\ &= 2^{-1} + \sum_{i=1}^{\infty} (i+1)2^{-2^{i-1}} - \sum_{i=2}^{\infty} i2^{-2^{i-1}} \\ &= 2^{-1} + 1 + \sum_{i=2}^{\infty} 2^{-2^{i-1}} \\ &= 1 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-8} + \sum_{i=4}^{\infty} 2^{-2^i}. \end{aligned}$$

Since

$$\sum_{i=4}^{\infty} 2^{-2^i} < \int_3^{\infty} 2^{-2x}\,dx = \int_8^{\infty} 2^{-y}\frac{dy}{y} < 2^{-3}\int_8^{\infty} 2^{-y}\,dy = \frac{2^{-11}}{\ln 2},$$

$$D < 1 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-8} + \frac{2^{-11}}{\ln 2}$$

$$< 1.8187.$$

If double-rail inputs of A and B are available, then from the P- and I-module logic expressions, it is obvious that d is 2 AND or OR gate delays, whenever we use two-level logic. Therefore, the average delay through the P- and I-modules is 3.6374 gate delay. Including the delay of the OR gates, which generate the completion signal and the final result, the time duration of processing ranges from 4 to $2(\log n) + 4$ gate delay, with the average less than 5.6374 gate delay. If only single-rail inputs of A and B are available, then we have to add in one more inverter delay.

We have assumed that the input configurations have a uniform distribution. However, in different implementation environments, the probability distributions might be different. Thus, in certain cases, the average computing time might be larger than the value derived here. This is particularly true when the equality of A and B, the worst case for the computation delay, occurs with relatively large frequency.

The number of inputs to the OR gates S (and G) is $(\log n) + 1$. Let r be the gate fan-in. Then for $(\log n) + 1 \le r$, i.e., $n \le 2^{r-1}$, this will not cause any problem. For very large n, such that $(\log n) + 1 > r$, i.e., $n > 2^{r-1}$, the completion circuit could be implemented by a tree-type multilevel gate network. This would introduce an additional delay, approximately $\log_r(\log n)$. On the other hand, to implement this tree-type gate network, approximately $2\{(\log n) - 1\}/(r-1)$ OR gates are required.

ACKNOWLEDGMENT

The author is deeply indebted to Prof. S. H. Unger for his encouragement and advice during this work. His special thanks are to Prof. J. F. Traub for his valuable comments.

REFERENCES

[1] W. H. Molesworth, "Simpler carry look-ahead digital comparison," *Electron. Eng.*, vol. 50, no. 610, p. 20, Aug. 1978.

[2] C. A. Papachristou, "Parallel implementation of binary comparison circuits," *Int. J. Electron.*, vol. 47, no. 2, pp. 187–192, Aug. 1979.

[3] M. V. Subba Rao and S. C. Mittal, "SN 7485 gives carry look ahead digital comparison," *Electron. Eng.*, vol. 50, no. 603, p. 21, Mar. 1978.

[4] *The TTL Data Book for Design Engineers*, 2nd ed., Texas Instruments, Inc., Dallas, TX.

[5] S. H. Unger, "The generation of completion signals in iterative combinational circuits," *IEEE Trans. Comput.*, vol. C-26, Jan. 1977.

[6] ——, "Tree realization of iterative circuits," *IEEE Trans. Comput.*, vol. C-26, Apr. 1977.

[7] ——, *Asynchronous Sequential Switching Circuits*. New York: Wiley, 1969.

Discrete Basis and Computation of Elementary Functions

JEAN-MICHEL MULLER

Abstract **— We give necessary and sufficient conditions in order that the infinite product or sum of the terms of a positive decreasing sequence generates the reals in a given interval.**

Such a sequence will be called a *discrete basis*. We derive a class of algorithms without multiplications for the computation of elementary functions. We then obtain a unified theory for different existing algorithms, in particular Volder and Walther's CORDIC methods.

Index Terms **— CORDIC-like algorithms, hardware computation of elementary functions, representation of real numbers by infinite series.**

I. INTRODUCTION

The aim of this work is to generate and justify some hardware algorithms, using only additions and shifts, for computing the elementary functions.

Such algorithms have been previously introduced: Volder and Walther's CORDIC system [7], [9], [13], [15], [17], [18], and De Lugish's algorithms [5].

Schelin [13] proves the convergence of the CORDIC scheme by the mean of a decomposition theorem, which gives a sufficient condition to approximate the reals in an interval by a sum of predefined coefficients. Our attempt is to generalize this concept and to show its ability to generate algorithms.

To compute $f(t)$, the basic idea of the CORDIC scheme is to approximate t by a sum:

Manuscript received July 9, 1984; revised November 19, 1984.

The author is with Laboratoire TIM3, Institut IMAG, Université de Grenoble, BP 68, 38402 Saint-Martin D'Hères Cedex, France.

0018-9340/85/0900-0857$01.00

$$t = d_o e_o + d_1 e_1 + \cdots + d_n e_n, \qquad d_i = \pm 1$$

where the e_i are precomputed constants, chosen such that

$$f_{i+1} = f(d_o e_o + d_1 e_1 + \cdots + d_{i+1} e_{i+1})$$

can be computed "easily" from f_i (i.e., with only a few additions and "shifts," where a shift is a multiplication or a division by a power of the radix of the computer).

Our method, derived from the precedent, is to build a sequence (t_n) such that

$$\begin{cases} f(t_o) \text{ is known,} \\ t_i \to t \quad \text{as } i \to \infty, \\ f_{i+1} = f(t_{i+1}) \text{ can be computed "easily" from } f_i. \end{cases}$$

Thus, if f is continuous, then we have

$$f_i \to f(t) \quad \text{as } i \to \infty.$$

We present two classes of methods for generating the sequence (t_n).

1) Additive Methods:

$$t_{n+1} = t_n + d_n e_n$$

where

$$\begin{cases} -\text{the } e_i\text{'s are precomputed constants,} \\ -d_i = 0, 1, 2, \cdots, p \ (p\text{-unidirectional method}), \\ \quad \text{or } d_i = -p, -p+1, \cdots, p \ (p\text{-bidirectional method}). \end{cases}$$

2) Multiplicative Methods:

$$t_{n+1} = t_n \cdot e_n^{d_n}$$

where

$$\begin{cases} -d_i = 0, 1, \cdots, p \ (p\text{-unidirectional method}) \\ \quad \text{or } d_i = -p, \cdots, p \ (p\text{-bidirectional method}) \\ -\text{the } e_i\text{'s are such that a multiplication by } e_i \text{ (and by } e_i^{-1} \text{ with} \\ \text{the bidirectional method) can be performed easily} \\ \text{(for instance, with a binary computer, a multiply by} \\ e_i = (1 + 2^{-i}) \text{ reduces to a shift and an addition).} \end{cases}$$

(e_i) must be chosen such that every t included in a predefined interval can be generated by an appropriate choice of the sequence (d_i).

In the first part of this paper, we present some characterizations of the proper sequences (e_i) and some algorithms for generating (t_i).

In the last part, we study the way to generate algorithms for the computation of some functions, using the theory of Section II, and we introduce some examples.

II. The Discrete Basis Theory

A. Additive Methods

We want here to characterize the sequences (e_i) such that any t included in a given interval can be written $t = \sum_{n=0}^{\infty} d_n e_n$ where d_i is in $\{-p, \cdots, p\}$ or $\{0, \cdots, p\}$. For reasons of simplicity, we shall take decreasing positive sequences, such that $\sum_{n=0}^{\infty} e_n < +\infty$. The following definition is a consequence of these choices.

Definition: If $B = (e_n)$ is a decreasing infinite sequence of strictly positive reals such that

$$E = \sum_{n=0}^{\infty} e_n < +\infty,$$

then for any integer $p \geq 1$:

i) B is a *bidirectional discrete basis of order p* (p-BDB) for $I = [-pE, +pE]$ if for any $t \in I$ there exists (d_n), $d_n \in \{-p, -p+1, \cdots, 0, 1, \cdots, p\}$ such that

$$t = \sum_{n=0}^{\infty} d_n e_n .$$

ii) B is a *unidirectional discrete basis of order p* (p-UDB) for $J = [0, pE]$ if for any $t \in J$ there exists (d_n), $d_n \in \{0, 1, 2, \cdots, p\}$ such that

$$t = \sum_{n=0}^{\infty} d_n e_n .$$

The following result reveals that (3^{-n}) is a 1-BDB but not a 1-UDB, (2^{-n}) is both, and (10^{-n}) is neither.

Theorem 1: B is A) a p-BDB if and only if for any n, $e_n \leq 2p \sum_{k=n+1}^{\infty} e_k$, and B) a p-UDB if and only if for any n, $e_n \leq p \cdot \sum_{k=n+1}^{\infty} e_k$.

Proof: We give only the proof of A), for $p = 1$; that of B) is similar and easier than the previous one for $p = 1$, and for $p > 1$ the proof is immediate if we consider the fact that (e_n) is a p-BDB (p-UDB) if and only if the sequence (a_n) defined by

$$a_0 = a_1 = \cdots = a_{p-1} = e_0$$

$$a_n = e_{\lfloor n/p \rfloor}, \quad \text{where } \lfloor x \rfloor \text{ is the integer part of } x,$$

is a 1-BDB (1-UDB).

1) The Condition Is Necessary: Let us suppose that there exists an integer n such that

$$e_n > 2 \sum_{k=n+1}^{\infty} e_k . \tag{i}$$

Define

$$\begin{cases} S = \sum_{k=0}^{n-1} e_k , \\ \sigma = \sum_{k=n+1}^{\infty} e_k = E - S - e_n , \\ I =]S + \sigma, S + e_n - \sigma[. \end{cases}$$

Equation (i) implies that $I \neq \emptyset$.

Let us suppose $a \in I$; we have $a \leq E$. We shall show that for any sequence (d_m), $\sum_{m=0}^{\infty} d_m e_m \neq a$.

Let $b = \sum_{m=0}^{\infty} d_m e_m$

If $d_n = 0$ or $d_n = -1$ then $b \leq S + \sigma < a$.

If $d_n = 1$ then

If every $k < n$ verifies $d_k = 1$ then

$$b \geq S + e_n - \sigma > a .$$

If there exists $i < n$ such that $d_i \neq 1$

then $b \leq (S - e_i) + e_n + \sigma$

$\leq S + \sigma$ because $e_i \geq e_n$

(the sequence (e_k) is decreasing)

$< a$

Hence, in any case, $b \neq a$.

2) The Condition Is Sufficient: The proof is given by the convergence of the following algorithm.

Algorithm 1-BBDB (Bidirectional on 1-BDB):

Let $B = (e_n)$ be a sequence verifying the conditions of Theorem 1, and t be such that $|t| \leq E$. If the sequence (t_k) is defined by

1-BBDB:

$$\begin{cases} \sigma_o = E \\ t_o = 0 \\ \sigma_{n+1} = \sigma_n - e_n \\ t_{n+1} = t_n + d_n e_n \\ \qquad \text{where } d_n = \begin{cases} +1 \text{ if } t \geqslant t_n + \sigma_{n+1} \\ -1 \text{ if } t \leqslant t_n - \sigma_{n+1} \\ \text{else } 0 \end{cases} \end{cases}$$

then we have

$$t_n \to t \text{ as } n \to +\infty,$$

and more exactly

$$|t_n - t| \leqslant \sigma_n. \tag{1}$$

Proof: We have obviously $\sigma_n = \sum_{k=n}^{\infty} e_k$; thus, $\sigma_n \to 0$ as $n \to +\infty$.

We want to prove (1) by induction.

—(1) is true if $n = 0$

—if $n \geqslant 0$, let us suppose (1) is true for n

if $t \geqslant t_n$ then we have $d_n = 1$ if $t \geqslant t_n + \sigma_{n+1}$ else 0

if $d_n = 0$ then $t_{n+1} = t_n \leqslant t \leqslant t_n + \sigma_{n+1} = t_{n+1} + \sigma_{n+1}$

thus $|t_{n+1} - t| \leqslant \sigma_{n+1}$

if $d_n = 1$ then

—if $t \geqslant t_n + e_n$ then

$t_{n+1} \leqslant t \leqslant t_n + \sigma_n = t_{n+1} + \sigma_{n+1}$

therefore $|t_{n+1} - t| \leqslant \sigma_{n+1}$

—if $t < t_n + e_n$ then

$t_n + \sigma_{n+1} \leqslant t \leqslant t_n + e_n = t_{n+1}$ (ii)

but we have $e_n \leqslant 2\sigma_{n+1}$

therefore $t_n + e_n \leqslant t_n + 2\sigma_{n+1}$

i.e., $t_n + e_n - \sigma_{n+1} \leqslant t_n + \sigma_{n+1}$

i.e., $t_{n+1} - \sigma_{n+1} \leqslant t_n + \sigma_{n+1}$

i.e., with (ii): $t_{n+1} - \sigma_{n+1} \leqslant t \leqslant t_{n+1}$

therefore $|t_{n+1} - t| \leqslant \sigma_{n+1}$

if $t \leqslant t_n$ then the proof is similar since the relations are symmetrical with respect to t.

This ends the proof of the theorem.

Example: The geometric sequences.

If $e_n = a^{-n}$, $n = 0, 1, 2, \cdots$, for some $a > 1$, then $E = a/(a-1)$ and $B = (e_n)$ is a p-BDB if and only if $a \leqslant 2p + 1$, while B is a p-UDB if and only if $a \leqslant p + 1$. To see this we note that

$$\sum_{k=n+1}^{\infty} a^{-k} = a^{-n}/(a-1).$$

These geometric sequences were studied by Renyi [12] for p-UDB, and are called "β-expansions," and by Derrida, Gervois, and Pommeau [6] for 1-BDB ("λ-expansions"). It is worth noting that if a is an integer, we find the classical notion of numeration basis; thus, our theory appears to be a generalization of that concept.

We give two other algorithms: a unidirectional and a bidirectional, which converge on p-UDB. The proof of their convergence is similar and easier than that of 1-BBDB.

Algorithm p-UUDB (Unidirectional on p-UDB):

$$\begin{cases} ((e_n) \text{ is a } p\text{-UDB}, 0 \leqslant t \leqslant pE) \\ t_o = 0 \\ d_n = \max\{j \leqslant p, t_n + je_n \leqslant t\} \\ t_{n+1} = t_n + d_n e_n \end{cases}$$

Algorithm p-BUDB (Bidirectional on p-UDB):

$$((e_n) \text{ is a } p\text{-UDB}, t \leqslant pE)$$

$$\begin{cases} t_o = 0 \\ \text{if } t_n \leqslant t \text{ then} \\ \qquad d_n = \max\{1 \leqslant j \leqslant p, t_n + (j-1)e_n \leqslant t\} \\ \text{else } d_n = \min\{-p \leqslant j \leqslant -1, t_n + (j+1)e_n \geqslant t\} \\ t_{n+1} = t_n + d_n e_n \end{cases}$$

It is interesting to note that if $e_n = 2^{-n}$, then for any $t \in [0, 2]$, 1-UUDB and 1-BBDB give the same sequence (d_n) because the variable σ_n of the 1-BBDB algorithm is equal to e_{n-1}. It is not true for other 1-UDB's. For instance, if $e_n = 1.5^{-n}$ and $t = 2$, then 1-BBDB gives $d_o = 1$ and $d_1 = 0$ while 1-UUDB gives $d_o = d_1 = 1$.

These algorithms still converge if we replace large inequalities by strict inequalities when comparing t_n to t.

With these algorithms, we have for every n

$$|t_n - t| \leqslant p \sum_{k=n}^{\infty} e_k. \tag{2}$$

The convergence of p-BUDB is a generalization of Schelin's decomposition theorem [13].

Theorem 1 enables us to verify if a sequence (e_n) is a p-BDB or a p-UDB. Unfortunately, in most cases, A) and B) are difficult to prove directly. We give here a second theorem which simplifies this task.

Theorem 2: Let $B = (e_n)$ be a p-BDB (p-UDB), and let f be a function of a real variable, verifying

$$\begin{cases} f' \text{ is continuous on } I = [0, 2pE] (I = [0, pE]) \\ f(0) = 0 \\ f \text{ is concave and strictly increasing on } I. \end{cases}$$

We have: $f(B) = (f(e_n))$ is a p-BDB (p-UDB).

Proof:

a) *$f(e_n)$ is a decreasing sequence.*

It is obvious, since (e_n) is a decreasing sequence, and f is increasing on $[0, E]$.

b) *For any n, $f(e_n) > 0$.*

f is strictly increasing, thus, for any n, $f(e_n) > f(0) = 0$.

c) $\sum_{n=0}^{\infty} f(e_n) < +\infty$.

This result is a consequence of the fact that for every integer n, $f(e_n) \leqslant e_n f'(0)$. To see this we note that if f is concave with f' continuous on I, then f' is decreasing on I; thus,

$$f(e_n) = \int_0^{e_n} f'(x)\,dx \leqslant f'(0) \int_0^{e_n} dx = e_n f'(0).$$

Thus,

$$\sum_{n=0}^{\infty} f(e_n) \leqslant f'(0) \sum_{n=0}^{\infty} e_n < +\infty.$$

d) *$f(B)$ verifies A) (respectively, B)).*

Let k be p if B is a p-UDB and $2p$ if B is a p-BDB. The concavity of f implies that for every sequence $a_1, a_2, \cdots, a_n$ of reals included in I such that $k(a_1 + a_2 + \cdots + a_n)$ is in I we have

$$f(k(a_1 + a_2 + \cdots + a_n)) \leqslant k(f(a_1) + f(a_2) + \cdots + f(a_n)).$$

f is continuous; therefore, if (a_n) is an infinite sequence such that $k \sum_{n=0}^{\infty} a_n$ exists and is included in I, then

$$f\left(k\sum_{n=0}^{\infty}a_n\right) \leqslant k\sum_{n=0}^{\infty}f(a_n).$$

For every n, we have $e_n \leqslant k\sum_{p=n+1}^{\infty} e_p$. Thus,

$$f(e_n) \leqslant f\left(k\sum_{p=n+1}^{\infty}e_p\right) \leqslant k\sum_{p=n+1}^{\infty}f(e_p).$$

Thus, $f(B)$ is a p-BDB (p-UDB).

Example: $(\mathrm{Arctg}(a^{-k}))$ and $(\mathrm{Log}_b(1+a^{-k}))$ $(b>1)$ are p-UDB for $1 < a \leqslant p+1$ and p-BDB for $1 < a \leqslant 2p+1$.

B. Multiplicative Methods

Definition: If $B = (e_n)$ is a decreasing infinite sequence of reals, $e_n > 1$, such that

$$1 < \prod_{n=0}^{\infty} e_n = K < +\infty,$$

then the following hold.

i) B is a multiplicative bidirectional discrete basis of order p (p-MBDB) for $I = [K^{-p}, K^p]$ if for any $t \in I$ there exists (d_n), $d_n \in \{-p, -p+1, \cdots, 0, 1, 2, \cdots, p\}$, such that

$$t = \prod_{n=0}^{\infty} e_n^{d_n}.$$

ii) B is a multiplicative unidirectional discrete basis of order p (p-MUDB) for $I = [1, K^p]$ if for any $t \in I$ there exists (d_n), $d_n \in \{0, 1, \cdots, p\}$, such that

$$t = \prod_{n=0}^{\infty} e_n^{d_n}.$$

The following result proves that an exhaustive study of multiplicative discrete basis is not necessary because the properties and the algorithms of the additive discrete basis can be easily translated to the multiplicative discrete basis. It reveals that $(1+a^{-n})$ is a p-MUDB if $1 < a \leqslant p+1$, which will be very useful for building some algorithms (see Section III).

Theorem 3: (e_n) is a p-MBDB if and only if $(\ln(e_n))$ is a p-BDB, while (e_n) is a p-MUDB if and only $(\ln(e_n))$ is a p-UDB. (The proof is obvious.)

III. Applications

A. How to Build an Algorithm for the Computation of a Function

1) The Primal Method: Let us assume that we want to compute $f(t)$ for any t included in an interval I.

f is expected to be continuous and to verify

$$f(xTy) = g(f(x), f(y)) \quad (P)$$

where T represents addition or multiplication.

Examples:

Logarithm:
$$\begin{cases} f(x) = \ln(x) \\ T = * \\ g(u,v) = u + v. \end{cases}$$

Exponential:
$$\begin{cases} f(x) = \exp(x) \\ T = + \\ g(u,v) = uv. \end{cases}$$

Sine/Cosine:
$$\begin{cases} f(x) = \exp(ix) \\ T = + \\ g(u,v) = uv. \end{cases}$$

Power Functions:
$$\begin{cases} f(x) = x^a \\ T = * \\ g(u,v) = uv. \end{cases}$$

Let us suppose that there exists a discrete basis (e_n) of order p, additive if $T = +$, and multiplicative if $T = *$, such that for any u, $g(u, a_n)$ can be computed with additions and shifts where $a_n \in \{f(e_n), z\}$ ($z = 0$ if $T = +$, 1 if $T = *$) (in that case we will take unidirectional algorithms), or $a_n \in \{f(e_n), f(e_n^*), z\}$ where $e_n^* T e_n = z$ (in that case we will take bidirectional algorithms).

Thus, while we have written $t = \sum_{n=0}^{\infty} d_n e_n$ or $t = \prod_{n=0}^{\infty} e_n^{d_n}$, we obtain $f_i \to f(t)$ as $i \to +\infty$ where

—$f_0 = z$,

—$f_{i+1} = g(f_i, a_i)$, with $a_i = f(e_i T e_i T \cdots T e_i)$ (k times) if $d_i = k$; z if $d_i = 0$; $f(e_i^* T e_i^* T \cdots T e_i^*)$ (k times) if $d_i = -k$.

This method (computation of $f(t)$ using a sequence $t_i \to t$ such that $f(t_{i+1})$ can be obtained from $f(t_i)$) will be called the *primal algorithm*.

Before studying another method (the *dual algorithm*), we shall see some examples of the primal algorithm. Our purpose is not to present all the functions that can be computed in such a way, but to show how our building-algorithms method can be used.

a) Computation of the exponential function: Property (P) is verified by the exponential function:

$$\exp(x+y) = \exp(x) * \exp(y).$$

$T = +$, and thus we have to use an additive discrete basis (e_n) such that the product by $\exp(e_n)$ can be done with additions and shifts only. We have seen that $\ln(1+b^{-n})$ is a $(b-1)$-UDB where $b \geqslant 2$ is an integer; moreover, in a radix b computer, the product $u(1+b^{-n}) = u + u.b^{-n}$ can be computed using only an addition and a shift. Thus, we choose $e_n = \ln(1+b^{-n})$.

Since we cannot easily divide a number by e_n, we choose $(b-1)$-UUDB algorithm. We present the obtained algorithm for a binary computer ($b = 2$).

If N is the number of steps of the following algorithm (we approximate $t = \sum_{n=0}^{\infty} d_n e_n$ by $t_N = d_o e_o + \cdots + d_N e_N$), it can easily be shown, using (2) and the fact that for every $x > 0$, $\ln(1+x) < x$, that $|t_N - t| \leqslant 2^{-N}$.

Thus, the relative error on the result "exp" of the following algorithm, equal to $|\exp(t_N) - \exp(t)|/\exp(t)$, will be bounded by $\exp(2^{-N}) - 1 \sim 2^{-N}$.

Algorithm Exponential-UUDB [15]:

```
(Input: the argument t. Output: exp)
(e_n = ln(1 + 2^-n) are precomputed and stored constants)
Begin
    x := 0;
    exp := 1;
    k := 0;
    while k ⩽ N do
        begin
            while (x + e_k > t) and (k ⩽ N) do k := k + 1;
            exp := exp + exp · 2^-k;
            x := x + e_k;
            k := k + 1;                                   (*)
        end;
End.
```

Interval of convergence: $[0, E] = [0, 1.56\cdots]$.

b) CORDIC computation of sine and cosine functions: Volder and Walther's CORDIC algorithms [7], [9], [15], [17], [18] are a class of algorithms included in the 1-BUDB class. Here we study only one application of CORDIC: the computation of sine and cosine functions.

We suppose that we have a binary computer, and we want to

compute $f(t) = \exp(it)$, using the 1-BUDB algorithm. We have

$$f(t_{n+1}) = f(t_n + d_n e_n) = \exp(it_n) \cdot \exp(id_n e_n).$$

If we choose

$$\begin{cases} e_n = \text{Arctg}(2^{-n}) \text{ (we saw that it is a 1-UDB)} \\ K_n^N = \prod_{k=n}^{N-1} \cos e_k = \left(\prod_{k=n}^{N-1} (1 + 2^{-2k})\right)^{-1/2} \\ x_n = \cos(t_n) * K_n^N \\ y_n = \sin(t_n) * K_n^N \end{cases}$$

then it can easily be shown that, for $t \in [-E, E], E = \sum_{n=0}^{\infty} \text{Arctan}(2^{-n}) = 1.74\cdots$, the following algorithm gives us

$$\begin{cases} x_N = \cos t \pm 2^{-N} \\ y_N = \sin t \pm 2^{-N}. \end{cases}$$

CORDIC:

```
Begin
    x_o := K_0^N; y_o := 0; z_o := t;
    for k := 0 to N - 1 do
        begin
            if z_k > 0 then d_k := 1 else d_k := -1;
            x_{k+1} = x_k - d_k y_k 2^{-k};
            y_{k+1} = y_k + d_k x_k 2^{-k};
            z_{k+1} := z_k - d_k e_k
        end
End.
```

One can modify this algorithm for use in a radix-p computer with the algorithm $(p-1)$-BUDB applied to the $(p-1)$-UDB $(\text{Arctan}(p^{-n}))$.

c) Square rooting: We saw that $(1 + p^{-n})$ is a $(p-1)$-MUDB; thus, it is easy to show that $((1 + p^{-n})^2)$ is a $(p-1)$-MUDB.

The basic idea of our algorithm is to write any t included in $[1, (\prod_{n=0}^{\infty}(1 + p^{-n}))^{2(p-1)}]$ in the form

$$t = \prod_{n=0}^{\infty} ((1 + p^{-n})^2)^{d_n}$$

and to obtain

$$\sqrt{t} = \prod_{n=0}^{\infty} (1 + p^{-n})^{d_n}.$$

We give this algorithm for $p = 10$, using a multiplicative version of the unidirectional algorithm.

SQRT:

(correct for $t \in [1, K^9], K = 4.948\cdots, K^9 = 1778586.2\cdots$)

```
Begin
    x := 1; sq := 1;
    for k := 0 to N do
        begin
            d := 0; u := 0;
            while (u ≤ t) and (d < 9) do
                begin
                    u := x + 10^{-2k} x + 10^{-k} x + 10^{-k} x;
                    if u ≤ t then
                        begin
                            x := u;
                            sq := sq + sq 10^{-k};
                        end;
                    d := d + 1
                end
        end
End.
```

This algorithm gives $\text{sq} = \sqrt{t}$ within an error given by $(\prod_{k=N+1}^{\infty}(1 + 10^{-k}))^{-9} \leq \text{sq}/\sqrt{t} \leq 1$; thus, the relative error is bounded by $1 - 1/(\prod_{k=N+1}^{\infty}(1 + 10^{-k}))^9 \sim 10^{-N}$. Such an algorithm can be built to compute $\sqrt[k]{t}$, using the $(p-1)$-MUDB $((1 + p^{-n})^k)$.

One can easily find algorithms for computing monomials (and therefore polynomials) using the same unidirectional multiplicative algorithm associated with the $(p-1)$-discrete basis $(1 + p^{-n})$ on a radix-p computer.

2) The Dual Method: Now, we shall study a way to compute f^{-1} when f is a monotonic function computable by the mean of the primal method.

Let us suppose that f is strictly monotonic on the interval $I = [0, pE]$ (or $[-pE, +pE]$, or $[1, K^p]$, or $[K^{-p}, K^p]$).

If we want to compute $t = f^{-1}(u)$, the monotonicity of f implies that comparing t' to t is equivalent to comparing $f(t')$ to u. Thus, if f is increasing, and if we replace, for instance, the line

$$d_n = \max\{j \leq p, t_n + je_n \leq t\}$$

of p-UUDB algorithm with the line

$$d_n = \max\{j \leq p, f(t_n + je_n) \leq u\},$$

then we obtain the same result: $t_n \to f^{-1}(u)$ as $n \to \infty$.

It is obvious that we can translate this modification to the other additive or multiplicative algorithms. There are two interesting consequences of that, as follow.

1) A dual algorithm differs from this primal version only on the choice of d_n; thus, for instance, one can transform the exponential-UUDB algorithm into an algorithm which computes the logarithm on the interval $[\exp(0), \exp(E)], \exp(E) = 4.7\cdots$, by replacing the line

"while $(x + e_k > t)$ and $(k \leq N)$ do $k := k + 1$;"

by

"while $((\exp + \exp \cdot 2^{-k}) > u)$ and $(k \leq N)$ do $k := k + 1$;"

The desired result will be $x = \ln(u) \pm ae, ae \leq 2^{-N}$.

It is interesting to note that this algorithm can be interpreted as a primal multiplicative algorithm using the 1-MUDB $(1 + 2^{-n})$.

2) The error at step N can easily be found using (2); if we take an additive algorithm, the absolute error will be $p \sum_{k=N+1}^{\infty} e_k$, and with a multiplicative algorithm, if we note $K_N = \prod_{i=N+1}^{\infty} e_i$, then the relative error will be bounded by $\max\{1 - 1/K_N, K_N - 1\} = K_N - 1$.

Examples:

1) The previous algorithm, which computes the natural logarithm [or any base b logarithm if we replace $\ln(1 + p^{-n})$ by $\log_b(1 + p^{-n})$], is also true for the exponential function.

2) The CORDIC "vectoring mode" [13], [17], [18] is the dual mode of the "rotation mode." For instance, the dual mode of the sine/cosine algorithm presented here enables us to compute the arctangent function.

3) Extension of the Domain: One can extend the domain of convergence of our algorithms by allowing the coefficients d_i to be greater than p (practically, it is equivalent to repeat several times the same e_i in the sequence). For instance, if we delete the line (*) of the exponential-UUDB, we obtain an algorithm which converges on $[0, +\infty]$ instead of $[0, E]$, but if we want a relative precision 2^{-N}, the time of computation is not necessarily proportional to N.

IV. Conclusion

The methods studied here enable us to find many of the hardware algorithms. These algorithms must be tested, but relation (2) is very useful for predicting their accuracy and speed. The principal interest of this study is the proof that some well-known methods, which seemed independent, are in fact based upon the same principles.

ACKNOWLEDGMENT

The author thanks the referees, whose helpful and detailed critiques were responsible for significant improvements in the paper, and Dr. M. Cosnard for his encouragement.

REFERENCES

[1] P. W. Baker, "More efficient radix-2 algorithms for some elementary functions," *IEEE Trans. Comput.*, vol. C-24, pp. 1049–1054, Nov. 1975.
[2] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios and square roots," *IBM J. Res. Develop.*, vol. 16, pp. 380–388, July 1972.
[3] W. Cody and W. Waite, *Software Manual for the Elementary Functions*. Englewood Cliffs, NJ: Prentice-Hall, 1980.
[4] J. M. Delosme, "VLSI implementation of rotations in pseudo-Euclidian spaces," in *Proc. 1983 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Boston, MA, Apr. 1983, pp. 927–930.
[5] B. De Lugish, "A class of algorithms for automatic evaluation of certain elementary functions in a binary computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. Illinois, Urbana, IL, June 1970.
[6] B. Derrida, A. Gervois, and Y. Pomeau, "Iteration of endomorphisms on the real axis and representation of numbers," Commissariat Énerg. Atomique, Serv. Phys. Théor., CEN Saclay.
[7] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-23, Oct. 1974.
[8] M. D. Ercegovac, "Radix-16 evaluation of certain elementary functions," *IEEE Trans. Comput.*, vol. C-22, June 1973.
[9] G. H. Haviland and A. A. Tuszynsky, "A CORDIC arithmetic processor chip," *IEEE Trans. Comput.*, vol. C-29, Feb. 1980.
[10] J. Kropa, "Calculator algorithms," *Math. Mag.*, vol. 51, no. 2, pp. 106–109, Mar. 1978.
[11] J. M. Muller, "Le logiciel FONCTELEM, Présentation et tests," Inst. IMAG, Grenoble, France, Rapport Recherches, to appear.
[12] A. Renyi, "Representations for real numbers and their ergodic functions," *Acta Math. Acad. Sci.*, Hungary, pp. 477–493, 1957.
[13] C. W. Schelin, "Calculator function approximation," *Amer. Math. Monthly*, vol. 90, no. 5, May 1983.
[14] H. Schmid and A. Bogocki, "Use decimal CORDIC for generation of many transcendental functions," *Elec. Des. Mag.*, pp. 64–73, Feb. 1973.
[15] O. Spaniol, *Computer Arithmetic and Design*. New York: Wiley, 1981.
[16] W. H. Specker, "A class of algorithms for $\ln(x)$, $\exp(x)$, $\sin(x)$, $\cos(x)$, $\arctan(x)$ and $\operatorname{arcot}(x)$," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 85–86, 1965.
[17] J. Volder, "The CORDIC computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.
[18] J. Walther, "A unified algorithm for elementary functions," in *Joint Comput. Conf. Proc.*, vol. 38, pp. 379–385.

Implementation of a Constrained Regularization Program (CONTIN) on a Desktop Computer

KEITH J. STELZER AND MICHAEL A. GORDON

Abstract **— A constrained regularization program for inverting linear algebraic and integral equations (CONTIN) [1]–[3] has been implemented on an MC68000 based desktop computer supplemented with floating point accelerator hardware. Implementation of CONTIN on a desktop computer system enhances interactive capabilities and allows economical and time-efficient use of this program on a regular basis within the laboratory.**

Index Terms **— Constrained regularization, molecular weight determination, particle size distribution, photon correlation spectroscopy, quasi-elastic light scattering, Stokes radius determination.**

Manuscript received January 22, 1985; revised March 1, 1985. This work was supported in part by NIH under Grant 7 RO1 NS10977-01, in part by USPHS under Training Grant ES 07079, and by a Procter and Gamble Fellowship.

The authors are with the Department of Pharmacology, Toxicology and Therapeutics, University of Kansas College of Health Sciences and Hospital, Kansas City, KS 66103.

INTRODUCTION

CONTIN is a constrained regularization program which has been developed and made available by Provencher [1]–[3]. This program is applicable to the analysis of several types of experimental data encountered in the natural sciences. CONTIN may be used to solve linear algebraic equations containing unknown noise components which may be representative of data obtained with experiments such as X-ray and neutron scattering, photon correlation spectroscopy, circular dichroism, and others [1]. The program has proven particularly useful for solving integral equations for effectively continuous distributions [2] of diffusion coefficients or molecular weights by photon correlation spectroscopy [4], [5].

IMPLEMENTATIONS

The particular application of the constrained regularization program is defined by numerous user variables and subprograms [1]–[3]. Typically, CONTIN is implemented in Fortran IV, with a minimum of ANSI extensions and exceptions [2], on time-shared mainframe systems, such as the IBM 370, or minicomputers, such as the VAX-11/780 [2]. Typical execution times on these systems have been reported to range from 180 to 360 s [2], [3]. The extent of user interaction required for most efficient use of CONTIN leads to certain difficulties with mainframe or minicomputer implementations. Since CONTIN is computationally intensive, one apparent difficulty rests in the cost of frequent use of the program on time-shared mainframe systems. Another difficulty may be the lack of availability of suitable large computer systems for actual implementation. However, the most significant difficulty may be that, fundamentally, CONTIN is not most appropriately implemented on large time-shared or undedicated systems.

The most effective use of CONTIN involves continuous evaluation of the process of convergence to an acceptable constrained model of the system under investigation. If user-defined parameters are, for example, noted to be inappropriate for a given data set, the most time-effective remedy may involve program termination with subsequent parameter correction and execution. This observation is reinforced by the fact that CONTIN provides a continuous set of graphical output for each potential solution. It is our view that the solution to these concerns rests with program implementation on a suitable low-cost desktop computer.

We have determined that a suitable compromise between speed of computation and cost can be met by implementation of CONTIN in Fortran 77 on an MC68000 (Motorola) based system with floating point hardware capabilities. The particular solution which we have implemented uses the Hewlett–Packard 9000 series 200 computer, model 220 (MC68000 8 MHz). A suitable Fortran 77 compiler for this computer is available from International Electronic Machinery, Inc. (Fort Collins, CO). This compiler compiles Fortran 77 source text into native 68000 object code. However, as such, there is insufficient computational speed for the necessary double precision floating point operations. The execution of resulting object code files is implemented under the Pascal 2.0 operating system with floating point acceleration provided by the use of an FP-200 floating point coprocessor provided by Infotek Systems (Anaheim, CA). This floating point coprocessor results in up to a 600 percent increase in floating point computational speed.

CONTIN was implemented on the desktop computer system described above under its default configuration for application to

0018-9340/85/0900-0862$01.00