

CORDIC およびその変形に もとづく初等函数計算

京大教理研 一松 信

0. 概論

昨年この研究集会で, CORDIC の紹介をした[5], その双曲型の場合から, 平方根, 指数函数, 対数函数の新しい計算法を考えたので, 報告するが, しかしこれらはむしろ加法定理の応用[3], あるいはある水準線のトレース[8]と考えたほうが見透しがよい. ここには主として理論的な原理をのべる. 能率の比較は, 次の小柳氏の講演にゆずる. この種の計算法は, ファームウェアの一環であって, 超小型機または超大型機に好適であることが, 多くの人によって指摘されている.

1. Chen の算法 [8]

(これは研究集会の前日 滝谷政昭氏から[8]を教わり, 予稿にはなかったものである).

$f(x_0) = y_0$ を計算するために, 補助変数 y を導入し, 適

当る函数 $F(x, y)$ を作り,

$$F(x_0, y_0) = z_0, \quad F(x_\omega, z_0) = z_0$$

であるようにする. — じつさいには F は y について線型

$$F(x, y) = g(x)y + h(x), \quad g(x_\omega) = 1, \quad h(x_\omega) = 0$$

とする. (x_0, y_0) から始めて, 変換

$$x_{k+1} = \varphi(x_k, y_k), \quad y_{k+1} = \psi(x_k, y_k)$$

を反復し, つねに $F(x_k, y_k) = z_0$ であるようにする. この

点列 (x_k, y_k) を $F(x_\omega, y_\omega) = y_\omega$ をみたす (x_ω, y_ω) — と

くは上の線型の場合には $g(x_\omega) = 1, h(x_\omega) = 0$ をみたす x_ω —

に近づけると, $z_0 = f(x_0) = F(x_0, y_0) = F(x_k, y_k) = F(x_\omega, y_\omega)$

$= y_\omega$ となって, y_ω が $f(x_0)$ の値 z_0 である.

以上はあまりに abstract であるか, たとえば指数函数 e^x に対しては, $F(x, y) = ye^x$, $x_\omega = 0$ とし, 変換を

$$x_{k+1} = x_k - \alpha_k, \quad y_{k+1} = y_k \times e^{\alpha_k}$$

とすればよい. このとき α_k を簡単な数とせず, e^{α_k} を簡単

な (たとえば 1 ± 2^{-k}) 数になるように α_k を選ぶ. 対数出

数 $\log x$ に対しては, $F(x, y) = y + \log x$, $x_\omega = 1$, 変換は

$$x_{k+1} = \beta_k \times x_k, \quad y_{k+1} = y_k - \log \beta_k$$

とし, β_k を簡単な数 (たとえば 1 ± 2^{-k}) とする. これは

つぎにのべる STL と本質的に同じになる. (ただし一つ所,

Chen 独得の名著想がある. 次節参照)

2. STL 法 [3]

上記の Chen の算法を具体化すると、加法定理に基づき Specker の STL (Successive Table Look-up) 法 [3] と同じになる。 $a_k = \log(1+2^{-k})$ ($k=1, 2, \dots$) をあらかじめ計算しておく。使用計算機は 2 進法のものとする。

対数函数 $1/2 \leq x_0 \leq 1$ とする。 $x := x_0$; $y := 0$ から始めて、つぎの算法を反復する。

(1) $w := (1+2^{-k}) \times x$; if $w < 1$ then begin
 $x := w$; $y := y - a_k$ end;

x が十分に近づくまで反復するが、 $x = 1-t \neq 0$ ならば $\log(1-t) = -t - t^2/2 - \dots$ で、 t^2 以降が無視できれば、 $\log x \approx x-1$ としてよい。ゆえに N ビット必要るときは、 $k = N/2$ まで反復して、あと $x-1$ を y に加えればよい。以上の計算は、シフトと加減算のみで可能である。マイクロプログラムによる $\log$ の計算は、この方式がもっとも速くて確実のようである。

指数函数 $0 \leq x_0 \leq \log 2 = 0.693\dots$ とする。 $x := x_0$; $y := 1$ から始めて、つぎの算法を反復する。

(2) $w := x - a_k$; if $w \geq 0$ then begin
 $x := w$; $y := (1+2^{-k}) \times y$ end;

N ビット必要ならば、 N 回反復すればよい。ただし半語長程

度の乗法をいともやれば、 $N/2$ 回をやめ、 $x \div 1$ のとき $e^x \div 1+x$ を利用して、 $y = 1+x$ を掛ける — (じつさいには x をかけて xy を求め、それを y に加える、こうすれば、少しくらいの誤差はあとで補正できる。Chen は、 n が大きければ

$$a_n = \log(1+2^{-n}) = 2^{-n} - 2^{-2n-1} + 2^{-3n}/3 - \dots \quad (\div 2^{-n})$$

であることを利用し、 $w \geq 0$ の判定を減法をせず、 x の n ビット目が 1 か否かで判定している。ビット単位の判定が速くできる計算機では、これは有用な能率化であろう。

その音指数関数の近似において、区間を細分し、分点は簡単な有限小数 α の $\log$ とした式があった。(こうすると α 倍の乗法で丸め誤差が少なくなる)。上記の式はこのような区間の細分を極度におしすすめ、最後は 1 次式で近似した式とも解釈できる。また乗除算の計算で、 2^{-n} を $a_n = \log(1+2^{-n})$ におきかえた擬似乗除算ともみられる。

Meggitt [2] は 10 進法でこのような擬似乗除算を論じているが、これを 2 進法に直すのは、かえって簡単である。

3. Non-restoring 式変形 (双曲型 CORDIC)

普通の 2 進法は、 $0.a_1a_2\dots = \sum 2^{-i}a_i$ とし、乗算では 1 を 3 加え、0 を 3 そのままとする。これに対して、1

は+1, 0は-1を表わすとし, 1を3加え, 0を3引くという non-restarting 方式がある. これはとくに2の補数表示で正負交通に扱うとき有利である.

前節の算法は, 1を3加え, 0を3何もしない方式であるが, これを1を3加え, 0を3引く, という形に変形することが出来る. たとえば(2)を変形して, $x := x_1$, $y := y_1$ から始め, つぎの算法の反復とする:

(3) if $x \geq 0$ then begin $x := x - b_k$; $y := y \times (1 + 2^{-k})$ end
else begin $x := x + b_k$; $y := y \times (1 - 2^{-k})$ end;

ただしこのとき1は,

$$b_k = \operatorname{arctanh} 2^{-k} = \frac{1}{2} \log \frac{1+2^{-k}}{1-2^{-k}}$$

である. $|x_1| \leq \log 2$ のとき, $x_\infty = 0$ なる1は,

$$y_\infty = y_1 \cdot \tilde{K} \cdot e^{x_1}, \quad \tilde{K} = \prod_{k=1}^{\infty} (1 - 2^{-2k})^{1/2} \doteq 0.84\cdots$$

$y_1 = 1/\tilde{K}$ ととれば, e^{x_1} 自体さうる.

これはCORDICの双曲型の算法と本質的に同一である.

ただし最後に $x \neq 0$ になつたとき, $e^x \doteq 1+x$ として, これをかける補正をしないと, 反復計算だけで1は収束しない隙間を生ずる. それを防ぐもっとも簡単な手段は, k が

$$k_0 = 1, \quad k_{i+1} = 3k_i + 1, \quad \text{すなわち } 4, 13, 40, \dots$$

1になって, 同じ定数でもう1回反復することである (厄年留年法 と仮稱しよう).

$\log$ については, (3)の逆として, $x := x_1, y := 0$ から始め, つぎの算法を反復する:

(4) if $x \leq t$ then begin $x := (1 + 2^{-k})x$; $y := y - b_k$ end
else $x := (1 - 2^{-k})x$; $y := y + b_k$ end;

$$b_k = \operatorname{arctanh} 2^{-k}, \quad y \rightarrow \log t.$$

ただし $1 - 2^{-1} = 1/2$ が小さすぎるので, 反復は $k=2$ から始める. x_1 は $\sqrt{3}/2\sqrt{2} = 1.045723138\dots$ とする. 収束域が $0.57\dots \leq t \leq 1.76\dots$ と狭いため, $3/4 \leq t \leq 3/2$ または $1/\sqrt{2} \leq t \leq \sqrt{2}$ と標準化しなければならない ([7]参照).

Non-restoring 方式の長所は, 1 に対して 2 対称であることと, $b_k = 2^{-k} + 2^{-3k}/3 + \dots$ であって, Nビット必要なら $N/3$ から先は $b_k = 2^{-k}$ としてよいため, 定数数が少なくて済む点であるが, 反面 1 でも 0 でも必ず掛ける手間がかかる上に, $\log$ の 1 に近い引数のとき, 1 ± 2^{-k} をかけて反復するため精度がおちたりして, 本来の STL 法にあるようである.

4. 複素数による $\cos x, \sin x$ の計算

指数関数の式に複素数代入すると

$$\exp(iz) = \left[\prod_{k=0}^n (1 + a_k z)^{-1/2} (1 + ia_k) \right] \cdot \exp(iz - \sum_{k=0}^n \operatorname{arctan} a_k)$$

となるから, $z_0 := t$ ($|t| \leq \pi/2$) から始めて

$$(5) \text{ if } z \geq 0 \text{ then } a := 2^{-k} \quad \text{else } a := -2^{-k};$$

$$z := z - \text{actan } a \quad (\pm \text{actan } 2^{-k} \in \frac{\pi}{4})$$

とする反復により, 必要なビット数まで反復すると

$$\cos t = \text{Re} \left[K^{-1} \prod_{k=0}^n (1 + i a_k) \right], \quad \sin t = \text{Im} \left[K^{-1} \prod_{k=0}^n (1 + i a_k) \right]$$

$$K = \prod_{k=0}^n (1 + 2^{-2k})^{1/2} = 1.72 \dots$$

となる. $\exp(iz) = x + iy$ とすれば, 上記の変換は

$$x_{k+1} = x_k - a_k y_k, \quad y_{k+1} = y_k + a_k x_k \quad (a_k = \pm 2^{-k})$$

を同時に並行して適用したものにほかわらず, これは CORDIC そのものである ([9]).

これを逆に適用すれば, actan がえられる. ただし actan については, $\log$ を複素変数とした形で, $a_k = 2^{-k}$ または 0 とした *restoring* 方式の反復も可能である. じつせい [3] にのべられている算法は, その形である. なお $\arcsin$, $\arccos$ も同じような方式で, 直接に計算することができる ([9] 参照).

上記の x, y の変換の最初の式だけを $x_{k+1} = x_k + a_k y_k$ にかえたのが, 双曲型の CORDIC であり, ここで $x_k = y_k$ として, $x_{k+1} = x_k (1 \pm 2^{-k})$ とすると, §3 の γ は non-restoring 方式の指数函数 (および対数函数) の計算である. 双曲座標では $|y| < x$ のはずだが, $x = y$ としても

正しいことは, [7] に証明したとおりである。

5. 平方根の計算

平方根の計算法は, Newton 反復がもっともよいようで, すでに研究されつくしたようであるが, Chen の算法で,
 $F = y x^{-1/2}$, $x_0 = 1$ とする方法が可能である。——なお
 $F = y/x$, $x_0 = 1$ とすれば除法になる。ここでは 3. の双曲型CORDIC によって, 偏向成分を 0 に近づけると, x 成分に $\tilde{K}\sqrt{x_1^2 - y_1^2}$ がえられることを利用した方法をのべる ([7])
 $x_1 := t + c$; $y_1 := t - c$ から始めれば,

$$x_n = 2\tilde{K}\sqrt{c}\sqrt{t}$$

となるから, $2\tilde{K}\sqrt{c} = 1$ であるように c をとっておけば, 直接に $\sqrt{t}$ をうる。このとき CORDIC の z 成分には

$$\operatorname{arctanh} \frac{y_1}{x_1} = \frac{1}{2} \log \frac{x_1 + y_1}{x_1 - y_1} = \frac{1}{2} \log \frac{t}{c}$$

がえられるけれども, 平方根と対数とが同時に必要なのはほとんどないので, 平方根の計算用には, z 成分および定数 b_n を捨てて, 次のようにすればよい。 $x := t + c$, $y := t - c$ から始め, つぎの計算をくりかえす:

(6) if $y \geq 0$ then $x := x - 2^{-k}y$; $y := y - 2^{-k}x$
else $x := x + 2^{-k}y$; $y := y + 2^{-k}x$;

ただしここで $\boxed{\quad}$ 内は同時に平行して実行する。すなわち
あとの式の x は、前の式でおきかえられた x でなく、おき
かえられる前の x である。逐次処理しかできないければ、たとえば

begin $w := x - 2^{-k}y$; $y := y - 2^{-k}x$; $x := w$ end

というように解釈しなければいけない。また収束を保証する
ために、 $k = 4, 13 (40, 121)$ で同じ k で 2 度計算する。

$$c = 1/4 K^2 = 0.36451229219 \dots$$

ただしこれは無限乗積の値である、TOSBAC-3400 (仮数部
37ビット) では、 $1/4 \leq c \leq 1$ において、なるべく誤差が全
体的に最小になるように半実験的に定めた所

$$c = 0.36451229226$$

とするのが最適であった、(これは 二進十進変換の誤差の影響
らしい)。

このとき

$$\begin{aligned} (x_n^2 - y_n^2)^{1/2} &= x_n \left(1 - \frac{y_n^2}{2x_n^2} - \dots \right) \\ &= K (x_1^2 - y_1^2)^{1/2} \end{aligned}$$

であるから、 $y_n^2 / 2x_n^2$ が無限でいくまで、したがって N ビ
ット必要なら $N/2$ 回 反復をやめてよい。じつはい TOSB
AC-3400 の実験でも、 $k=38$ までは反復するより、 $k=19$
で止めたほうが、かえって 誤差が少なかった。

以上はまったくシフトと加減算のみで実行する方法である

が、除算をいとおなれば、(6)を何段階かやってから、その近似値 x_k を出発値として、Newton 法にきりかえてもよい。この場合には、初期値を $\sqrt{t}$ の折れ線近似(必ずしも最良近似ではない)で作らだしたものとも考えられる。

この式式の収束域は、 $e^{-2B} \leq t/c \leq e^{2B}$,

$$(B = \sum_{k=0}^n \operatorname{arctanh} 2^{-k} \doteq 1.117, \quad e^{2B} \doteq 9.34 \dots, \quad c \doteq 0.36 \dots)$$

で、 $\{1/4 \leq t \leq 1\}$ も $\{1/16 \leq t \leq 1\}$ も十分に含まれるが、 $t=1$ に近いとき、はじめの a の $x_k > 1$ で"あふれ"を生ずるので、 $\{1/8 \leq t \leq 1/2\}$ に標準化するほうがよい。このように狭域でよければ、 $k=2$ ($a_2=1/4$) から始め、 c を $k=2$ からの積、(たがって前記の位の $3/4$ 倍 ($c=0.27338421920$) とし、 $k=7 (=2 \times 3 + 1)$ において同じ k で反復するようを修正も可能である(反復が2回入り、精度もよくなる.)。

反対に収束域を広くしたければ、 $a_0=3/4$ から始め、 $a_k=2^{-k}$ のうち、 a_4 と a_5 の中間に $a_{4.5}=a_4 \times (3/4)$ 、 a_{13} と a_{14} の間に $a_{13.5}=a_{13} \times (3/4)$ を補う(定数 b_k も補い c も変更する)手がある。7プログラムは応介にあるが、 $c \doteq 0.81$ 、収束域はほぼ $0.014 \leq t \leq 50$ となり、 $\{1/64 \leq t \leq 16\}$ を含むので、16進のまま始めるのに有用である。このときには、 a_4 くらいまでやめて、あとを Newton 法にきりかえるほうが有利と思われる。

参考文献

- [1] J.E.Volder, Binary computation algorithms for coordinate rotation and function generation, Convair Report IAR-1 148, 1956.
- [2] J.E.Meggitt, Pseudo division and pseudo multiplication processes, IBM Research & Devel., 6 (1962), 210-226.
- [3] W.H.Specker, A class of algorithms for $\text{Ln } x$, $\text{Exp } x$, $\text{Sin } x$, $\text{Cos } x$, $\text{Tan}^{-1}x$, $\text{Cot}^{-1}x$, IEEE Trans. on Elec. Comp. 14 (1965), 85-86.
- [4] J.S.Walther, A unified algorithm for elementary functions, Spring Joint Comp. Conference 1971, 379-385.
- [5] 一松信, 初等函数の統一の計算法 —CORDICによる—
数理解析研講究録 190 (1973), 156-165.
- [6] 萩原博・渡辺勝正・小柳滋, マイクロプロセッサによる
初等関数近似, 第14回情報処理学会年会予稿集 (1973),
171-172.
- [7] S.Hitotumatu, A new method for the computation of square root, exponential and logarithmic functions through hyperbolic CORDIC, will appear in Rev. Inst. Comp. Cluj.
- [8] Tien Chi Chen, The automatic computation of exponential, logarithms, ratios and square roots, IBM Research RJ 970(16884), 1972, p.32
- [9] M.D.Perle, Cordic Technique reduces trigonometric function look-up, Computer Design, 1971 June, p.72-78)

(但し p.73, 75, 77は広告).