数理解析研究所講究録
第746巻 1991年 211-216

平方根を計算する高次収束法とその計算量の比較

— $O(n^2)$の高速除算法による計算に関して —

小沢　一文（仙台電波高専）

1.はじめに

著者は、平方根に収束する高次収束法の解法群を提案し、その多倍長演算における時間計算量を考察し、数値実験との比較検討を行なってきた［1］。その結果、演算桁数が固定されている演算（ここでは「固定長演算」と呼ぶことにする）では、計算法を工夫することによって、高次の解法を2次収束法よりも高速化することが可能になった。一方、演算桁数を必要に応じて増減できる演算（ここでは「可変長演算」とよぶことにする）を用いた場合、除算の乗算に対する時間比がかなり大きい場合 — 例えば2～3程度 — には、平方根の逆数に収束する「逆平方根法」が除算を含まないため、最も高速であることが判明した。

著者はその後、除算と乗算の時間比がほぼ1となる除算法［2］を提案したので、その除算法を用いることを前提とし、「可変長演算」における高次収束法の解法群の時間計算量を考察し、「逆平方根法」との比較を行なう。

2.高次収束法とその時間計算量

平方根$\sqrt{a}$に収束する$r(>1)$次収束法の反復式は以下の通りである［1］：

$$x_{k+1} = x_k + \Delta_r(x_k), \quad k=0,1,\cdots, \tag{2.1}$$

$$\Delta_r(x_k) = \frac{A_r(x_k)}{B_r(x_k)},$$

$$A_r(x) = \sum_i \left[\binom{r}{2i} - \binom{r}{2i+1} \right] a^i x^{r-2i}, \tag{2.2.1}$$

$$B_r(x) = \sum_i \binom{r}{2i+1} a^i x^{r-2i-1} \tag{2.2.2}$$

ここでは、t_0桁近似の初期値x_0を与え、r次収束法 (2.1)をp回反復し、 $s(=r^p t_0)$桁近似を得るのに要する時間計算量$T_r(t_0;s)$を導出し、何次収束法が最も高速かを考察する。

反復法(2.1)を用いて$\sqrt{a}$を計算するとき、仮にx_kが$\sqrt{a}$のt_k桁正しい近似であるとすると、増分$\Delta_r(x_k)$はその上位 $(r-1)t_k$桁だけが正しく計算されていれば、x_{k+1}はr $t_k(=t_{k+1})$桁正しい近似となる。すなわち、数値的にはr次収束性が保証される。そのため、式 (2.1)を常に一定の桁数で計算するのではなく、増分Δ_rを上位$(r-1)t_k$桁だけ計算し、その桁数を反復一回ごとにr倍し

ていく、という方針によって計算する方が計算量が少なくなるので有利である。ここでは、乗除算は$O(n^2)$の時間計算量を持つアルゴリズムを用いることにする。具体的には、n桁の数の乗除算に要する時間をn^2、n桁の数を二乗するのに要する時間を$\frac{1}{2}n^2$とする。

増分$\Delta_r(x_k)$は、次数rが偶数のとき次式のような形をしている：

$$\Delta_r(x_k) = \frac{r(a-Y_k)*A'_r(Y_k)}{(r-1)x_k*B'_r(Y_k)}, \tag{2.3}$$

$$Y_k = x_k^2$$

ここで、$A'_r(Y)$, $B'_r(Y)$はそれぞれ最高次の係数が1となる $r/2-1$次の多項式である。第$k+1$ ステップで増分$\Delta_r(x_k)$を上位$(r-1)t_k$だけ正確に計算するためには、まず$Y_k = x_k^2$を $2t_k$桁計算し、次に、残りの演算の全てを$(r-1)t_k$ で行なう必要がある。多項式の計算は因数分解などの前処理を行なわないで済むHornerの方法を用いることにする。このとき、第$k+1$ステップ全体では$(r/2-1)$次多項式を2回計算し、その他、多倍長乗算（多倍長数×多倍長数）を2回、多倍長除算（多倍長数/多倍長数）を1回、多倍長数×単精度数を2回（rは単精度数と仮定する）行なうので、このステップでの全計算量は$((r-1)^3+2)t_k^2 + O(t_k)$となる。したがって、 t_0桁近似を初期値として $s(s>>t)$桁近似を得るのに要する時間計算量は、 $T_r(t_0;s)$は$O(t_k)$の項を無視すると、

$$T_r(t_0;s) = ((r-1)^3 + 2) \sum_{k=1}^{p} t_{k-1}^2 \tag{2.4}$$

$$\approx K(r)s^2 ,$$

$$K(r) = \frac{(r-1)^3 + 2}{r^2 - 1}, \quad p = \log_r(s/t_0) \tag{2.5}$$

として与えられる（rが奇数のときも全く同じ結果が得られる）。計算量を決定する定数$K(r)$の値を表1に示す。この表より、「可変長演算」を用いてr次収束法を計算した場合、解法群 (2.1) の中では2次収束法が最も高速になりことがわかる。また、$K(2)=1$ であるから、2次収束法を用いると乗算一回に相当するだけの時間計算量で平方根が求まることになる。

表1．rと$K(r)$の関係

r	2	3	4	5	6	7	8
$K(r)$	1.00	1.25	1.93	2.75	3.63	4.54	5.48

3．「逆平方根法」との比較

平方根$\sqrt{a}$の計算法としては、平方根の逆数に収束する2次収束法を用いる方法がある。これをここでは「逆平方根法」と呼ぶことにする。このアルゴリズムは除算を含まないため、除算が乗算に比べ遅い場合には有利になることもある。ここではこの方法を改良した方法［3］の時間計算量を解析し、r次収束法(2.1)と比較する。

計算法を具体的に示すと以下のようになる：

(1)$1/\sqrt{a}$に収束する次の2次収束法を収束の一歩手前で止める。

$$y_{k+1} = y_k + 0.5y_k(1 - ay_k^2),\ k=0,1,\cdots,p-2 \tag{3.1.1}$$

(2)次に、

$$x_{p-1} = y_{p-1}*a \tag{3.1.2}$$

を計算し、

(3)最後の仕上げは

$$x_p = x_{p-1} + 0.5y_{p-1}*(a - x_{p-1}^2) \tag{3.1.3}$$

を用いる。

この反復法を用いて計算する場合、以下に示す桁数で計算すれば2次収束性が保証される。

先ず、反復法(3.1.1)の第$k+1(k=0,1,\cdots,p-2)$ステップでは

演算	桁数	時間(a=多)	時間(a=単)
y_k^2	$2t_k$	$2t_k^2$	$2t_k^2$
$a*y_k^2$	$2t_k$	$4t_k^2$	0
$y_k*(1-ay_k^2)$	t_k	t_k^2	t_k^2

最終ステップでは、

演算	桁数	時間(a=多)	時間(a=単)
$x_{p-1}=y_{p-1}*a$	$s/2$	$s^2/4$	0
x_{p-1}^2	s	$s^2/2$	$s^2/2$
$y_{p-1}*(a-x_{p-1}^2)$	$s/2$	$s^2/4$	$s^2/4$

となる。ただし$s=2^p t_0$である。

これより、aが多倍長数のときは、

$$T_2(t_0;s) = \sum_{k=1}^{p-1} 7t_{k-1}^2 + s^2 \qquad (3.2.1)$$

$$\approx 1.58s^2$$

であり、一方、単精度数のときは

$$T_2(t_0;s) = \sum_{k=1}^{p-1} 3t_{k-1}^2 + \frac{3s^2}{4} \qquad (3.2.2)$$

$$\approx 1.00s^2$$

となる。したがって、「逆平方根法」はaが単精度数のときは、2次収束法（式(2.1)で $r=2$としたもの）と同等であるが、aが多倍長数のときはあまり高速でないことがわかる。

4.数値実験

次に、著者が自作した多倍長浮動小数点演算のルーチンを用い、r次収束法(2.1), (3.1)を用いて実際に平方根を計算し、各計算法の効率を比較する。ここでは、10進8桁程度の初期値を与え、各種の反復法を何回か反復し、 32,768桁を超えない最大精度に達した時点で反復を止めた。計算機は東北大学大型計算機センターのACOS 2020 を用い、言語はFORTRAN 77(OPT=0)を用いた。ここで用いた多倍長演算のルーチンの演算時間は、 32,768桁の浮動小数点数の乗算に要する時間が15,436[msec]、同じく除算に要する時間が14,470[msec]、二乗に要する時間が7,564[msec]である。

用いたアルゴリズムは以下の通りである：

逆平方根法（ｒ２）

$$y_{k+1} = y_k + 0.5y_k*(1 - a*x_k^2)$$

以下$Y_k = x_k^2$と置く。

2次収束法（ｓ２）

$$x_{k+1} = x_k + \frac{(a - Y_k)}{2x_k}$$

3次収束法（ｓ３）

$$x_{k+1} = x_k + \frac{2x_k*(a - Y_k)}{3Y_k + a}$$

4次収束法（ｓ４）

$$x_{k+1} = x_k + \frac{(a - Y_k)*(3Y_k + a)}{4x_k(Y_k + a)}$$

5次収束法（ｓ５）

$$x_{k+1} = x_k + \frac{4x_k*(a - Y_k)*(a + Y_k)}{(5Y_k + 10a)*Y_k + a^2}$$

6次収束法（ｓ６）

$$x_{k+1} = x_k + \frac{(a - Y_k)*((5Y_k + 10a)*Y_k + a^2)}{x_k*((6Y_k + 20a)*Y_k + 6a^2)}$$

表2.計算結果

method	s	time(msec)	time/s^2 (ratio)
r 2(単)	30,095	14,498	1.60×10^{-5} (1.19)
r 2(多)	30,288	23,394	2.55×10^{-5} (1.89)
s 2	32,260	14,081	1.35×10^{-5} (1.00)
s 3	17,225	5,244	1.77×10^{-5} (1.31)
s 4	32,260	28,379	2.73×10^{-5} (2.02)
s 5	24,613	23,793	3.93×10^{-5} (2.91)
s 6	10,207	5,838	5.60×10^{-5} (4.18)

$s = -\log_{10}$ |相対誤差|

5.おわりに

本報告では、乗算との時間比がほぼ１となる除算法を用い、「可変長演算」によって平方根を計算し、各種解法の時間計算量の比較検討を行なってきた。その結果、除算を含まないため有利であると考えられてきた「逆平方根法」は、多倍長数の平方根を計算するときは必ずしも高速で

ないことが判明した。一方、従来から用いられてきた２次収束法（ニュートン法）は、単精度数の平方根を計算するときも多倍長数の平方根を計算するときも、あらゆる解法の中で最高速であることが判明した。この２次収束法を「可変長演算」で計算すると乗算１回に相当する計算量で平方根が計算できることが判明した。

今後はＦＦＴを用いた乗・除算法による平方根の計算法を検討する必要があろう。

＜参考文献＞

［１］小沢一文：多倍長演算のための平方根の高次収束法、情報処理学会論文誌 **31** (1990),953-963.

［２］小沢一文、海野啓明：多倍長浮動小数点数の除算を高速に行なう方法 — $O(n^2)$ のアルゴリズムに関して —、情報処理学会数値解析研究会資料 **NA32-3** (1990),17-24.

［３］二宮市三（中部大）、山下真一郎（富士通）両氏のコメントによる。