

多倍長演算のための基本関数サブルーチン

平 山 弘*

Multiple-Precision Subroutines for Elementary Functions

Hiroshi HIRAYAMA

Abstract

Fortran multiple-precision subroutines for evaluating elementary function are described. These subroutines are almost machine independent and the precision is arbitrary. The design and performance of the subroutines are discussed, and numerical examples are given.

1. は じ め に

多倍長計算による高精度の演算の研究^{1~5)}は、電子計算機が発明された当時から多くの研究者によって行われている。高精度計算が安価にかつ高速に出来るならば、科学技術計算プログラム作成は非常に容易になり、また信頼性のあるプログラムを作ることができる。しかしながら、多倍長による高精度計算は、電子計算機の主記憶装置を大量に使うだけでなく著しく計算速度を低下させる。このため、これまで特別な場合を除き多倍長計算を実用問題に使われることはほとんど無かった。

最近の著しい半導体技術は、計算機の高速度と低価格を実現している。このような状況から10年前では実用にはならなかった多倍長計算が多くの問題で使うことができるようになってきた。このように安価で非常に速い計算機が大量に使えることを前提にした多倍長の計算プログラムはまだ存在しない。大中・安井⁴⁾や R.P. Brent¹⁾のプログラムが有名であるが、大中・安井のプログラムは、そのプログラム開発後多くの効率の良いアルゴリズムが発見されているため、精度があまり高くない場合には現在の計算方法と遜色ないと思われるが、精度が高い場合効率的ではない。FORTRAN 77 が出来る以前のプログラムなので、文字を

扱うことが出来ないためか多倍長数の入出力は使い易くはなっていない。誤差評価が可能であるが、関数計算は出来ない。Brent のプログラムは効率的なアルゴリズムが多数発見された後のプログラムであるが、その当時の計算機の制約のためか、目的の計算桁数が小さいため効率的なアルゴリズムは使っていない。入出力は大中・安井と同様に FORTRAN 77 が出来る以前のプログラムであるためあまり使いやすくなっていない。関数計算に関しては非常に豊富で、基本的な初等超越関数だけでなく、ベッセル関数、ガンマ関数など高度な関数が豊富に準備されている。

著者⁷⁾は、以前高速フーリエ変換を利用した多倍長数の基本演算とその入出力プログラムを開発した。10進で600から1,000桁を越えた場合この演算ルーチンによって高速に計算できることを示した。このプログラムの開発段階で関数計算プログラムの必要性は感じていたが時間的な制約で作ることが出来なかった。今回、多倍長の基本的な超越関数である指数関数、対数関数、三角関数、双曲線関数を作る事が出来たので、その計算方法およびその性能について論じる。また、以前プログラムからの改良についても論じる。最後にそれを利用した数値例を示す。

2. 高速フーリエ変換を利用した乗算

高速フーリエ変換を利用した多倍長数の積については多くの文献⁶⁾で論じられている。このような計算に

は、整数論のガロア体を利用した FFT が考えられるが、この計算には誤差の無い厳密な計算が可能であるがこのとき使う FFT のルーチンでは、剰余計算のために多数の除算が必要であるため、あまり高速化は期待できない。通常の FFT サブルーチンを利用した場合、その計算に誤差が入るため、計算が不可能のように思えるが、最終的な結果が整数になることがわかっているため、誤差が 0.5 より小さいならば丸め処理によって厳密な計算が可能である。

この時必要な FFT ルーチンは、正 FFT ルーチンとその逆 FFT ルーチンである。FFT には、2, 4, 8 等を基数としたものがあるが、基数 4 を使う計算方法は基数 2 を使う方法より効率が良い⁸⁾。基数 8 を使う方法は基数 4 よりも理論上ほんの少し効率が良い。以前のプログラムは基数 2 による FFT を利用したプログラムであったが、今回は基数 2, 4, 8 を使う FFT を利用するように改良した。この結果、乗算の計算速度はおよそ 2 倍に高速化することができた。通常の計算方法の部分も無駄な演算をしないように細かな点の改良によって、およそ 2 倍の高速化がはかることができた。このため、通常の乗算より FFT を使う乗算が高速になる精度は今までの精度とほぼ同じで 10 進数で約 630 桁ある。2 乗の計算は 2 つの異なる数の乗算と比較して通常の方法で 2 倍、FFT を使う乗算方法で約 1.5 倍に高速化することができるので、FFT が通常の計算方法より高速になるのはさらに高精度の計算の時である。それは 10 進数で約 1080 桁である。

これらの数値は、インテル社のマイクロプロセッサ (80286+80287) を使う計算機で求めたものであるが、モトローラ社のマイクロプロセッサ (68030+68881) を使う計算機でもほぼ同じ数値であった。

このような結果から、作成した多倍長計算ルーチンでは、乗算では 630 桁以下なら通常の計算方法を使い、630 桁を越えた場合 FFT を使うようになっている。2 乗の計算ルーチンでは、乗算ルーチンと同じように 1,080 桁以下ならば、通常の計算方法で計算し、それを越えた場合、FFT を使うようにしてある。

除算は、除数の逆数近似値を求め、以下に示す反復公式 (2.1) によって精度をあげ、それに被除数を掛ける方法を使っている。

$$x_{n+1} = x_n \cdot (2 - a \cdot x_n) \quad (2.1)$$

ここで、 a は除数である。この公式を

$$x_{n+1} = 2 \cdot x_n - a \cdot x_n^2 \quad (2.2)$$

のように変形して 2 乗ルーチンを使うように変形し高速化を計った。FFT を使った場合と使わない場合の計算時間を表に示した。2 つの数の積、2 乗および除算の時間をそれぞれ表 1, 表 2, 表 3 に示した。この測定に使った計算機は Apollo DN-4000 (MC-68020+68881 25MH) である。

Table 1. Timing data (in milli-seconds) of Multiplication of Two numbers of significant decimal digits of N by FFT and Conventional Methods. (on Apollo DN-4000)

N	Conv.	FFT
40	2	23
80	4	23
160	11	24
320	38	57
640	139	111
1280	536	243
2560	2114	564
5120	8388	1202
10240	33394	2588
20480	133239	5678

Table 2. Timing data (in milli-seconds) of Square of a number of significant decimal digits of N by FFT and Conventional Methods. (on Apollo DN-4000)

N	Conv.	FFT
40	1	16
80	2	16
160	7	17
320	20	41
640	71	92
1280	266	184
2560	1042	402
5120	4112	871
10240	16264	1872
20480	65756	4079

Table 3. Timing data (in milli-seconds) of Dividing of two numbers of significant decimal digits of N by FFT and Conventional Methods. (on Apollo DN-4000)

N	Conv.	FFT
40	14	
80	27	
160	77	
320	245	244
640	870	782
1280	3266	1827
2560	12685	4153
5120	50111	9241
10240	199101	20040
20480	796914	43885

3. 基本関数の計算

作成したプログラムは、指数関数、対数関数、三角関数、逆三角関数、双曲線関数、逆双曲線関数である。これらの関数は、すべて指数関数とほぼ同じ性質を持つので、ほぼ同じような計算方法を使った。

このような関数については、R.P. Brent が相加相乗平均の極限値を利用した高速計算方法²⁾を提案してい

Table 4. Timing data (in milli-seconds) of square root of a numbers of significant decimal digits of N by partial FFT and full FFT Methods. (on Apollo DN-4000)

N	part	FFT
40	23	114
80	40	151
160	116	224
320	377	505
640	1190	1192
1280	3039	2605
2560	8302	5770
5120	24400	12686
10240	77446	27333
20480	269990	59676

Table 5. Timing data (in milli-seconds) of exp(x) and log(x) of a numbers of significant decimal digits of N by FFT and conventional Methods. (on Apollo DN-4000)

N	exp (1)	log (2)
40	41	89
80	100	189
160	429	762
320	2364	3938
640	12457	22057
1280	49642	89258
2560	205434	375944
5120	853495	1592707

る。また、T. Sasaki and Y. Kanada は、対数関数についてさらに高速な計算方法³⁾を提案している。このような計算方法は、将来このルーチンがさらに高精度の計算に使われるようになった場合、必須なものになると思われるが現在のところこれらのアルゴリズムは使っていない。これらの高速アルゴリズムについては次の段階で導入する予定である。

これらのプログラムを使って、exp (1.0) および log (2.0) を計算したときの実行時間を表 5 に示した。

3.1 exp(x) の計算

ここでは exp(x) の計算を例にとって説明する。exp(x) を計算するには、収束半径が無限大だからといって、非常に大きな値をもつ x を Taylor 級数に代入することは非実用的である。実際には、

$$e^x = (e^{\frac{x}{2}})^2 \quad (3.1)$$

を利用して、x の値の範囲を小さくして、

$$|x| \leq 2^{-p} \quad (3.2)$$

となるようにする。ここで、p は 2 進数で表したときの計算精度である。(計算精度が p 進数で p 桁であることを意味する。)(3.2) の結果は、与えられた精度を得るために計算しなければならない Taylor 級数の項数と (3.1) を使って元の x の値に戻すための計算量の和が最小になるように選ぶことによって得られる。Taylor 級数の 1 項多く計算を進めるための計算時間と (3.1) を利用して 2 倍の変数の値に対応する関数値を求める

ための計算時間は同じだと仮定している。実際には、精度が低い場合、通常の計算を使うので、(3.1)の2乗計算が約2倍速く計算出来る。高精度の場合、FFTを使うので(3.1)の2乗計算は約1.5倍速く計算出来る。従って、その仮定は正しくないことになるが、基本的な考え方が同じなので話を簡単化のためにこの仮定が正しいものとして論じる。

(3.2)の式の値は、 p が動いてもその変化はあまり大きくない。このため、(3.2)の数値は一定の数値としてプログラムを作成している。計算精度が非常に高い場合、この数値をうまく選ぶと計算速度は2倍以上高速化ができる。

x の値が大きくなると、 x を(3.1)の関係式を何回も適用し、 x を小さくしなければならぬ。このため計算時間が増加する。通常の計算機では、 $\exp(x)$ の計算出来る範囲は非常に小さいので、このような計算時間はあまり問題にならないが、大きな指数部をもつ多倍長の浮動小数点数の場合にはこのための計算時間は無視する事ができない。80桁の精度でいろいろな大きさの x について計算したとき、 x の値を10倍または10分の1にすると計算時間は $x=1$ のときの計算時間のおよそ7.7%それぞれ増加、減少する。

$\exp(x)-1$ は x が小さいとき、 $\exp(x)$ から1を引くと桁落ちが生じ、その精度は著しく悪くなる。このために、 $\exp(x)-1$ を直接計算するためのサブルーチンを準備した。 x が小さい場合、Taylor展開によって計算するのは、 $\exp(x)$ の計算と同じであるが、(3.1)の式による元の x の関数値の計算は使う事ができない。このために、次の式を使った。

$$e^{2x}-1=(e^x-1)^2+2(e^x-1) \quad (3.3)$$

この式は、(3.1)と比較してかなり複雑に見えるが、多倍長の数値の積が1回含むだけで、計算時間はほぼ(3.1)と同じである。

3.2 対数関数

$\log(x)$ の計算は、この値の近似値 y をFORTRANの中の標準関数を使って求めることができる。この値を使って、 $\log(x)$ の x の範囲を非常に小さくすることができる。次の関係式

$$\log(x)=y+\log(x/\exp(y)) \quad (3.4)$$

を利用して、 $\log(x)$ を求める。(3.4)の右辺の第2項の括弧の中はほとんど1であるから、 $\log(1+x)$ の

Taylor展開を利用して計算することができる。 $\exp(y)$ の計算は3.1の $\exp(x)$ を利用する。 $\exp(x)-1$ の逆関数として、 $\log 1(x)=\log(1+x)$ も作成した。計算時間は $\exp(x)$ の約1.5倍である。

3.3 三角関数

三角関数は周期 2π の関数であるから、 2π で割ってその余りを求めることによって、 x の範囲を小さくすることができる。 x の値が非常に大きくなると余りを高精度で求められなくなるので、その計算精度も高精度で求められなくなる。 2π で割る計算は、 2π の逆数を計算しておいてそれを掛けることによって求めている。

(3.1)に相当する式は

$$\begin{cases} \sin(2x)=2\sin(x)(\cos(x)-1) \\ \quad +2\sin(x) \\ \cos(2x)-1=-2(\sin(x))^2 \end{cases} \quad (3.5)$$

である。ここで、 $\cos(x)$ の代わりに $\cos(x)-1$ を使っている。このようにすることによって、 x が0に近いときの $\cos(x)$ の精度低下の影響を低減できる。

(3.5)の式からわかるように、 \sin や \cos を単独では計算出来ない。必ず同時に計算されることになる。 $\tan x$ は

$$\tan x = \frac{\sin x}{\cos x} \quad (3.6)$$

によって計算する。

3.4 逆三角関数

$\tan^{-1}x$ の近似値 y をFORTRANの標準関数を利用して求め、以下の関係式を使って変数の範囲を小さくする。

$$\tan^{-1}x = y + \tan^{-1} \frac{x - \tan y}{1 + x \tan y} \quad (3.7)$$

(3.7)の第2項の分子はほとんど0であるから、Taylor展開で計算してもかなり速く収束する。

$\sin^{-1}x$ の計算は、 $\sin^{-1}x$ と $\tan^{-1}x$ の関係式

$$\sin^{-1}x = \tan^{-1} \frac{x}{\sqrt{1-x^2}} \quad (3.8)$$

から計算する。 $\cos^{-1}x$ は $\sin^{-1}x$ との関係式

$$\cos^{-1}x = \frac{\pi}{2} - \sin^{-1}x \quad (3.9)$$

から計算する。

3.5 双曲線関数

双曲線関数は、関係式のほとんどが三角関数とほぼ同じである。双曲線関数は周期関数でないので指数関数と同じように、関数の独立変数を次々と2等分し、(3.3)の関係式が成り立つような領域に帰着させる。この値の $\sinh(x)$ と $\cosh(x)-1$ を求め以下のような倍角の公式を使って元の変数の値を求める。

$$\begin{cases} \sinh(2x) = 2 \sinh(x) (\cosh(x) - 1) \\ \quad + 2 \sinh(x) \\ \cosh(2x) - 1 = 2 (\sinh(x))^2 \end{cases} \quad (3.10)$$

三角関数と同様に \sinh だけ \cosh だけの計算は出来ない。したがって \sinh と \cosh が同時に計算される。

$\tanh x$ は次の関係式から求められる。

$$\tanh x = \frac{\sinh x}{\cosh x} \quad (3.11)$$

3.6 逆双曲線関数

$\sinh^{-1} x$ はつぎの関係式から \log の計算に帰着させ計算する。

$$\sinh^{-1} x = \log(x + \sqrt{1+x^2}) \quad (3.12)$$

$\cosh^{-1} x$ はつぎの関係式から \log の計算に帰着させ計算する。

$$\cosh^{-1} x = \log(x + \sqrt{x^2-1}) \quad (3.13)$$

$\tanh^{-1} x$ はつぎの関係式から \log の計算に帰着させ計算する。

$$\tanh^{-1} x = \frac{1}{2} \log \frac{1+x}{1-x} \quad (3.14)$$

4. 関数計算サブルーチン一覧

初等超越関数を計算するためのサブルーチン一覧を以下に示す。A, B, C は多倍長の数値である。

- | | | | |
|-----------|--------------------|-------------|----------------------------|
| 1) 円周率 | CALL RKPI (A) | 4) 正弦関数 | CALL RSIN (A, B) |
| | A=3.14159265358... | | B=SIN (A) |
| 2) 指数関数 | CALL REXP (A, B) | 5) 余弦関数 | CALL RCOS (A, B) |
| | B=EXP (A) | | B=COS (A) |
| 3) 指数関数 1 | CALL REXPI (A, B) | 6) 正接関数 | CALL RTAN (A, B) |
| | B=EXP (A)-1 | | B=TAN (A) |
| | | 7) 三角関数 1 | CALL RSCN (A, B, C) |
| | | | B=SIN (A), C=COS (A) |
| | | 8) 三角関数 2 | CALL RCOS1 (A, B) |
| | | | B=COS (A)-1 |
| | | 9) 三角関数 3 | CALL RSCN1 (A, B, C) |
| | | | B=SIN (A), C=COS (A)-1 |
| | | 10) 逆正弦関数 | CALL RASIN (A, B) |
| | | | B=ASIN (A) |
| | | 11) 逆余弦関数 | CALL RACOS (A, B) |
| | | | B=ACOS (A) |
| | | 12) 逆正接関数 | CALL RATAN (A, B) |
| | | | B=ATAN (A) |
| | | 13) 双曲線関数 1 | CALL RSINH (A, B) |
| | | | B= SINH (A) |
| | | 14) 双曲線関数 2 | CALL RCOSH (A, B) |
| | | | B= COSH (A) |
| | | 15) 双曲線関数 3 | CALL RTANH (A, B) |
| | | | B= TANH (A) |
| | | 16) 双曲線関数 4 | CALL RSCNH (A, B, C) |
| | | | B= SINH (A), C= COSH (A) |
| | | 17) 双曲線関数 5 | CALL RCOSH1 (A, B) |
| | | | B= COSH (A)-1 |
| | | 18) 双曲線関数 6 | CALL RSCNH1 (A, B, C) |
| | | | B= SINH (A), C= COSH (A)-1 |
| | | 19) 双曲線関数 7 | CALL RASINH (A, B) |
| | | | B= ASINH (A) |
| | | 20) 双曲線関数 8 | CALL RACOSH (A, B) |
| | | | B= ACOSH (A) |
| | | 21) 双曲線関数 9 | CALL RATANH (A, B) |
| | | | B= ATANH (A) |
| | | 22) 自然対数関数 | CALL RLN (A, B) |
| | | | B= LN (A) |
| | | 23) 常用対数関数 | CALL RLN10 (A, B) |
| | | | B= LOG (A) |

24) 対数関数 1 CALL RLN1 (A, B)
B=LN (1+A)

5. 数 値 例

以下に計算例を示す。

5.1 円周率の計算

Salamin and Brent 方法⁹⁾による円周率の計算の収束状況を与える。計算は一回の反復計算で精度が2倍になる状況がわかる。円周率を与えるサブルーチン RKPI はこのルーチンから計算された円周率の値を DATA 文として持っている。反復を20回行うと100万桁の精度の円周率が得られる。このプログラムを記憶容量を小さくするように、変更し、20回の反復を実行すれば100万桁の円周率が得られる。このプログラムをワークステーション sun 4 で実行したところ約12時間で実行できた。この結果は、途中で誤差が入る FFT ルーチンの限界を知る上で非常に重要である。100万桁の精度でも十分に使えることがわかる。

以下の数値は反復計算の回数によってどのように収束するかをみるための計算である。出力の () 中の数値は R.P. Brent の論文における値である。

反復回数	誤 差
1	4.607998912231539E-0002 (4.6E-2)
2	8.763970786005546E-0005 (8.8E-5)
3	3.056532575402714E-0010 (3.1E-10)
4	3.717219427128438E-0021 (3.7E-21)
5	5.497896207848598E-0043
6	1.202688653704705E-0086
7	5.755281465665246E-0174
8	1.317932829037409E-0348
9	6.911092292112558E-0698
10	1.900437211979097E-1396

5.2 非常に整数に近い数値

EXP (PI * SQRT(163)/3) および EXP (PI * SQRT(163)) は非常に整数に近い数であることが知られている。高精度で計算しないと整数かどうか判定できない。

出力は計算機の出力を編集したものである。この数値は高精度で計算しなければ、その値が整数かどうか判定が難しい問題として有名である。ここでは、約 115

桁で計算した。その結果を編集して出力したものである。

PI = 3.1415926535 8979323846 2643383279
5028841971 6939937510 5820974944
5923078164 0628620899 8628034825
3421170679

P(PI * SQRT(163)/3) =
640320.0000000006 0486373504 9016039471
7418188185 3947577148 5760366591
8194652218 2582869425 3634081582
2646477589

EXP(PI * SQRT(163)) =
262537412640768743.9999999999 9925007259
7198185688 8793538563
3733699086 2707537410
3782106479 1011860731
2951181346 1860645041

5.2 主な数値の 50 桁の数表

単純な整数、円周率やオイラー数に関する数値を計算した。以下にその結果を示す。

PI: 円周率 GAM: オイラー数

	主要定数		
PI	3.1415926535	8979323846	2643383279
	5028841971	6939937511	
GAM	0.5772156649	0153286060	6512090082
	4024310421	5933593992	
180/PI	57.2957795130	8232087679	8154814105
	1703324054	7246656432	
X	SQRT (X)		
2	1.4142135623	7309504880	1688724209
	6980785696	7187537695	
3	1.7320508075	6887729352	7446341505
	8723669428	0525381038	
5	2.2360679774	9978969640	9173668731
	2762354406	1835961153	
7	2.6457513110	6459059050	1615753639
	2604257102	5918308245	
11	3.3166247903	5539984911	4932736670
	6866839270	8854558935	
13	3.6055512754	6398929311	9221267470
	4959462512	9657384525	
17	4.1231056256	1766054982	1409855974
	0770251471	9922537362	

19	4.3588989435	4067355223	6981983859		0000000000	0000000000
	6156591370	0392523244		40	0.6427876096	8653932632 2643409907
PI	1.7724538509	0551602729	8167483341		2634329075	5988420568
	1451827975	4945612239		50	0.7660444431	1897803520 2392650555
GAM	0.7597471058	8559194629	7866442501		4166739358	3245708040
	7843733849	6357611951		60	0.8660254037	8443864676 3723170752
X	EXP (X)				9361834714	0262690519
1	2.7182818284	5904523536	0287471352	70	0.9396926207	8590838405 4109277324
	6624977572	4709369996			7314699362	0813426446
2	7.3890560989	3065022723	0427460575	80	0.9848077530	1220805936 6743024589
	0078131803	1557055185			5230136706	4325171984
3	20.0855369231	8766774092	8529654581	90	1.0000000000	0000000000 0000000000
	7178969879	0783855415			0000000000	0000000000
4	54.5981500331	4423907811	0261202860	X (度)	TAN (X)	
	8784027907	3703861407		0	0.0000000000	0000000000 0000000000
5	148.4131591025	7660342111	5580040552		0000000000	0000000000
	2796234876	6759387899		10	0.1763269807	0846497347 1090386868
PI	23.1406926327	7926900572	9086367948		6189861216	3306234810
	5473802661	0624260021		20	0.3639702342	6620236135 1047882776
GAM	1.7810724179	9019798523	6504103107		8340438904	7178375374
	1795491696	4521430343		30	0.5773502691	8962576450 9148780501
X	LOG (X)				9574556476	0175127013
2	0.6931471805	5994530941	7232121458	40	0.8390996311	7728001176 3127298123
	1765680755	0013436026			1813646874	3428301235
3	1.0986122886	6810969139	5245236922	50	1.1917535925	9420995870 5308071860
	5257046474	9055782275			4193369307	0040770854
4	1.3862943611	1989061883	4464242916	60	1.7320508075	6887729352 7446341505
	3531361510	0026872051			8723669428	0525381038
5	1.6094379124	3410037460	0759333226	70	2.7474774194	5462227876 1664026497
	1876395256	0135426852			6727177518	7259917083
6	1.7917594692	2805500081	2477358380	80	5.6712818196	1770953099 4418439863
	7022727229	9069218300			9644216253	7826068975
PI	1.1447298858	4940017414	3427351353	X	ASIN (X)	
	0587116472	9481291531		0.0	0.0000000000	0000000000 0000000000
GAM	-0.5495393129	8164482233	7661768802		0000000000	0000000000
	9077883306	9898126306		0.1	0.1001674211	6155979634 5523179452
X (度)	SIN (X)				6933185686	7597222963
0	0.0000000000	0000000000	0000000000	0.2	0.2013579207	9033079145 5125552217
	0000000000	0000000000			6234102400	3808140223
10	0.1736481776	6693034885	1716626769	0.3	0.3046926540	1539750797 2002961227
	3147960003	7567718407			5291669545	6003170678
20	0.3420201433	2566873304	4099614682	0.4	0.4115168460	6748801938 4737897617
	2595807630	8336751416			3356048557	0113512703
30	0.5000000000	0000000000	0000000000	0.5	0.5235987755	9829887307 7107230546

	5838140328	6156656252	
0.6	0.6435011087	9328438680	2809228717
	3226380415	1059111531	
0.7	0.7753974966	1075306374	0353352714
	9871135557	8873864116	
0.8	0.9272952180	0161223242	8512462922
	4288040570	7410857224	
0.9	1.1197695149	9863418668	6677055845
	3996158951	6218640330	
1.0	1.5707963267	9489661923	1321691639
	7514420985	8469968755	

6. ま と め

精度が 1000 桁以下であるならば、時間および精度の点で十分な性能を発揮する関数計算サブルーチンをつくることができた。FFT が有効に働く 1000 桁以上の計算では、Brent のアルゴリズムがかなり威力を発揮すると思われるので、このアルゴリズムを使ったプログラム開発が必要となる。また、ガンマ関数、誤差関数、ベッセル関数なども作らなければならない。楕円柱関数など普通の関数プログラムとしてもあまり作られていない関数プログラムをこのような多倍長高精度演算ルーチンで作ることは関数の性質の調査や関数値の検証に役立つのでこのようなプログラムは非常に重要である。

このような多倍長計算の有用な応用の一つは最良近似式を求めることである。このようなプログラムの開発も重要だと思われる。

ここで開発したプログラムは、FORTRAN で作成している。この選択は現状では仕方がないものである

が、将来、C++などで作れば多倍長数の計算はサブルーチンの呼び出しの連続したプログラムから通常の数値を扱うような記述になるため、非常に使い易いものになると思われる。

参 考 文 献

- 1) Richard P. Brent: A Fortran Multiple-Precision Arithmetic Package, ACM Trans. Math. Software 4, 1 (March 1978), 71-78
- 2) Richard P. Brent: Fast Multiple-Precision Evaluation of Elementary Functions, J. Assoc. Compt. March. 23 (1976), 242-251
- 3) T. Sasaki and Y. Kanada: Practically Fast Multiple-Precision Evaluation of $\text{LOG}(X)$, J. Inf. Process., Vol. 4, 4 (1982), 247-250
- 4) 大中幸三郎, 安井 裕: 誤差評価の可能な多重精度演算, 情報処理, Vol. 15, No. 5 (1974), 110-117
- 5) 伊藤正俊: 丸め誤差の評価できる多重精度演算ツール, 名城大学理工学部研究報告, No. 30 (1990), 246-251
- 6) A.V. エイホ, J.E. ホップクロフト, J.D. ウルマン (野崎, 野下訳): アルゴリズムの設計と解析, サイエンス社 (1977)
- 7) 平山 弘: 多倍長浮動小数点演算プログラムの開発, 神奈川工科大学研究報告 B 理工編第 13 (1989)
- 8) 森, 名取, 鳥居: 数値計算, 第 5 章, 岩波講座情報科学 18, 岩波書店 (1982)
- 9) E. Salamin: Computation of π using Arithmetic-Geometric Mean, Math. Comp. 30 (1976), 565-570