数理解析研究所講究録
1085巻 1999年 170-180

可変精度多倍長演算ルーチンの応用

神奈川工科大学 平山　弘 (Hiroshi HIRAYAMA) *

1　はじめに

計算機で準備されている浮動小数点数では、すこし大きな計算を行うと、桁落ちのために、有効桁数が著しく小さくなり、計算が無意味なものになることがよく生じる。数式処理等で行われる厳密な計算では、演算桁数が異常に大きくなり計算ができなくなる場合や数式が異常に大きくなり計算できなくなる場合がある。

このような場合、多倍長の浮動小数点数が便利な場合がある。この計算方法は、通常の数値計算と基本的には同じであるため、計算のために必要なメモリの容量等が計算を始める前に予測できるためにその扱いが大変容易である。

このような目的のために多倍長演算ルーチンを開発した。最初のバージョンは、すでにある多くの多倍長演算ルーチン [2][7-9] と同じくFORTRAN77のサブルーチンを呼び出す形式 [4] であったが、プログラムをサブルーチンの呼び出し形式に変換する必要があるため、この変換が容易なC++言語で開発 [5] した。C++言語ではオペレータ・オーバーローディング機能が使えるため、プログラム作成・プログラムの変換が大変容易になる。

たとえば、次のような二次方程式 $2x^2-8x+3=0$ の解の計算プログラムは、倍精度では次のようなプログラムになる。

```
 1:  #include <iostream.h>
 2:  void main()
 3:  {
 4:        double a,b,c,x ;
 5:        a=2;
 6:        b=-8;
 7:        c=3 ;
 8:        x=(-b+sqrt(b*b-4*a*c))/(2*a) ;
 9:        cout << "x=" << x << endl ;
10:  }
```

*hirayama@mse.kanagawa-it.ac.jp

実行結果：

```
x=3.58113883008419
```

これを、多倍長計算プログラムでは、次のようになる。

```
 1:  #include "long_num.h"        // 高精度ルーチンの宣言を読み込む。
 2:  void main()
 3:  {
 4:        long_float a,b,c,x ;  // double の変数を long_float に変更
 5:        a=2;
 6:        b=-8;
 7:        c=3 ;
 8:        x=(-b+sqrt(b*b-4*a*c))/(2*a) ;
 9:        cout << "x=" << x << endl ;
10:  }
```

実行結果：

```
x=3.581138830084189665999446772216359266859777569662 61
```

多倍長プログラムで修正した部分は、右にコメントを書いた二カ所である。この変更で、既定値の精度 128 桁の計算を行うことができる。それ以外は元のプログラムと全く同じである。このような変更を C 言語や FORTRAN77 で行うとすると、かなり作業となる。最新の JIS 規格の Fortran90 では、C++と同様な記述が可能になっている。

2　可変長精度多倍長演算ルーチン

多倍長計算を行っていると、それほど精度を必要としない計算部分が多く現れる。この部分の計算精度下げて、高速化することができるプログラムが必要である。上記のプログラムを、改良し、実行時に精度を指定できるようにした。また、大きな桁数の数値の乗算を行うために、FFT を利用した乗算アルゴズムを使ったルーチンを組み込んだ。Karatsuba の方法は、プログラムは準備してあるが、現在使用している計算機では有効範囲が存在しなかったので、利用していない。複素数の乗算は、精度に関係なく効果的なので、このアルゴリズムを利用している。

プログラムは、機械に依存しない形式で作成しているものと機械に依存する x86 用のものが準備されている。

2.1 多倍長演算ルーチンの構成

最初の目的は、高精度浮動小数の計算であったが、計算の中には、有理数で計算すれば効果的なものがあるので、高精度整数、高精度有理数も準備した。さらに、高精度複素数を作成してある。

2.2 高速 Fourier 変換 (FFT) の利用

高速フーリエ変換 (FFT) を使うと大きな桁数の数値を高速に乗算することができる。本プログラムでは、乗算と二乗計算に、このアルゴリズムを使っている。計算する数値の精度をそれぞれ n、m とすると、低精度では、通常の乗算法（計算量 $O(nm)$）で計算し、ある程度以上の精度では、計算量 $O(\frac{n+m}{2}\log(\frac{n+m}{2}))$ の FFT を使うアルゴリズムを使う乗算法で計算するようになっている。

同じ桁数の数値の掛け算を行うとき、Pentium II 266MHz の計算機では、10000 進数で約 380 桁（10 進数で約 1500 桁）以上のとき、FFT を使った計算法を使い、それ以下では通常の乗算法を使っている。二乗の計算では、通常の計算法が乗算の約半分の計算時間計算できるので、その値は、10000 進数で約 410 桁（10 進数で約 1600 桁）である。計算精度を上げていくと、途中再び通常の計算法が一旦速くなるが、計算時間にほとんど差がないので、その部分でも、FFT を使った計算法を適用している。通常の計算法と FFT を利用した計算法の限界の桁数は、高速計算機ほど、大きな数値になる傾向がある。最初に FFT を利用したプログラムを作成した計算機 (Intel 社 i286+287) では、乗算の限界が 10 進数で約 700 桁、二乗の限界が約 800 桁であった。

Intel 社の i 386 を使い、C++言語でプログラムを作成した場合、精度 n の数値の乗算を計算量 $O(n^{\log_2 3})$ で計算できる Karatsuba のアルゴリズム [3] が有効な精度も存在したが、現在の高速な計算機では、通常の乗算法がキャッシュ等の効果で非常に速くなり、Karatsuba のアルゴリズムが有効な精度領域が存在しなくなった。このため、本パッケージでは、高精度の複素数の乗算法としては、使っていますが、このアルゴリズムは多倍長数の乗算法としては、このアルゴリズムは組み込んでいない。FFT としては、実数用基数 8 の FFT のプログラムを利用している。Pentium II 266MHz の計算機 (PC-9821Rv II+Borland C/C++5.0) で測定した実行時間を表 1 に示す。

2.3 可変精度

可変精度環境では、ユーザーが適切な精度を選択することによって、計算効率をあげることができる。このプログラムでは、精度を自由に設定できるだけでなく、計算結果が、精度が桁落ち等で小さくなる場合、自動的に計算精度が小さくなるように作られている。

表 1: 乗算の実行時間（単位秒）

桁数	FFT	通常
10,000	0.047	0.141
20,000	0.094	0.516
40,000	0.187	2.125
100,000	0.422	13.453
200,000	1.109	53.859
400,000	2.391	215.984
1,000,000	4.937	1830.375

このため、数値 a の逆数を求める計算で、Ne w ton 法を使ったとき、次のような反復を繰り返す。

$$x_{n+1} = 2x_n - ax_n^2$$

この式を

$$x_{n+1} = x_n - x_n(ax_n - 1)$$

と書くと、自動的に高速計算が可能になる。m 桁の数値の乗算時間を M とし、x_n は a の逆数の近似で m 桁の精度であるとする。このとき、上の式では、m 桁の数値 x_n の二乗と $2m$ 桁の数値同士の乗算が必要なので、計算量は、二乗計算は乗算の半分の時間で計算できることを利用すると、$\frac{9}{2}M$ となる。下の式では、$ax_n - 1$ の計算では、桁落ちが生じ、m 桁の数値になる。これに m 桁の x_n を掛けることになる。したがって、この部分では、$2m$ 桁の数値 a と m 桁の数値 x_n の乗算と m 桁の数値同士の乗算が行われることになる。このときの計算量は、$3M$ となる。

この方法は、平方根の逆数を求める計算でも有効である。通常

$$x_{n+1} = \frac{1}{2}x_n(3 - ax_n^2)$$

と計算する。これを

$$x_{n+1} = x_n - \frac{1}{2}x_n(ax_n^2 - 1)$$

として計算する。逆数の計算と同じ理由で計算が高速化される。

3 応用例

多倍長計算ルーチンの応用例を挙げる。

3.1 連立一次方程式

連立一次方程式は、四則演算だけで解けるので、高精度有理数を使えば厳密に解くことができる。このような計算は公式を導くときなどよく現れるので、実際の計算でも高精度有理数は、かなり利用され可能性がある。プログラムは、なるべく短くするため、ガウスージョルダンの解法を使っている。計算時間を短くするには、ガウスの解法を使えば短くなる。このプログラムは、基本的に通常の数値計算における連立一次方程式の解法と同じである。約５０行あるプログラムで変更しなければならない部分は、宣言部分だけである。

このプログラムでは、通常のプログラムから変換が容易であることを強調するために、最適化を行っていないが、多倍長数の場合、数値の入れ替えは、ポインターの入れ替えを行った方が速くなる。

3.2 最良近似式

コンパイラーなどに組み込まれる数学関数では、最良近似多項式などを利用して高速に計算するが、この場合、数学関数の精度は、計算精度まで正しいことを期待される。このような近似式を求めるには、高い精度で計算する必要が生じる。

関数 $f(x)$ を n 次の多項式 $p_n(x)$ で近似する。このとき、$f(x)$ と $p_n(x)$ の差を $e_n(x)$ とする。すなわち

$$e_n(x) = f(x) - p_n(x)$$

と定義する。$p_n(x)$ が最良近似式であるとき、$e_n(x)$ は次の性質をもつことが知られている。

- $e_n(x)$ の相隣る極大極小値は符号が異なる。
- $e_n(x)$ の極値は、区間内で少なくとも (n+2) 個ある。
- $e_n(x)$ の極大極小値の絶対値はすべて等しい。

の性質があります。この性質から、$p_n(x)$ を求めることができます。この性質は、有利多項式で近似する場合も成り立ちます。

ここでは、Gamma 関数 $\Gamma(x+2)$ の区間 $-\frac{1}{2} \leq x \leq \frac{1}{2}$ における分子７次分母７次の有利近似式を求めることを考える。

Gamma 関数は、階乗を拡張したもので、

$$\Gamma(s) = \int_0^\infty x^{s-1} e^{-x} dx \tag{1}$$

と定義さる。ガンマ関数は階乗の拡張なので、s が整数値のとき

$$\Gamma(x) = (s-1)! \tag{2}$$

となる。このことは、部分積分によって容易に証明できる。この関数は、次のように漸近展開 [1] できる。

$$\log(\Gamma(s)) = (s - \frac{1}{2})\log s - s + \frac{1}{2}\log(2\pi) + \sum_{m=1}^{\infty} \frac{B_{2m}}{2m(2m-1)s^{2m-1}} \tag{3}$$

ここで、B_{2m} は Bernoulli 数です。この級数は、発散級数なので、級数の値を計算するには、何項かまで計算し、そこで計算を打ち切らなければならない。この級数の値と計算値の違いは、この打ち切った項の大きさ程度となるから、級数の項が十分に小さくなったところで計算を打ち切るようにしなければならない。

級数の項を小さくするには、s を大きくする。s は、ガンマ関数の性質

$$\Gamma(s+1) = s\Gamma(s) \tag{4}$$

を何回か使って、n を整数とすると

$$\Gamma(s) = \frac{\Gamma(s+n)}{s(s+1)(s+2)\ldots(s+n-1)} \tag{5}$$

が得られます。この公式によって、計算するガンマ関数の s の値をいくらでも大きくすることができる。この s を使って、発散級数 (3) からガンマ関数の値を求める。、(5) の式から元の s のガンマ関数の値を計算する。s を十分に大きいならば、(3) の級数は速く収束し、計算が容易になるが、(5) によって元の s の値の関数値に戻すのが大変になる。この兼ね合いから、(3) の式を計算する最適な s の値を決めることができる。最適な s の値は、計算精度に依存する。

(3) の級数の計算には、ベルヌイ数が必要なので、実際の計算に使えるベルヌイ数の個数などの制限が付く。ここでは、(3) の級数を計算する時の s の値を計算精度 (10 進数で) の 150 パーセントとして計算している。計算精度を 10 進数で 100 桁ならば、s として 150 を選ぶことになる。計算精度 200 桁以内ならば、この方法で計算できる。

計算方法は、山下の方法 [6] によって計算した。
結果は、

```
 分子の係数   A=
0      1.00000000000000000055621491053605825465149523
1      8.87739364132926796030972682246349015708 68e-01
2      4.14325944493523124264794711643188166770 07e-01
3      1.33620185777182017282576756154657525060 16e-01
4      3.06916617382709265946963497426879751045 01e-02
5      5.41990644416282297021381721558038709662 88e-03
6      6.45639092888369872669666964419622242478 00e-04
7      5.50186802413990693873076811915229342058 02e-05
 分母の係数   B=
0      1.00000000000000000000000000000000000000000000
1      4.64955029034459615456525490167704690584 45e-01
```

```
 2     -1.940900887339388351642337795530335312307 5e-01
 3     -5.738571714630403590387874653295800282933 9e-02
 4      2.708960672202071825808355361931172577992e-02
 5      1.030468929992717432515539764901916145179 9e-03
 6     -1.080056858468987196135885870589939949376 7e-03
 7      1.102644168834990126145668121328544449032 4e-04
 極値点   X=
 1      5.000000000000000000000000000000000000000 0e-01
 2      4.886915916788727867172092626111707722005 3e-01
 3      4.552676521513130615543343451153804235660 1e-01
 4      4.012382277261837680738073911858866346750 2e-01
 5      3.290012758480127995860927922760008102541 0e-01
 6      2.419476530013301297182987523037233284785 7e-01
 7      1.440709627036103443748247760600600993636 2e-01
 8      3.988914457925056625265529506486589898360 7e-02
 9     -6.547928597843709294487601416495761289811 8e-02
10     -1.673879779602066672343323525790165138763 1e-01
11     -2.613017908553269375403165016026917633465 6e-01
12     -3.432916891076508507599632839463036060835 7e-01
13     -4.101974888561240237227457139981362071646 7e-01
14     -4.595429583564636312285456255072503286922 7e-01
15     -4.898164846034407466975292611992651630826 6e-01
16     -5.000000000000000000000000000000000000000 0e-01
 極値    EX=
 1      1.485226550412034320112966935189990129690 0e-18
 2     -1.485226550421132213861068591452134636330 0e-18
 3      1.485226550437059400299532416489253286830 0e-18
 4     -1.485226550461572216718335293535934346530 0e-18
 5      1.485226550481311153293539080701742405460 0e-18
 6     -1.485226550490078268025834327824775228900 0e-18
 7      1.485226550479536245017397296556196703730 0e-18
 8     -1.485226550464522615043183219868820420550 0e-18
 9      1.485226550460638224495907281489067006979e-18
10     -1.485226550476960884660513053451972657444 8e-18
11      1.485226550492714638725954479099454399382 1e-18
12     -1.485226550472357640933037027591298635101 7e-18
13      1.485226550437417381510200499718730041552 2e-18
14     -1.485226652343259348649664458320361297066 0e-18
15      1.485226550490354220723044640608085274563 1e-18
16     -1.485226550531040230660805543724433226017 7e-18
```

極値は、すべて絶対値が約 1.484×10^{-18} でほぼ同じで、隣の極値とは符号が異なっており、最良近似の条件をほぼ満たしている。最大誤差は、1.484×10^{-18} となるので、倍精度実数の近似公式としては、十分な精度を持っていることがわかる。

計算機による出力は、40 桁以上の精度で出力されているが、近似式の係数としては、最大誤差より小さい係数部分は、不要である。

3.3 Gauss 型数値積分公式

重み関数 $w(x)$ と整数 n を与えたとき、$f(x)$ が $(2n-1)$ 次以下の多項式の時、近似式

$$\int_a^b w(x)f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

が厳密になるように、重み w_i と分点 x_i を探し出す事ができる場合がある。この重み w_i と分点 x_i を使って計算する数値積分法をガウス (Gauss) 型数値積分法と呼ばれている。 上の式からわかるように、重み と分点 が分かれば、その計算は単純で非常に能率的である。高精度結果を与える公式として良く知られている。

3.3.1 いろいろなガウス型積分公式

ガウス型数値積分公式には、特定の分点を利用するようにした公式も存在する。一つの点を固定した積分公式として、ラドー (Radau) の公式が有名である。この公式は

$$\int_{-1}^{1} f(x)dx \approx \frac{2}{n^2}f(-1) + \sum_{i=1}^{n-1} w_i f(x_i)$$

と書ける。 また、2 点を固定した固定した公式として、ロバット (Lobatto) の公式が有名である。この公式は

$$\int_{-1}^{1} f(x)dx \approx \frac{2}{n(n-1)}(f(-1)+f(1)) + \sum_{i=2}^{n-1} w_i f(x_i)$$

と書ける。上の一点固定および二点固定の公式は、いずれもガウス型積分において、$w(x)=1$ の最も簡単な場合で、それ以外の具体的な研究は、見受けられない。

3.3.2 ガウス型積分公式の問題点

これらの公式の問題点は、重み w_i と分点 x_i を計算しておかなければならない点である。この計算は、通常、高次の代数方程式となる。このため、通常の精度で計算した場合、十分な精度が得られない場合や、計算が収束しないことがある。ガウス型数値積分公式が効率的であることがわかったとしても、具体的に、重み w_i と分点 x_i が計算できないために、ガウス型積分公式を使えない場合や、使えたとしてしても、十分な次数の公式を使えない場合がある。このような高次代数方程式の問題点に対して多くの研究が成されているが、高精度で計算するのが最も確実な方法と思われる。

3.3.3 ガウス型積分公式の分点と重みの計算プログラム

一般にガウス型数値積分公式と言うと、ガウスールジャンドル、ガウスーラゲール、ガウスーエルミートおよび上のラドー、ロバット型の積分公式であるが、ここでは一般に、m 点固定の n 次のガウス型数値積分公式を作るプログラムを作成した。　重み関数 $w(x)$ が、積分区間で、正（一定符号）であれば、ガウス型数値積分公式が存在することが知られているが、ここでは、このような仮定を設けないで、重み関数が積分区間内で符号を変えるような場合も適用できるようなプログラムにした。　プログラムで与えなければならないのは、モーメント M_j と固定点の座標 x_j である。モーメント M_j は

$$M_j = \int_a^b w(x)x^j dx \qquad j = 0, 1, \ldots, k$$

で与えられるものである。$2N$ 個のモーメント M_j、m 個の固定点の座標 x_i を与えると、$(m+N)$ 個の分点を持つ $(2N+m)$ 次のガウス型数値積分公式を計算するようになっている。また、

$$\int_a^b w(x)f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + Ef^{(2n)}(c) \qquad a \le c \le b$$

としたときの、係数 E も計算できるようになっている。これらの計算途中で解かなければならない代数方程式は、すべての根が実数なので、ニュートン法を使って解いている。もし、根が複素数である場合には、計算量が増えて、ガウス型数値積分公式の利点を失うので、このような計算は行っていない。

根は、絶対値が大きい方から求めている。このように計算すると減次のとき、誤差が大きくなることが知られているので、係数を逆に並べた逆数を根にもつ方程式を使って、その方程式の減次を行っている。

3.3.4 計算例 1

次のような 1 個の固定点を与えた積分のガウス型数値積分公式を計算する。

$$\int_0^1 \log(\frac{1}{x})f(x)dx = w_c f(x_c) + \sum_{i=1}^{n} w_i f(x_i)$$

ここで、$n = 7$、$w_c = 0.3$ とする。

```
    Abscissas =
c  0.3000000000 0000000000 0000000000 0000000000 0000000000 0000000000 0000000
1  0.0000903853 6467977222 3371202489 2232819682 4450282475 0596494859 1047901
2  0.0415692073 7737008675 7468170497 7584522602 3150520124 2678066343 2727469
3  0.1459917131 4346717064 6206816898 8887488831 6522817733 8498231388 8313734
4  0.4827511628 0799830262 8863677874 1391463312 5683014159 7828526103 3537790
```

```
5  0.6686676300 4700496657 8720568740 0339859987 5991651980 0325398879 6343584
6  0.8313440243 8100330859 3639326659 0962189942 2059681296 9988568331 1565671
7  0.9474860113 7383926986 8558661119 6241195178 9504487544 4147034192 1391941
     Weight Factors =
c  2.0736447667 5058303544 5860984435 4805214527 9946969491 2998834676 5331547e-01
1  6.1845552088 0237927119 6791045151 6429200094 8833063090 1139484402 1035799e-02
2  2.3569043657 2431283866 7274538560 2598150219 0630112109 6832097823 5532628e-01
3  2.5463046819 3659771132 7604779017 3966318764 9741695814 7881744545 6011315e-01
4  1.3756768197 0803041954 6207100873 6043838690 2748183426 5517180986 6966867e-01
5  7.1836022012 3985538135 9230481955 9047942299 8372024971 4299526793 0211031e-02
6  2.6396252914 6959751765 1965879509 4824397277 6435417855 0848140466 0514154e-02
7  4.6691095729 2927779922 5385645155 5632383045 6898856601 4142680149 8154413e-03
```

このように、固定点を与えたガウス型積分公式が計算できる。このような計算は常に存在するわけではないが、このように求められる場合多い。ただし、上の例題のようない分点が積分区間内にあるようなケースは少ない。

3.3.5 計算例2

次のような積分の5次のガウス型数値積分公式を計算する。誤差評価の係数も計算する。

$$\int_0^1 \log(\frac{1}{x}) f(x)dx = \sum_{i=1}^{n} w_i f(x_i) + Ef^{(2n)}(c) \qquad a \leq c \leq b$$

```
     Abscissas =
0.0291344721 5197205330 3726762115 4416923961 0793130238 2446684650 5773601
0.1739772133 2089762870 1139710828 5651744303 9224068336 5511959418 0932522
0.4117025202 8490204317 4931924645 6586476161 5211948171 3129100855 4415325
0.6773141745 8282038070 1802667998 0061802828 9928763756 3967397051 4995032
0.8947713610 3100828363 8886204455 1724608950 8556524206 4779090429 9594718
     Weight Factors =
2.9789347178 2894457272 2578778879 3820241638 4126309288 7522186802 2577679e-01
3.4977622651 3224180375 0718703073 0988562653 9184965038 1263406036 7168586e-01
2.3448829004 4052418886 9068579425 9698649663 7927295106 2288596794 6294803e-01
9.8930459516 6331469761 8071144039 5543862072 2363956020 8977546562 3020884e-02
1.8911552143 1957964895 8268242175 9381598366 5250349648 0280557101 6568429e-02
      E   =
1.8151588908 2490819283 1808603735 3810612374 5609032510 4612407618 0468701e-13
```

このように、誤差評価のための係数も容易計算できる。

4 まとめ

高精度計算プログラムを利用すると、通常の精度では非常に困難な計算を容易に計算できる。このような計算ができれば、最良近似、ガウス型数値積分公式などが容易に作るこ

とができ、通常の精度の計算も効率的な計算が可能である。

5 参考文献

[1] M. Abramowitz and I. A. Stegun: Handbook of Mathmatical Function, Dover, 1965

[2] R.P.Brent : A Fortran Multiple-Precision Arithmetic Package, *ACM Trans. Math. Soft.*, **4**, 1978,57–70

[3] Karatsuba, A. and Ofman, Y. : Multiplication of mutidigit numbers on automata, *Doklady Akad. Nauk SSSR*, **145**, 1962, 293–294

[4] 平山 弘 : 多倍長計算プログラムパッケージ MPPACK, 東京大学大型計算機センター, 1992

[5] 平山 弘 : C++言語による高精度計算パッケージの開発, 日本応用数理学会, **5**(3), 1995, 123–134

[6] 山下 眞一郎 : 有理式の最良近似を求めるプログラム, 情報処理, **10**, 1969, 442–446

[7] 大中幸三郎, 安井裕 : 誤差評価可能な多重精度演算, 情報処理, **15**, 1974, 110–117

[8] Smith, D.M. : A FORTRAN Package For Floating-Point Multiple-Precision Arithmetic, *ACM Trans. Math. Soft.*,**17**, 1991, 273–283

[9] Watt, W.T., Lozier, P. W. and Orser, P. J. : A Portable Extended Precision Arithmetic Package and Library with Fortran Precompiler, *ACM Trans. Math. Soft.*, **2**, 1976, 209–231