Vol.2018-HPC-163 No.5
2018/2/28

Binary Splitting Algorithmによる初等超越関数の高精度計算

平山 弘[1,a)]

概要：いろいろな名前で呼ばれている小さな数からなる分数等は、そのまま分数等で計算し、大きい桁数の数の計算を避け高精度計算を効率的に計算する方法がある。級数縮約法、分割有理数化法、Binary Splitting Algorithm(BSA 法、二分法）等と呼ばれている。基本的な原理は同じである。
この計算法は、計算量が n が計算精度だとすると、$O(n(\log n)^3)$ と非常に高速に計算可能性のある方法である。ある特定の数値の計算だけでなく、一般の数値計算の可能性について調べた。
この方法を使って、いろいろな精度の数値の初等超越関数（指数関数）を計算し、その有効性を調べた。
1 桁の数値で 10 進数 400 桁程度までの計算する場合、BSA 法はそれほど有効ではない。40 桁以下の数値で与えられた数の指数関数は、もし、5000 桁以上計算するならば非常に有効である。

キーワード：Binary Splitting Algorithm，高精度計算，初等超越関数

Highly precise calculation of elementary transcendental function by Binary Splitting Algorithm

Hiroshi Hirayama[1,a)]

Abstract: Fractions which consist of the small number called under the various names are calculated for a fraction just as it is. There is a way to avoid numerical calculation of the big number of figures and calculate highly precise calculation efficiently.
This method is callded Reduction of Series Method, Divide and Rationalize Method and Binary Splitting Algorithm (BSA way and dichotomy). The basic principle of these methods is same.
When n is the calculation precision, the calculated amount is $O(n(\log n),^3)$ and the way to have a calculation possibility at high speed by these calculation methods. It was checked about a possibility of the general numerical value calculation as well as the calculation of the specific numerical value.
Elementary transcendental function of the numerical value of the various precision (exponential function) was calculated and its validity was checked using these way. BSA way isn't also so effective in a calculated case to about 400 digits of decimal number for 1 digit of numerical value. When it's calculated more than 5000 digits, the exponential function of the number to which it was given by less than 40 digits of numerical value is very effective.

Keywords: Binary Splitting Algorithm, High precision calculation, elementary transcendental function

1. はじめに

級数の高精度計算を有理数を使って小さい桁数の数値にして計算することによって、計算速度を上げる方法が多くの人によって使われている。この方法は 1998 年頃から、右田等の級数縮約法 [5][6]、後等の分割有理数化法

[1] 神奈川工科大学創造工学部自動車システム開発工学科
Department of Vehicle System Engineering, Faculty of Creative Engneering, Kanagawa Institute of Technology, Shimo-Ogino 1030, Atsugi, Kanagawa, 243-0292, Japan
[a)] hirayama@kanagawa-it.ac.jp

(Divide and Rationalize Method(DRM 法))[7] や平山 [4] によって、級数計算への適用の再現が行われている。この計算法の起源を調べると 1997 年にすでに B.Haible and T.Papanikolaou[2] によって公表されていることがわかる。この論文では、トーナメント法とか縮約法とか呼ばれているこれらの計算方法を Binary Splitting Algorithm（以降 BSA 法と略す）と呼んでいる。この論文では、右田等、後等や平山で扱われている級数より一般的な級数を高速に計算する方法を提案している。

$$U=\sum_{k=0}^{\infty}\left[\frac{a_k}{b_k}\left(\sum_{j=0}^{k}\frac{c_j}{d_j}\right)\frac{\prod_{j=0}^{k}p_k}{\prod_{j=0}^{k}q_k}\right]$$

これを使えば、次の第 2 種 0 次の変形ベッセル関数 $K_0(x)$ も高速に計算できる。

$$\begin{aligned}K_0(x) &= -\left\{\log\left(\tfrac{x}{2}\right)+\gamma\right\}I_0(x)\\ &+ \frac{\frac{x^2}{4}}{(1!)^2}+\left(1+\tfrac{1}{2}\right)\frac{\left(\frac{x^2}{4}\right)^2}{(2!)^2}+\left(1+\tfrac{1}{2}+\tfrac{1}{3}\right)\frac{\left(\frac{x^2}{4}\right)^3}{(3!)^2}+\cdots\end{aligned}$$

この論文によると BSA 法は、1976 年に R. P. Brent、1987 年に Borwein 兄弟によって公表され、Chudnovsky 兄弟による円周率 10 億桁の計算が行われた 1989 年当時でもよく知られた方法であったことがわかる。BSA 法は、2 進数を 10 進数などに変換する基数変換の高速計算法も与える。後等や平山は、級数の他に、次のような連分数の高速計算法を与えている。

$$\frac{P_n}{Q_n}=b_0+\cfrac{a_1}{b_1+\cfrac{a_2}{b_2+\cfrac{a_3}{\ddots+\cfrac{a_n}{b_n}}}}$$

この計算法は計算する桁数を p として、十分大きな数である場合、 $O(p(\log p)^3)$ の計算量となる計算法である。本論文では、BSA 法を使って初等関数の計算を行い、その性質を調べた。

以下の計算には、コンパイラとして、Visual Studio 2015 C++、計算機としてショップブランド・コンピュータ (Intel Core i7 7700K 4.2GHz) を利用した。高精度プログラムとして、平山 [3] のプログラムを改良したものを使用した。

2. Binary Splitting Algorithm

BSA 法は、分割統治法の一種で、大きな問題を小さな問題に分割し高速化をはかるものである。本方式は、分割した問題をトーナメント形式に計算することで高速化をはかるものである。

高精度数の乗算については高速フーリエ変換 (FFT）を使う高速な計算方法がよく知られている。FFT は 10000 桁以上の数値の乗算において有効である。BSA 法は 10000 桁以下の数値の計算でも高速に計算できる上、原理が単純なので、様々な計算において容易に利用することが出来る。

2.1 階乗の計算

簡単な例として、階乗の計算を例に挙げて説明する。

$$n!=\prod_{k=1}^{n}k=1\cdot 2\cdot 3\cdots n \tag{1}$$

n が小さい場合、前から順番に計算しても計算時間に大きな違いが生じることはないが、n が大きい場合、前から順序よく計算する方法は効率的ではない。

$n=2^m$ の場合、2 個づつ組み合わせ

$$n!=(1\cdot 2)\cdot(3\cdot 4)\cdots((n-1)\cdot n) \tag{2}$$

さらに、まとめたものを 2 個組み合わせ

$$n!=\{(1\cdot 2)\cdot(3\cdot 4)\}\cdots\{((n-3)\cdot(n-2))\cdot((n-1)\cdot n)\} \tag{3}$$

というように、次々と計算していく。

このように計算することで、通常の計算方法よりも計算桁数が小さいままで計算を行うことが出来るので、計算速度の向上が期待できる。小さい数から順序よく計算すると、大きな数同士の計算は起こらないため、FFT を利用した効率的な計算法を適用できない。しかしながらこのように分割しながら計算すると、大きな数同士の積を行わなければならない。ここで FFT を利用すれば、一層効率的な計算法となる。分割して計算するため、並列計算も容易にできる。この方法は $n=2^m$ である場合に限らず、一般の n の場合においても、効率は少し落ちるが同様のことがいえる。

2.2 計算量

簡単化のため、すべての数値は K 桁とし、K 桁の数の乗算の計算量を $M(K)$ とする。最初の段階において、計算量は、$\frac{n}{2}$ 回の K 桁の乗算を必要とするので $C\frac{n}{2}M(K)$(C: 定数) となる。

次の段階において項数は最初の段階の $\frac{1}{2}$ となり $\frac{n}{4}$、乗算は各項についての計算量は $C\frac{n}{4}M(2K)$ となる。

以降、計算桁数は倍、計算項数は半分となるので総計算量 TM は、次の式で与えられる。

$$TM=Cn\{\frac{1}{2}M(K)+\frac{1}{4}M(2K)+\cdots+\frac{1}{n}M(\frac{n}{2}K)\} \tag{4}$$

n が大きいとき、FFT を利用すると $M(n)=O(n(\log n)^2)$ であることから、

$$TM\approx Cn(\log n)\frac{1}{n}M(\frac{n}{2}K)\approx Cn(\log n)^3 \tag{5}$$

となる。K 桁の数値が複素数や行列等でも同様に評価できる。

3. 級数の行列の乗算化

次のような、級数の計算を考える。

$$S = \frac{1}{C_0}(A_0 + \frac{B_0}{C_1}(A_1 + \frac{B_1}{C_2}(A_2 + \frac{B_2}{C_3}(A_3 + \frac{B_3}{C_4}(A_4 + \cdots \tag{6}$$

この級数を第 n 項まで計算するために、次の式を計算する。

$$\begin{pmatrix} Q_n & P_n \\ 0 & R_n \end{pmatrix} = \begin{pmatrix} B_0 & A_0 \\ 0 & C_0 \end{pmatrix} \begin{pmatrix} B_1 & A_1 \\ 0 & C_1 \end{pmatrix} \cdots \begin{pmatrix} B_n & A_n \\ 0 & C_n \end{pmatrix}$$

第 n 項までの和を S_n とすると、$S_n = \frac{P_n}{Q_n}$ であることがわかる。このことは、(6) の途中式を次のように変形出来ることからわかる。行列 $M_n = \begin{pmatrix} B_n & A_n \\ 0 & C_n \end{pmatrix}$ と定義し、さらに $M_{m,n} = \prod_{k=m}^{n} M_k$ と定義する。ここで、次のような s 個の行列積を考える。途中の n 番目の行列と $n+1$ 番目の行列に注目する。

$$M_{0,s} = \prod_{k=0}^{s} M_k \cdot (M_n \cdot M_{n+1}) \cdot \prod_{k=m+2}^{s} M_k \tag{7}$$

(7) の部分を行列として計算すると、次のようになる。

$$\begin{pmatrix} B_n & A_n \\ 0 & C_n \end{pmatrix} \begin{pmatrix} B_{n+1} & A_{n+1} \\ 0 & C_{n+1} \end{pmatrix} = \begin{pmatrix} B_nB_{n+1} & A_nC_{n+1} + B_nA_{n+1} \\ 0 & C_nC_{n+1} \end{pmatrix}$$

これを元の級数で見ると、以下のようになり、行列の表現が正しいことがわかる。

$$\cdots + \frac{B_{n-1}}{C_n}\left(A_n + \frac{B_n}{C_{n+1}}\left(A_{n+1} + \frac{B_{n+1}}{C_{n+2}}(A_{n+2} + \cdots\right.\right.$$

途中を展開すると

$$\cdots + \frac{B_{n-1}}{C_nC_{n+1}}\left(A_nC_{n+1} + B_nA_{n+1} + \frac{B_nB_{n+1}}{C_{n+2}}(A_{n+2} + \cdots\right.$$

n は任意であるから、行列の積の表現と同等であることがわかる。

4. 関数の計算

多くの関数は、級数に展開できるので、次のように行列の積を計算して、級数の和を計算できる。

4.1 指数関数 e^x の計算

指数関数 e^x は次のように Taylor 展開出来る。

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

これから、関連する行列積表現が得られる。

$$e^x = \lim_{m\to\infty} \frac{P_m}{R_m}$$

$$\begin{pmatrix} Q_m & P_m \\ 0 & R_m \end{pmatrix} = \begin{pmatrix} x & 1 \\ 0 & 1 \end{pmatrix} \prod_{k=1}^{m} \begin{pmatrix} x & 1 \\ 0 & k \end{pmatrix}$$

4.2 関数 $e^x - 1$ の計算

指数関数の引数 x の絶対値が非常に小さいとき、指数関数の代わりに、$e^x - 1$ が使われる。一部のプログラム言語では、exp1(x) とか expm1(x) のような名前で用意されていることがある関数である。引数 x の絶対値が小さいときに利用する関数である。高精度計算するときも、これらの関数を使えば、高精度で計算できる。

$$\frac{e^x - 1}{x} = \sum_{k=0} \frac{x^k}{(k+1)!}$$

の Taylor 展開式から、関連する行列表現が得られる。

$$\begin{pmatrix} Q_m & P_m \\ 0 & R_m \end{pmatrix} = \prod_{k=1}^{m} \begin{pmatrix} x & 1 \\ 0 & k \end{pmatrix}$$

4.3 関数 $\log(1+x)$ の計算

対数関数は $x = 0$ に特異点を持つため、Taylor 展開することができない。このため、$x = 1$ で Taylor 展開する。

$$\frac{\log(1+x)}{x} = \sum_{k=0}^{\infty} \frac{(-x)^k}{k+1}$$

の展開式から、前の式とほぼ同様な関連する行列表現が得られる。

$$\begin{pmatrix} Q_m & P_m \\ 0 & R_m \end{pmatrix} = \prod_{k=1}^{m} \begin{pmatrix} -kx & 1 \\ 0 & k \end{pmatrix}$$

4.4 関数 $\log\left(\frac{1+x}{1-x}\right)$ の計算

$$\frac{1}{2x}\log\left(\frac{1+x}{1-x}\right) = \sum_{k=0}^{m} \frac{x^{2k}}{2k+1}$$

の展開式から

$$\begin{pmatrix} Q_m & P_m \\ 0 & R_m \end{pmatrix} = \begin{pmatrix} (2k-1)x^2 & 1 \\ 0 & (2k-1) \end{pmatrix}$$

Taylor 級数で規則的な表現出来る関数は、このように規則的な行列の積から計算できる。上に示した関数の他に、三角関数、逆三角関数、双曲線関数など多くの関数がこのように行列の積を通して計算できる。

5. 連分数の乗算化

次のような、連分数の計算を考える。

$$\frac{P_n}{Q_n} = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{\ddots + \cfrac{a_n}{b_n}}}} = b_0 + \frac{a_1}{b_1} + \frac{a_2}{b_2} + \cdots + \frac{a_n}{b_n}$$

この連分数に対して、よく知られているように以下の漸化式が成り立つ。

$$\begin{aligned} P_n &= b_n P_{n-1} + a_n P_{n-2} \\ Q_n &= b_n Q_{n-1} + a_n Q_{n-2} \end{aligned}$$

ただし、$P_0 = b_0,\ Q_0 = 1,\ P_{-1} = 1,\ Q_{-1} = 0$ とする。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} P_{n-1} & P_{n-2} \\ Q_{n-1} & Q_{n-2} \end{pmatrix} \begin{pmatrix} b_n & 1 \\ a_n & 0 \end{pmatrix}$$

この関係式から

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} b_1 & 1 \\ a_1 & 0 \end{pmatrix} \begin{pmatrix} b_2 & 1 \\ a_2 & 0 \end{pmatrix} \cdots \begin{pmatrix} b_n & 1 \\ a_n & 0 \end{pmatrix}$$

このように式を展開することによって、次のように書くことも出来る。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 & 1 \\ 1 & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} b_k & 1 \\ a_k & 0 \end{pmatrix}$$

6. 連分数展開された関数の計算

以下の行列の積で表示された結果から関数の値は P_n/Q_n を計算することによって得られる。

6.1 指数関数 e^x の計算

6.1.1 公式 1

$$e^z = 1 + \cfrac{z}{1 + K_{k=1}^{\infty} \cfrac{-kz}{z+k+1}}$$

これから、次の行列表示ができる。

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ z & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} z+k+1 & 1 \\ -kz & 0 \end{pmatrix}$$

6.1.2 公式 2

$$e^z = 1 + \cfrac{z}{1 - z + K_{k=1}^{\infty} \cfrac{kz}{k-z+1}}$$

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1-z & 1 \\ z & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} k-z+1 & 1 \\ kz & 0 \end{pmatrix}$$

6.2 $\log x$

6.2.1 公式 1

$$\log(1+x) = \cfrac{z}{1 + K_{k=1}^{\infty} \cfrac{k^2}{k+1-kz}}$$

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ z & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} k+1-kz & 1 \\ k^2 z & 0 \end{pmatrix}$$

6.2.2 公式 2

$$\log\left(\frac{1+z}{1-z}\right) = \cfrac{2z}{1 + K_{k=1}^{\infty} \frac{-k^2 z^2}{2k+1}}$$

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 2z & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} 2k+1 & 1 \\ -k^2 z^2 & 0 \end{pmatrix}$$

6.3 関数 $\tan^{-1} x$ の計算

$$\tan^{-1} z = \cfrac{z}{1 + K_{k=1}^{\infty} \cfrac{k^2 z^2}{2k+1}}$$

$$\begin{pmatrix} P_n & P_{n-1} \\ Q_n & Q_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ z & 0 \end{pmatrix} \prod_{k=1}^{n} \begin{pmatrix} 2k+1 & 1 \\ k^2 z^2 & 0 \end{pmatrix}$$

7. 数値例

以下の計算では、10 進数で約 5000 桁以上の数値の乗算に FFT を使用するようになっている多倍長計算プログラム（MPPACK)[3] を使用して計算した。

7.1 簡単な数値例

n までの階乗 $n!$ の計算を考える。これを単純に、1 から順序よく計算する通常計算法と BSA 法による計算法によって計算し、経過時間を測定した。その結果を表 1 に示す。計算は 10 進数で 100000 桁で計算した。$4000! = 1.8288 \times 10^{12673}$ であるから、12674 桁の数値で十分な計算精度である。

この結果から分かるように、300! の計算から BSA 法が速くなっている。1000! あたりから、FFT による乗算の高速化が出ていると思われる。通常の計算法では、常に最大で 4 桁程度の数値を掛けているため、FFT による乗算を行う場面が生じないため、計算時間の増加が大きくなっている。

7.2 指数関数 e^x 計算

ここでは、$e^{\pi} = 23.1406926\cdots$ を計算する。$\pi = 3.14159265\cdots$ の 1 桁すなわち e^3 を計算する。これをいろいろな精度で計算した。その結果を以下の表 2 に示す。

π の数値を 9 桁使ったときすなわち $\pi = 3.14159265$ としたときの計算結果を以下の表 3 に示す。

π の数値を 41 桁使ったときすなわち $\pi = 3.14159265358979323\cdots$ としたときの計算結果を以下の表 4 に示す。

指数関数の計算法は多数ある。ここで言う通常計算法とは、使用した多倍長計算プログラム（MPPACK) に組み込まれた計算法である。次の方法で計算したものである。

表 1 Computing Time of $n!$ (msec)

n	通常計算法	BSA 法	速度向上率
100	0.015	0.024	0.650
200	0.043	0.048	0.893
300	0.103	0.074	1.400
400	0.189	0.102	1.860
500	0.309	0.129	2.391
600	0.460	0.163	2.827
700	0.643	0.194	3.315
800	0.860	0.228	3.765
900	1.111	0.259	4.290
1000	1.398	0.292	4.792
2000	6.275	0.648	9.676
3000	14.183	1.120	12.664
4000	26.126	1.632	16.007

表 2 Computing Time of e^3 (msec)

計算精度	通常計算	BSA 法	通常/BSA
128	0.03461	0.06430	0.53829
512	0.21180	0.17470	1.21237
1024	0.65871	0.33585	1.96129
5120	18.39282	2.27632	8.08008
10240	70.72436	4.83528	14.62673
51200	728.94216	33.65989	21.65611
102400	2238.97991	76.79892	29.15380
204800	6817.69679	169.65524	40.18560

表 3 Computing Time of $e^{3.14159265}$ (msec)

計算精度	通常計算	BSA 法	通常/BSA
128	0.03400	0.12965	0.26221
512	0.21195	0.44357	0.47783
1024	0.65268	0.87370	0.74703
5120	18.07117	6.00592	3.00889
10240	69.86567	13.38563	5.21945
51200	723.98416	83.20254	8.70147
102400	2237.19166	188.50146	11.86830
204800	6778.46393	417.92320	16.21940

表 4 Computing Time of $e^{3.141592653589793238\cdots}$ (msec)

計算精度	通常計算	BSA 法	通常/BSA
128	0.03385	0.19073	0.17748
512	0.22279	0.85437	0.26077
1024	0.67070	1.90091	0.35283
5120	18.56865	16.00578	1.16012
10240	71.22502	39.02078	1.82531
51200	730.04855	240.74945	3.03240
102400	2236.41045	530.06423	4.21913
204800	6829.09573	1167.02072	5.85173

$$e^x = \left(e^{\frac{x}{2^n}}\right)^{2^n} \tag{8}$$

この式は、$t = \frac{x}{2^n}$ と置くと、次の二つの式に分かれる。

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots \tag{9}$$

と

$$e^x = \left(e^t\right)^{2^n} \tag{10}$$

に分割できる。(9) と (10) の掛け算の総量が小さくなるように n を選ぶ。このとき、n は計算を 10 進数 p としたとき $n = 2.35\sqrt{p}$ とする。ただし、2.25 の数値は数値実験によって決定したものである。使用する計算機によって変化すると思われる。

この結果を見ると、入力の数値が 1 桁であっても、この表では示しいないが 400 桁よりも精度が低い計算の場合、通常の計算法が速いことがわかる。

入力数値が 40 桁程度ならば、計算が 5000 桁以上の精度ならば、BSA 法は効率的であることがわかる。

7.3 逆正接関数 $\tan^{-1} x$ の計算

この計算例として、円周率の計算を行う。

$$\pi = 176\tan^{-1}\frac{1}{57} + 28\tan^{-1}\frac{1}{239} - 48\tan^{-1} + 96\tan^{-1}\frac{1}{12943}$$

これを BSA 法で計算する。比較のために、相加相乗平均法（AGM 法)[1] を使った計算法で計算した時間も表 5 に示す。

表 5 Computing Time of π (sec)

計算精度	BSA 法	AGM 法
1024000	8.687	6.055
512000	3.766	2.698
256000	1.631	1.161
128000	0.693	0.540
64000	0.283	0.235

ここでの AGM 法は、MPPACK に組み込まれたもの定数関数を使用している。改良すれば円周率の 100 万桁の計算は 3.7 秒程度で可能である。

この結果を見ると、円周率の計算では、AGM 法が若干有利である。このような場合、BSA 法は並列計算が容易であるから、並列計算すれば、容易に AGM 法よりも高速に計算できるもとの思われる。

8. まとめ

BSA 法は、分数などを利用して、出来るだけ小さな数値で計算することによって高速化をはかる計算法である。小さな数値とは、相対的なもので、$O(1)$ の数値であれば、最終的計算精度の 100 分の 1 程度の精度を持つものであればかなり効率的に計算できる。

使用した多倍長計算プログラムで並列計算（OpenMP) ではうまく動作しなことがわかったので、今回は行わなかったが、計算式から容易に分かるように、並列化も容易である。

多くの計算機がマルチコアになった現在、並列化に計算できるように、アルゴリズムを改良して行くのがこれからの課題と思われる。

参考文献

[1] R.P.Brent,A Fortran Multiple-Precision Arithmetic Package, *ACM* Trans. Math. Soft., 4, 1978,57-70
[2] Haible B., Papanikolaou T., Fast multiprecision evaluation of series of rational numbers, Technical Report No. TI-7/97, Darmstadt University of Technology, 1997, http://www.informatik.tu-darmstadt.de/TI/Mitarbeiter/papanik/
[3] 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, Vol. 5, No.3, (1995)123-134
[4] 平山 弘, "連分数の多倍長精度高速計算法",/情報処理学会論文誌,41(SIG 8),(2000)85–91
[5] 右田剛史、天野晃、浅田尚紀、藤野清次,"級数の集約による多倍長数の計算法とπの計算への応用", 情報処理学会研究報告,vol.98, No. 74, (1998)31–36
[6] 右田剛史、天野晃、浅田尚紀、藤野清次, 級数の再帰的集約による多倍長数の計算法とπの計算への応用、情報処理学会論文誌、vol.40, No. 12, (1999)4193-4200
[7] 後保範、金田 康正、高橋 大介, 級数に基づく多数桁計算の演算量削減を実現する分割有理数化法, 情報処理学会論文誌 ,Vol.41, No.6, (2000)1811–1819