Vol.2016-HPC-154 No.8
2016/4/25

有理算術演算における最大公約数計算について

寒川　光[a)]

概要：現在開発中の「有理数計算プログラミング環境」では，桁数の多い分母・分子を動的に拡張可能な配列に格納し，有理数の四則演算のたびに結果の分母・分子の最大公約数（GCD）を求めて割り，約分している．行列計算のような，浮動小数点演算ではお馴染みの数値計算アルゴリズムでは，演算のたびに GCD を求めて約分しないと，桁数が爆発して計算が進まなくなる．しかし反対に，正則連分数展開のように，計算途中に現れる分数がすべて既約で，GCD 計算が不要なアルゴリズムもある．GCD 計算は，除算を用いるユークリッドの互除法ではなく，2 進 GCD アルゴリズムを使用したほうが，一般的に高速である．「有理数計算プログラミング環境」は，浮動小数点計算では解明できない問題を解析することを主たる目的としているが，2 進浮動小数点数を有理数変換すると分母は 2 のべき乗になる．加減算と乗算だけの計算では，分母の 2 べきは保存される．2 進 GCD は，分母が 2 のべき乗の有理数の GCD 計算では，除算を用いるものよりもかなり高速である．本稿ではこのような GCD 計算にまつわる話題を集めて，高速化の観点から報告する．

キーワード：有理算術演算，最大公約数，ユークリッドの互除法，分母 2 べき，LDL 分解，行列式，連分数，特性多項式

Greatest Common Divisor computation in rational arithmetic

Hikaru Samukawa[a)]

Abstract: In our currently developing “rational arithmetic programming environment”, large numerators and denominators of rational numbers are held in dynamically expandable array. The rational numbers are divided by greatest common divisors (GCD) of their numerator and denominator. In matrix computation algorithms which are popular in floating-point arithmetic, the GCD calculation is needed after every four basic arithmetic operation, otherwise an explosion of the number of digits prevents us to continue computation. On the contrast, however, there are algorithms which do not need GCD calculation such as continued fraction expansion, in which all fraction numbers are already reduced. For the GCD calculation, a binary algorithm is, in general, faster than Euclidian algorithm implemented with division operation. An aim of our programming environment is to analyze the problems occuerred in numerical simulation programs caused by rounding errors of the floating-point arithmetic. A binary floating-point number is converted exactly to rational number consisting of denominator with power of two. As far as the computations are performed with additions, subtractions, and multiplications, the denominators’ power two is kept. The binary GCD algorithm is much faster than the GCD algorithm with division operations. In this paper, we report the numerical and performance behavier related to GCD calculations.

Keywords: rational arithmetic, greatest common divisor (GCD), Euclidian algorithm, denominators’ power two, LDL factorization, determinant, continued fraction, characteristic polynomial

1. はじめに

数値計算の多くは浮動小数点演算によって行われており，計算の効率は高く，融通がきくが，信頼性は低い．演算の高精度化には，多重精度算術演算（multiple precision arithmetic）があるが，これは浮動小数点演算の高精度化である．正確な計算は，有理算術演算やモジュラー算術演算によって実現される．身近に有理算術演算を体験できる十進 BASIC の有理数モードが存在する[*1]．我々は，浮動小数点演算を用いる数値計算のプログラミング環境に，有理算術演算を加えることで，浮動小数点計算で生じた丸め誤差に起因する問題を解決する目的で使用できる「有理数計算プログラミング環境」を開発中である．有理数の分母・分子を約 30 万桁まで動的に拡張可能な多桁の自然数を格納できる配列で保持し，有理数の四則演算ごとに分母・分子を既約な分数に約分する．演算結果は丸めないので計算誤差は介入せず，桁溢れしないかぎり，計算は正確である．変数型 `rational` を C++ によるプログラミング環境に追加するだけで，有理数型の変数と浮動小数点型の変数は共存できる．

この環境は次の使用を目的としている [2].

- 丸め誤差の理論を学ぶ前の学生を対象とした数学とプログラミングの教育
- 数値シミュレーションで用いられる，浮動小数点演算による数値計算プログラムで発生した精度に関係する問題分析
- ベンチマークの検収など，計算機システムの稼働確認

本稿では，次章でプログラミング環境の概要と GCD 計算について述べる．3 章で GCD 計算の役割を，対照的な 2 例，既約な分数にしなければ桁数の爆発で計算不能になるアルゴリズムと，GCD 計算を行わないか，間引きして行う連分数計算について述べる．4 章で，無理数の解を，2 つの有理数によって上下限を抑える区間に挟み込んでから縮小反復に持ち込むアルゴリズムで使用する丸め方法について述べる．5 章でまとめる．

[1] 早稲田大学理工学術院
Waseda University, Shinjuku-ku, Tokyo 169–8555, Japan
[a)] samukawa@sic.shibaura-it.ac.jp

[*1] 十進 BASIC は，数学教育を目的に，文教大学の白石教授が開発されたプログラミング言語であるが，これによって有理数計算を体験することができる [1]．十進 BASIC は 5 つの計算モードを持つ．十進 15 桁，十進 1000 桁，2 進数（Intel x86 倍精度浮動小数点），2 進による複素数，有理数である．BASIC の言語規格により，数値は単一の表現に統一されるので，浮動小数点数と有理数を同時に使用することはできない．

2. 有理算術演算プログラミング環境

有理算術演算は計算機科学の黎明期からの研究テーマである [3].「有理数計算プログラミング環境」では，有理数を，分母と分子を多桁の自然数で保持し，これに符号を付加することで表現する．四則演算は次のように行う．

$$r_1 \pm r_2 = \frac{b}{a} \pm \frac{d}{c} = \frac{bc \pm ad}{ac} \tag{1}$$

$$r_1 \times r_2 = \frac{b}{a} \times \frac{d}{c} = \frac{bd}{ac} \tag{2}$$

$$r_1 \div r_2 = \frac{b}{a} \div \frac{d}{c} = \frac{bc}{ad} \tag{3}$$

r_1, r_2 は有理数，a, b, c, d は多桁の自然数である．

n 桁と n 桁の数の積は $2n$ 桁になるので，式 (1) の ac，bc，ad などの数の桁数は演算のたびに増える．そこで有理数を構成するときに，分母，分子の最大公約数（Greatest Common Divisor, GCD）で割り既約な分数とする．

2.1 longint クラスと GCD アルゴリズム

「有理数計算プログラミング環境」では，上記の a, b, c, d などの多桁の自然数を **longint** 型で扱う．

$$z = \sum_{i=0}^{l-1} d_i r^i \tag{4}$$

ここに d_i は各桁の数である．多桁の自然数 z を，基数を $r = 2^{32}$ として 32 ビット符号なし整数型の配列に格納し，桁数 l を整数型で保持する．零は桁数が零（$l = 0$）とする．配列長は 64 から始め，不足すると動的に倍，倍と拡張する可変長とした．最大長は 32,768 としている（10 進数で約 31 万桁）[*2]．多桁数の階層（longint クラス）では，式 (1)(2)(3) の左辺の演算の本体や，GCD 関数をもつ．

2 進 GCD アルゴリズムを示す [3], p. 317．2 数 a と b の共通因子のうち偶数のもの 2^k をシフト演算で先に分離しておき，分離された 2 数の奇数の最大公約数 c を，減算とシフト演算で求め，$\gcd(a, b) = 2^k \cdot c$ を得る．

2 進 gcd アルゴリズム
$k = 0$; $u = a$; $v = b$
while(both(u and v)are even){

[*2] $\log_{10} 4294967296 = 9.633\cdots$ である．計算可能な最大の数は，リコンパイルで変更可能である．

$k = k + 1$
$u = u \div 2; \quad v = v \div 2$
}
if(u is odd) $t = -v$ else $t = u$
while (t != 0) {
while(t is even) {
$t = t \div 2$
if ($t > 0$) $u = t$ else $v = -t$
}
$t = u - v$
}
$\gcd(a, b) = 2^k \cdot u$

前半のループは，2 数 u と v がともに偶数の間，両者を 2 で割る操作を繰返し，少なくとも一方が奇数になるようにする．これは，両者が偶数の場合は $\gcd(u, v) = 2 \cdot \gcd\left(\frac{u}{2}, \frac{v}{2}\right)$ に基づいている．このループを抜けた時点で，u と v のうち少なくとも一方は奇数である．u が奇数なら $-v$ を，偶数なら u を t に格納する．

後半のループは，$\gcd(u, v) = \gcd(u - v, v)$ に基づき，減算で 2 数を小さな数に置き換え，最大公約数の奇数因子を求める．$t = u - v$ として，$\max(u, v)$ を，$t > 0$ なら $u = t$ で，$t < 0$ なら $v = -t$ で置き換えて，u と v のうち大きいほうが $|t|$ に入るようにする．ここで t は偶数なら 2 で割るが，これは $\gcd(u, v) = \gcd\left(\frac{u}{2}, v\right)$ に基づいている．アルゴリズムは $|u - v| < \max(u, v)$ なので停止し，$\gcd(a, b) = 2^k \cdot u$ が得られる[*3]．

2 進 GCD アルゴリズムを用いると，演算量が桁数の 1 次オーダーである減算を主にできるので，除算を用いる方法よりも速い場合が多い．アルゴリズムは教科書的に `while` ループで記述したが，2 で割る操作の繰り返しは，下位の零ビットを数えてシフト演算で行う．したがって一方が 2 のべき乗である 2 数に対して用いると，除算による方法よりも圧倒的に速い．

「有理数計算プログラミング環境」では，関数へのポインタを切り替えることで，除算による GCD アルゴリズム `lgcd`，2 進 GCD アルゴリズム `lgcd2`，常に 1 を返す関数 `lgcd1` を，適宜切り替えられるようにした．`lgcd1` 関数は GCD 計算の省略である．

2.2 rational クラスと 2 進浮動小数点数の有理数表現

有理数階層（rational クラス）はその上に構築される．有理数は 2 つの `longint` 型の数と符号で保持される．式 (1)(2)(3) の有理数演算のエントリルーチン，有理数演算を有理数型の変数を用いて `z=x+y` のように記述するための演算子多重定義のほか，`abs`，`min`，`max`，`floor`，`ceil` などの関数や，浮動小数点数を有理数に変換する関数 `Rdset`，多項式の除算などのユーティリティルーチンを提供する．

倍精度浮動小数点数の有理数表現を考える．浮動小数点数は，有限のビット数で表現される数なので，数学的には有理数である．IEEE 754 形式の倍精度浮動小数点数 a を，e を指数部，d_i を 0 または 1 として 2 進数で次のように表す．

$$a = \pm\left(\sum_{i=0}^{52} d_i 2^{-i}\right) \cdot 2^e = \pm A \cdot 2^e \qquad (5)$$

A を構成する d_i，$i = 0, \cdots, 52$ を有意桁（significand）という．$B = A \cdot 2^{52}$ と置き換えると，B は 2^{54} よりも小さいので，10 進数では 17 桁以下で表わせる．B を用いると式 (5) の数は，$a = \pm B \cdot 2^{e-52}$ である．$C = 52 - e$ とおくと $a = \pm B \div 2^C$ である．B の最下位ビットが 1 であれば，分子は B，分母は 2^C と有理数表現される．B の下位 k ビットが 0 であれば，分子は $B \cdot 2^{-k}$，分母は 2^{C-k} と有理数表現される．倍精度浮動小数点数は，正確に有理数変換されるが，その分母は 2 のべき乗である[*4]．

ここでは倍精度の浮動小数点数を例に述べたが，単精度浮動小数点数や，4 倍精度数，多倍精度数など，有意桁が 2 進数の浮動小数点数ならこの変換は同じ形で成立する．したがって 2 進の有意桁をもつ浮動小数点数が有理数表現されると，分母は 2 のべき乗になる．また，アルゴリズムが加減算 (1) と乗算 (2) だけで行われる場合，「分母 2 べき」は継続する．

2.3 interval クラス

有理数を四則演算で扱うアルゴリズムは，有理数の分母または分子が longint 型の数の最大桁数を超えないかぎり，有理算術演算により正確に計算できる．しかし無理数を扱う数式に対しては，正確な計算は実現されない．例えば，有理数係数の多項式の零点を求める非線形方程式の求解では，正確な無理数解は得られない．このようなアルゴリズムに対しては，

[*3] 数値例として $a = 128 = 2^7$，$b = 20$ を示す．前半のループで 2^2 で割られて，$u = 32 = 2^5$，$v = 5$ となり，後半のループでも 2^5 で割られて $u = 1$ となり $\gcd(u, v) = 1$ を得て，最終的に $\gcd(a, b) = 4$ が得られる．

[*4] 倍精度浮動小数点で “0.4” を例にとると $0.4 \fallingdotseq \frac{3,602,879,701,896,397}{9,007,199,254,740,992}$ で，分母の 4 倍に 2 を加えると分子の 10 倍になる．0.1 は 2 進数では循環小数になるので，その 4 倍の 0.4 も循環小数であり，有限のビット数で有意桁を 2 進表現すると丸められる．また分母は 2^{53} である．

区間算術演算（interval arithmetic）を用いる．2 つの有理数で区間の下限と上限を把握する `interval` 型の変数とそれを扱う関数をサポートする interval クラスを，rational クラスの上に設けた．`interval` 型の変数によって区間を数のように扱える．`interval` 型の変数同士，また `interval` 型と `rational` 型の変数の算術演算は，記号 $+,\ -,\ *$ でプログラミングできる．区間 $x = (a, b)$ を 2 つの有理数 a と b で表すには，`interval x(a,b)` と書くとコンストラクタが変数 `x` を作る．区間 x からその下限を rational 型の変数に取り出すには（メンバー関数により）`x.lower()`，上限を取り出すには `x.upper()` と書く．区間 $x = (a, b)$ の符号を反転すると $(-b, -a)$ になるが，これには `y=x.neg()` と書く．区間 $x = (a, b)$ の逆数 y は `y=x.inv()` と書く．除算は逆数を掛ける．

これらの階層の上に有理数 BLAS（Basic Linear Algebra Subprograms）階層（rblas クラスとその並列版をサポートする prblas クラス）をもち，数値線形代数（NLA: numerical linear algebra）プログラムの並列化を容易にする [4].

3. 数値計算アルゴリズムでの GCD 計算の役割

前章では，四則演算のたびに GCD を求めて約分すると述べたが，これを行わないとどのようになるかを，対称行列の三角分解を例に述べる．一方，GCD 計算が不要な連分数によるアルゴリズムも存在する．本章ではこの対照的な 2 つのアルゴリズムによって GCD 計算の役割を紹介する．

3.1 GCD 計算が必須のアルゴリズム

対称正定値行列 A を係数行列とする連立 1 次方程式を解く問題を例とする．

$$A\boldsymbol{x} = \boldsymbol{b} \tag{6}$$

解法は修正コレスキー分解（modified Cholesky factorization，LDL 分解）

$$A = LDL^T \tag{7}$$

と，対応する代入計算を使用する．ここでは下三角要素を転置して $t_{ij} = l_{ji}$ によって L^T の要素を表す．

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} t_{ki} u_{kj}, \qquad (i \le j) \tag{8}$$

$$t_{ij} = \frac{u_{ij}}{u_{ii}} \qquad (i < j) \tag{9}$$

プログラミング例を示す．$n \times n$ の対称行列 A と次数 n を入力に渡すと，対角行列 D の対角項と上三角行列 L^T の対角項以外の項を，行列 A の要素の渡された領域に上書きして返す関数 `LDL` を示す．

```
void LDL(rat::matrix<rational>& a, const int n){
    rational s,t;
    int i,j,k;
    for (j=1; j < n; ++j) {
        for (i=1; i < j; ++i) {
            s=0;
            for (k=0; k < i; ++k) {
                s = s + a[k][i] * a[k][j];
            }
            a[i][j] = a[i][j] - s;
        }
        s=0;
        for (k=0; k <= j-1; ++k) {
            t = a[k][j] / a[k][k];
            s = s + t * a[k][j];
            a[k][j] = t;
        }
        a[j][j] = a[j][j] - s;
    }
}
```

数値計算の教科書の LDL 分解と同じループ構成を採用したので，表面的には浮動小数点演算による LDL 分解と似ている [5]．しかしデータ型は `double` 型ではなく **rational** 型なので，計算機の仕事は大きく異なる．引数 `a` の行列 A の要素と，2 つの変数 `s` と `t` が rational 型である．“`a[i][j]=a[i][j]-s`” などの有理算術演算は，動的にメモリを拡張しつつ，正確に行われる（`double` 型では有効ビット数 53 に丸められる）．

桁数の増加は，データ依存性が強い．係数行列を，フランク行列 $a_{ij} = n - \max(i, j) + 1$，ヒルベルト行列 $a_{ij} = \dfrac{1}{i + j - 1}$，浮動小数点演算でヒルベルト行列を作成して有理数変換した行列，その全行列要素の分母の最小公倍数（least common multiple，LCM）を求めて掛けた整数行列，分子を 2^{31} よりも小さい乱数（乗算合同法），分母を $2^{31} - 1$ とした行列，分子・分母ともに独立した $2^{31} - 1$ 以下の乱数の 6 つの場合の，次数 $n = 40$ での計算時間（秒）の比較を示す．

係数行列	時間（秒）
フランク行列	0.19
ヒルベルト行列	0.20
倍精度ヒルベルト行列	1.29
LCM 倍した倍精度ヒルベルト行列	1.43
分子乱数・分母 $2^{31}-1$	1.43
分子乱数・分母乱数	251.40

計算機はノートパソコンで，1 スレッドのみを使用した．このように，計算時間はデータによって大きく変化する [2]．乱数を使用しても，分母が定数であれば，それを掛けることで整数行列に変換できる．定数が分子と同程度の 10 進数で 10 桁程度であれば，整数行列に変換しても桁数は 20 桁に収まる．しかし分母も乱数を使用すると LCM は巨大な数になる．有理算術演算でも最後の 2 つは本質的な違いが，計算時間に現れる..

これらの計算では，四則演算のたびに既約にしている（分母・分子の GCD を求めて分母・分子を割る）．既約にしないと，有理数を構成する分母・分子の桁数は膨大になる．次数 4 のフランク行列を分解して得られる対角行列 D と上三角行列 L^T を，上三角行列の対角項（$=1$）の上に重ねて D の対角項を記述して示す．

$$\begin{pmatrix} \frac{4}{1} & \frac{3}{4} & \frac{1}{2} & \frac{1}{4} \\ & \frac{3}{4} & \frac{2}{3} & \frac{1}{3} \\ & & \frac{2}{3} & \frac{1}{2} \\ & & & \frac{1}{2} \end{pmatrix} \tag{10}$$

GCD 計算を行わないと，行列要素の桁数は増加し，次のようになるが，$d_4=\dfrac{1}{2}$ である．

$$\begin{pmatrix} \frac{4}{1} & \frac{3}{4} & \frac{2}{4} & \frac{1}{4} \\ & \frac{3}{4} & \frac{8}{12} & \frac{4}{12} \\ & & \frac{128}{192} & \frac{12288}{24576} \\ & & & \frac{452984832}{905969664} \end{pmatrix} \tag{11}$$

フランク行列は行列式の値が 1 になる特殊な行列なので，桁数の増加が例外的に小さいが，ヒルベルト行列の場合は桁数がより大きくなる．ここでは有理数のヒルベルト行列ではなく，倍精度浮動小数点演算で作成したヒルベルト行列を有理数変換した例で示す．次数 4 の行列の場合，d_4 は，GCD を求めて割る方式では $\frac{1}{2800}$ になるが，既約にしないで計算すると $\frac{44030125670400}{123284351877120000}$ になる（分子の 2800 倍が分母）．連立 1 次方程式の解ベクトルが，すべての要素が 1 になる問題を解いた場合，解ベクトルの全要素は「分母＝分子」つまり 1 であるが，その桁数は 200 桁を超える．次数 4 というごく小規模な計算ですらこの桁数であるから，ガウスの消去法のような数値計算アルゴリズムを，GCD 計算を省略して行うことは考えられない（桁数の爆発で計算不能に陥る）．

一方，有理数の桁数が大きく減少する文脈もある．整数行列の行列式は，行列式の定義式が乗算と加減算だけなので，整数である．LDL 分解を経由して $\det(A)=\prod_{i=1}^{n} d_i$ で求めると，d_i は分母・分子ともに桁数の多い数であるが，総積の最終結果である行列式は分母が 1 になり，分子の桁数も中間結果の分子より小さな数になることが多い．

次数 4 の正確なヒルベルト行列を LDL 分解すると $d_1=1$，$d_2=\dfrac{1}{12}$，$d_3=\dfrac{1}{180}$，$d_4=\dfrac{1}{2800}$ となり，$\det(A)=\dfrac{1}{6048000}=0.000000165\cdots$ となる．これに対し，倍精度浮動小数点演算で作成したヒルベルト行列を有理数変換すると，$\dfrac{1}{3}$ が丸められるので，行列式は分子が 58 桁，分母が 65 桁の数になる．これは途中の d_3 や d_4 が桁数の多い数になっているからである．

行列要素の分母の LCM を掛けて整数行列にすると，中間の d_3 や d_4 はやはり桁数が同様の桁数になるが，行列式を求める総積計算では一気に減って，分母は 1 になる[*5]．このように，最終結果が整数（分母が 1）になる問題では，桁数が激減する文脈をもつことがある．この性質は，（浮動小数点演算では利用する機会がないが）有理算術演算ではプログラミングで利用できる．本稿では「結果が整数になることが数学的に保証される性質」を**整数性**と呼ぶことにする．

3.2 GCD 計算が不要のアルゴリズム

3.2.1 正則連分数計算

1 より大きな実数 x の連分数展開は，$x_0=x$ とおき，$i=0,1,2,\cdots$ について「x_i の整数部分を取り出して a_i とし，残りの逆数を求めて x_{i+1} とする」操作を x_{i+1} が整数になるまで繰り返す．

$$x_0=a_0+\frac{1}{x_1}=a_0+\cfrac{1}{a_1+\cfrac{1}{x_2}}=a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{x_3}}}=\cdots$$

場所を節約した $x=[a_0,a_1,a_2,a_3,\cdots,a_{n-1},x_n]$ の書き方も使用される．a_n までとると**第 n 近似分数**と

[*5] 次数 n の行列 A に係数 s を掛けた行列 sA の行列式は，$\det(A)$ の s^n 倍である．

いう．無理数を連分数展開すると無限に続くが，有理数は有限で止まる．これは分子が 1 の形の正則連分数展開が，2 つの自然数の最大公約数を求める GCD アルゴリズムの変形だからである．

高校の『数学 B』には，自然数の除算 “$a \div b = q \cdots r$”，つまり “$a = bq + r$” により $\gcd(a,b) = \gcd(bq+r,b) = \gcd(b,r)$ を繰り返す BASIC プログラムが掲載されていた[*6]．被除数を除数で，除数を剰余で置換えて割るので「互除法」という．“$d_{-2} = a$” と “$d_{-1} = b$” とおき漸化式

$$d_{i-2} = d_{i-1}q + d_i \qquad (12)$$

に書き直す．$i = 0, 1, 2, \cdots$ について $d_i = 0$ になるまで繰り返すと，最後の d_{i-1} が $\gcd(a,b)$ として得られる．**漸化式 (12) の両辺を d_{i-1} で割り，分数 $\frac{d_{i-2}}{d_{i-1}} = x_i$ とおくと連分数展開の漸化式 $x_i = a_i + \frac{1}{x_{i+1}}$ が得られる**（商 q を a_i とした）．

π の 1000 桁の近似値が与えられたとき，その第 n 近似分数まで連分数展開するプログラムを示す．

```
int main(void){
    rational pi,p,r,x,y;
    uint32_t i,n,m,mm;
    std::cout << "Enter n" << std::endl;
    std::cin >> n;
    longint pin("31415926535897932384 略 ");
    longint pid("10000000000000000000 略 ");
    rational pi1000(pin,pid);
    rat::vector<rational> a(n+1);
    x=pi1000;
    p = x.rational::floor();
    a[0] = p;
#ifdef LGCD1
    longint::lgcd_p=longint::lgcd1;
#endif
    m=n;
    for(i=1; i<=n; i++){
        y=x-p;
        if(y==rational::ZERO){m=i-1;break;}
        x = y.inv();
        p = x.rational::floor();
        a[i] = p;
    }
    r=a[m];
    for(i=m-1;i>0;i--)r=a[i]+rational::ONE/r;
    pi=a[0]+rational::ONE/r;
    cout << "pi="<< pi.format(n+10,n)<< endl;
    return 1;
}
```

有理数 x の整数部分を floor 関数で変数 p に得る．前半のループが連分数展開を，rational 型の配列 a[i] に作成する．x-p で有理数型変数 y に小数部分を得て，その逆数を inv 関数で得て[*7]，x_i として，その整数部分が連分数の第 i 項になるので a[i] に格納する．y が零なら，展開は終わって反復を出る．

後半のループは最終項から逆に連分数を分数（有理数）にすることで π の第 n 近似分数を得る．

pi の表示は format 関数に桁数を指定して書くが[*8]，ここでは小数点以下 n 桁とした．pi1000 が 10 進数で 1000 桁の精度であるが，連分数展開で得られたこの数値に対する最良近似分数の分母も分子も 104 桁である[*9]．なお，π の 1000 桁の近似値は有理数なので，連分数展開は有限回（1997 回）の反復で停止する．

正則連分数は GCD アルゴリズムと同等なので，変数 y の有理数は既約である [6]．したがって LGCD1 を指定してコンパイルすると，不必要な GCD 計算を行わないので格段に高速化される．

3.2.2 円周率の計算

円周率の計算の多くは平方根を使用する．ここでは，π を挟む区間の上下限を有理数で求められる方法を例にする．

$\arctan x$ の連分数展開は次式で表される [7], p. 51.

[*6] GCD アルゴリズムは日本語では「ユークリッドの互除法」と呼ばれるが，英語では Euclidian algorithm で，「の互除」は日本か中国の数学者が加えた意訳であろう．ユークリッドは定規とコンパスを用いて幾何学で解いたので，除算はない（除算は位取り記数法が発見された後に現れる）．アルゴリズムの基本は “a が b の倍数のとき $\gcd(a,b) = b$” と “$\gcd(a,b) = \gcd(a-b,b)$” にあって，“$\gcd(a,b) = \gcd(b,r)$” は後者を繰り返すことで得られる．2 進 GCD アルゴリズムは除算を使わない．

[*7] 逆数は 1 を割ってもできるが，GCD 計算は不要なので inv 関数を使ったほうがよい．

[*8] format 関数は指定された小数点以下の桁数まで 10 進小数で表示する．

[*9] 第 n 近似分数 $\frac{p_n}{q_n}$ の誤差は，$\frac{1}{q_n^2}$ より小さく，分母の大きさの割に高精度である．

$$\arctan x = \cfrac{x}{1+\cfrac{x^2}{3+\cfrac{(2x)^2}{5+\cfrac{(3x)^2}{\ddots\cfrac{}{2n-1+\cfrac{(nx)^2}{\ddots}}}}}} \tag{13}$$

この式は，べき級数によって定義されるガウス超幾何関数によって表された arctan を連分数展開して得られる [8]．$x=1$ とおいて 4 倍すると π が得られる．

$$\pi = \cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7+\cfrac{4^2}{9+\ddots}}}}} \tag{14}$$

具体的に第 2 近似分数の計算例を示す．

$$\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5}}} = \cfrac{4}{1+\cfrac{1^2\cdot 5}{3\cdot 5+2^2}} = \frac{4\cdot 19}{19+5} = \frac{76}{24} = 3.16666\cdots$$

第 3 近似分数の計算例を示す．

$$\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7}}}} = \cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2\cdot 7}{35+3^2}}} = \cfrac{4}{1+\cfrac{1^2\cdot 44}{3\cdot 44+28}} = \frac{160}{51} = 3.1372549\cdots$$

第 2 近似分数は π より大きく，第 3 近似分数は π より小さい[*10]．

このアルゴリズムは，式 (13) の規則性から，展開のループは不要で，a_n から逆順のループ計算で第 n 近似分数が得られる．a_n と a_{n-1} による項を有理数型の変数 u と t に作成したら，大小比較して区間型の変数 v の下限と上限として格納する．ループの添字を i とすると，各反復では「これに $(2i+1)$ を加えてから i^2 を割る」．ループ内の z= $\cdots$ と v= $\cdots$ のステートメントは，interval 型と rational 型の算術演算なので，1 ステートメントで上限と下限に対して演算を行う．

```
uint32_t n,m,k;
std::cout << "input n=" << std::endl;
std::cin >> n ;
std::cout << "input k=" << std::endl;
std::cin >> k ;  パラメータ k を与える
rational p,q,t,r,s,u,x,tmp;
longint::lgcd_p = longint::lgcd1;
p=n*n; q=2*n+1;
tmp=2u*(n-1u)+1u; t=p+q*tmp;
tmp=(n-1u)*(n-1u); p=q*tmp;
q=t;            t=p/q;
r=(n-1u)*(n-1u); s=2u*(n-1u)+1u; u=r/s;
interval v;  //  interval 型変数
if( t >= u){ v=interval(u,t);
}else{ v=interval(t,u); }
rational vlow=v.lower();
int lastdgts=getdgts(vlow);
for(uint32_t i=n-2; i>0; i--){
    x=i;
    interval z=v+(rational::TWO*x+rational::ONE);
    interval w=z.inv();
    rational vlow=v.lower();
    if(getdgts(vlow)/lastdgts >= k){
        longint::lgcd_p = longint::lgcd2;
        v=w*(x*x);
        longint::lgcd_p = longint::lgcd1;
        vlow=v.lower(); lastdgts=getdgts(vlow);
    }else{
        v=w*(x*x);
    }
}
```

formatprn 関数によって，$n=2000$ を与えた場合の結果を示す[*11]．

```
3.14159265358979323 (1500 digits) 596023648066508830
3.14159265358979323 (1500 digits) 596023648066556961
```

この計算も，GCD 計算を省略したほうが速い．これは連分数形式を近似分数に戻す計算が，桁数の小さな自然数の演算であり，これに対し GCD 計算がはるかに重い演算だからである．

そこで適当なパラメータ k を与えて，桁数の増加がこのパラメータを越えたときだけ lgcd2 関数で近似値 v を既約にする．getdgts 関数は，与えられた有理数の分母と分子の 2^{32} 進数での桁数の和を返す．π の近似値は interval 型の変数 v に得られる．その下限を有理数型の変数 vlow に取り出し，桁数

[*10] 第 n 近似分数（例えば $\frac{76}{24}$）も，途中に現れる分数（例えば $\frac{28}{44}$ など）も既約でないところが正則連分数と異なる．

[*11] formatprn 関数は，interval 型の変数の上限と下限の差から，10 進数での小数点以下の値の違う位置を調べて，その桁以下数桁までを表示する．このとき同じ数は省略し，省略した桁数を表示する．

を lastdgts に記憶する．反復で桁数は増加していくが，最後に既約にした状態から k 倍になったら，既約にする．実測した範囲では $k=2$ の近傍が適当であるようだ．次表にパソコンでの結果（秒）を示す．

n	$k=2$	$k=3$	$k=4$
10000	2.7	2.8	2.9
20000	9.6	9.6	10.8
30000	20.2	21.2	

4. 有理算術演算での丸め方式

有理算術演算で，非線形方程式 $f(x)=0$ の解を求める問題を考える．解は一般的には無理数なので，正確には得られない．そこで，解が 1 つだけ存在する区間を interval 型の変数に包含（inclusion）し，この区間を縮小反復して，必要な精度まで解を追い込む．縮小反復には「2 分法」や「挟み撃ち法」などを用いる．2 分法は，解の存在する区間 $(a,\ b)$ の中点 $c=\dfrac{a+b}{2}$ で関数値を計算して，$p(a)\cdot p(c)<0$ なら区間 (a,c) を，$p(c)\cdot p(b)<0$ なら区間 (c,b) を選択する．したがって，反復ごとに区間幅を半分にしてゆける．挟み撃ち法は，2 点 $(a,f(a))$ と $(b,f(b))$ を直線で結び，x 切片

$$c=\frac{af(b)-bf(a)}{f(b)-f(a)} \tag{15}$$

を次の反復での新たな a または b として採用する．両者を比較すると，2 分法が圧倒的に高速である場合がある．この理由は，有理数の分母・分子を，多桁数 $a_n,\ a_d$ を用いて $a=\dfrac{a_n}{a_d}$ のように記述すると，$c=\dfrac{a_nb_d+a_db_n}{2a_db_d}$ で，a_d と b_d が 2 のべき乗の場合，c の分母も 2 のべき乗が維持されるからである．しかし挟み撃ち法では $f(b)-f(a)$ による除算があるので，分母が 2 のべき乗ではなくなる．このため，桁数の増大に加えて，GCD 計算に時間を要するようになり，遅くなって使用に耐えない．

区間を縮小反復する場合，両端点は正確である必要はないので，適当に丸めてもよい．丸め方式に自由度があるので，計算の高速性を重んじる．「有理数計算プログラミング環境」では，有理数 $x=\dfrac{x_n}{x_d}$ を，分母が 2 のべき乗の有理数 $y=\dfrac{y_n}{2^k}$ に丸める roundrat(x,m) 関数を用意した．つまり，与えられた精度の 2 進数への丸めである．m は 1024 の倍数を指定する（$k=1024m$）．有理数 $x=\dfrac{x_n}{x_d}$ と分母 k を与えると，

$$y_n=\left(\lfloor\frac{2^kx_n}{x_d}\rfloor,\lceil\frac{2^kx_n}{x_d}\rceil\right) \tag{16}$$

を区間で返す．記号 $\lfloor x\rfloor$ は x を，最も近い小さいほうの 2 進数に丸め，記号 $\lceil x\rceil$ は x を，最も近い大きいほうの 2 進数に丸める．

挟み撃ち法で得られた式 (15) による座標値 c を含む区間を roundrat 関数で得て，区間 (a,b) の中心側の区間端点を d として選ぶ．d と区間端点 a または b への近いほうを幅 $d-a$ または $b-d$ を倍々に探索して符号変化を調べて次の候補とする．

この変形した挟み撃ち法を，行列要素がすべて整数の対称行列の特性多項式の解を，必要なだけ高精度に高める計算に用いた[*12]．特性多項式の係数は，有理算術演算で正確に求めるが，特性方程式の解は浮動小数点演算で近似解を得る（有理数変換すれば分母は 2 のべき乗である）．近似解でも包含が成立すれば，ここで述べた方法で限りなく高精度化することができ，整数性から因数分解の代替となる因子探しができる [9]．測定されたデータを当て嵌めて逆問題を解くように，近似固有値を当て嵌めることで因子探しにより因数分解を代替できる．このときの精度は 200 桁以上に及ぶ．すべて（26 個）の精度改良を 2 分法で行うと，17000 回以上の反復が必要になるが，変形した挟み撃ち法ではこれを 800 回程度に減らすことができ，精度改良の時間を数分の 1 にすることができた．

5. まとめ

本論文では，有理算術演算で数値計算を行う場合に，ホットスポットとなりやすい GCD 計算に焦点をあてていくつかの事例を紹介した．浮動小数点演算で使用される数値計算アルゴリズムを有理算術演

[*12] 2 次元トラス構造解析プログラム CT2D が倍精度演算で生成した行列の固有値問題の多重度解析に使用した [5]．行列の次数は 26 で，拘束条件をもたない正方形状の構造で，理論的には 3 つの零固有値（多重度 3）と 6 組の重根が存在する．有理算術演算による多重度解析では零固有値は零ではなく，丸め誤差に起因する小さな値をもち，これを含めて 7 組の重根が存在することが分かった．しかし CT2D を生成するときのコンパイラの最適化オプションを指定すると，重根は消えた．多重度解析では，行列を有理数変換し，行列要素の分母の LCM を掛けた整数行列に対し，特性多項式を求める．倍精度演算で得られた行列の固有値の近似値を，包含を成立させ，精度改良によって，候補を根と係数の関係公式に代入し，得られる多項式係数の整数性から因子の候補を探すことができる．候補は特性多項式を割り切るかどうかで，因子か否か判定できる．ここで述べる計算は，次数 26 で 26 個の固有値が単根として得られるケースで，特性多項式が $p_{26}(\lambda)=r_1(\lambda)s_{12}(\lambda)t_{13}(\lambda)$ と因数分解されるときの因子 $s_{12}(\lambda)$ と $t_{13}(\lambda)$ を整数性を用いて探すところの計算に用いた．

算で使用すると，GCD 計算がどのように活躍するかを，小さな例で桁数の増大の激しさを紹介した．一方，桁数が激しく減る文脈もあることも紹介した．さらに連分数計算の例で，GCD 計算を省略したほうが速くなる計算もあることを紹介した．

倍精度浮動小数点計算で得られた近似値は，有理数変換すると分母が 2 のべき乗になる．この数を整数性を利用できるまで精度改善する部分では，区間演算で分母 2 べきを保存して GCD 計算の高速性を追求すると効果があることを確かめた．

謝辞　本研究の一部は科学研究費補助金・基盤 (C) 課題番号 25330145「有理数計算ライブラリの並列化と誤差診断ツールの開発」から支援を頂いた．記して謝意を表す．

参考文献

[1] *http://hp.vector.co.jp/authors/VA008683/*.
[2] 寒川 光, 有理数計算による実対称行列の正確な 3 重対角化による固有値の高精度計算, *HPCS2013* **論文集**, pp. 11–22, 2013.
[3] D. Knuth., *The Art of Computer Programming, Volume 2, Third Edition*, Addison-Wesley, 1998, 有澤　誠，和田英一監訳：The Art of Computer Programming, Third Edition , 株式会社アスキー, 2004.
[4] 寒川 光, 有理数線形代数計算における有理数 blas の提案, *HPCS2014* **論文集**, pp. 57–64, 2014.
[5] 寒川 光, 藤野清次, 長嶋利夫, 高橋大介, *HPC* **プログラミング**—*ITText* **シリーズ**, オーム社, 2009.
[6] 寒川 光, **講座　第** *3* **回「有理数計算」**, **シミュレーション**, 34(3):42–50, 2015.
[7] 小林昭七, **円の数学**, 裳華房, 1999.
[8] 中川　仁, **円周率の連分数展開について**, /http://www.juen.ac.jp/math/nakagawa/pi2002.pdf, 2002.
[9] 寒川 光, **講座　第** *4* **回「有理数計算」**, **シミュレーション**, 34(3):46–55, 2015.