有理算術演算における最大公約数計算について

寒川　光, 早稲田大学理工学術院

平成 28 年 7 月 6 日

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．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 2 べき」は保存される．2 進 GCD は,「分母 2 べき」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，有理算術演算プログラムの高速化で GCD 計算が強い影響を与えるアルゴリズムについて述べる．

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 occurred 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.

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プログラミング環境に十進 BASIC の有理数モードがある[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自然数を格納できる配列で保持し，有理数

[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 x87 FPU 命令セットの倍精度浮動小数点数），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．

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].

- プログラミングを用いた数学教育
- 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，2 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，約分しなければ桁数膨張で計算不能になる行列計算と，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，非線形方程式の無理数解を，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縮小反復するときの丸め方式と GCD アルゴリズムの関係について述べる．5 章でまとめる．

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，分母と分子を可変長の配列に格納することで，大きくなっても桁溢れしない自然数（以下，多桁数）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．そこで計算結果は，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．

本章では「有理数計算プログラミング環境」を構築する階層の中で，多桁数演算クラス，有理算術演算クラス，有理数で上下限を抑える区間演算クラスについて述べる．

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，式 (1)(2)(3) の a, b, c, d などの自然数を **longint 型**で扱う．

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

自然数 z を，基数を $r = 2^{32}$，各桁の数 $0 \le d_i < r$ は 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（$z = 0$）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] $\log_{10} 4294967296 = 9.633\cdots$ 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．

2 進 GCD アルゴリズムは，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 アルゴリズムを示す [3, p. 317].

2 進 gcd アルゴリズム
$k = 0;\ u = a;\ v = b$
while(both(u and v)are even){
    $k = k + 1$
    $u = u \div 2;\quad v = v \div 2$
}
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．なお，デフォールトは lgcd2 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` 型の数と符号による **rational 型**の変数によって保持される．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.

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

[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．

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を有理数に変換すると，その分母は 2 のべき乗である**．

0.4 を例にとると

$$0.4 \fallingdotseq \frac{3,602,879,701,896,397}{9,007,199,254,740,992} = \frac{3,602,879,701,896,397}{2^{53}} \tag{6}$$

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．

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，**加減算 (1) と乗算 (2) だけで表される数式の計算では「分母 2 べき」は保存される**．

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桁溢れしないかぎり，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，正確な解は得られない．例えば，有理数係数の多項式の零点を求める問題では，解が有理数なら正確に得られるが，無理数なら正確には得られない．このような問題に対しては，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` 型の変数の加減算と乗算は，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 $x = (a, b)$ の符号を反転した $y = (-b, -a)$ を得るには `y=x.neg()` と書く．区間 $x = (a, b)$ の逆数 z は `z=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．

$$A\boldsymbol{x} = \boldsymbol{b} \tag{7}$$

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

$$A = LDL^T \tag{8}$$

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．

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} t_{ki} u_{kj}, \qquad (i \le j) \tag{9}$$

$$t_{ij} = \frac{u_{ij}}{u_{ii}} \qquad (i < j) \tag{10}$$

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）倍した整数行列，分子を 2^{31} よりも小さい乱数（乗算合同法）で分母を $2^{31}-1$ 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[4]．

係数行列	時間（秒）
フランク行列	0.02
ヒルベルト行列	0.05
倍精度ヒルベルト行列	0.73
LCM 倍した倍精度ヒルベルト行列	0.73
分子乱数・分母 $2^{31}-1$	0.78
分子乱数・分母乱数	168.78

[4]計測に使用した計算機は，Intel の Sandy Bridge マイクロアーキテクチャの Xeon CPU E5-2670 2.6GHz を 2 つ搭載する．CPU は 8 コア（16 スレッド）なので，並列度は 16 ウェイ，32 スレッドであるが，ここでは並列化はしていない．メモリは DDR-3 を 32GB 搭載している．OS は Linux version 2.6.32，コンパイラは GNU gcc 4.4.7 である．

このように，計算時間は行列の数学的性質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．分母の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．このため，分母が定数の行列は，有理数のまま LDL 分解しても，整数行列に変換しても計算時間に大差はない．しかし分母と分子に独立した乱数を使用した行列では LCM は巨大な数になるので，整数行列に変換すると各要素は大きな数になる．このため，有理数のまま LDL 分解したほうが，整数行列に変換して分解するよりも速い．有理算術演算で LDL 分解のような行列計算アルゴリズムの計算時間を予測する場合，行列要素を整数変換して考えるとヒントが得られることがある．

3.1.1 桁数膨張と GCD 計算

これらの計算では，四則演算のたびに約分している（分母と分子の GCD を求めて分母・分子を割る）．約分しないと，有理数を構成する分母・分子の桁数は膨張する．次数 4 のフランク行列を分解すると，次のようになる．

$$\begin{pmatrix} 4 & 3 & 2 & 1 \\ 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{2} & \frac{2}{3} & 1 & \\ \frac{1}{4} & \frac{1}{3} & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 4 & 3 & 2 & 1 \\ & \frac{3}{4} & \frac{1}{2} & \frac{1}{4} \\ & & \frac{2}{3} & \frac{1}{3} \\ & & & \frac{1}{2} \end{pmatrix} \tag{11}$$

上三角行列 L^T の対角項（$=1$）の上に対角行列 D を重ねて記述する（行列要素は式 (10) の行列 T と対角項 $u_{ii} = d_i$ である）．等式で両辺に行列があるが，左辺が GCD を求めて約分した計算，右辺が GCD 計算をスキップして分母・分子の桁数を削減せずに計算した場合を示す．

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

3 列目について見ると，$l_{31} = u_{13} \div u_{11} = \dfrac{2}{4}$ である．$u_{23} = a_{23} - l_{21}u_{12} = 2 - \dfrac{3}{4} \cdot 2 = \dfrac{2}{4}$ になり，$t_{23} = u_{23} \div u_{22} = \dfrac{2}{4} \div \dfrac{3}{4} = \dfrac{8}{12}$ になる．これらをオペランドに用いる対角項は $u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} = 2 - \dfrac{4}{4} - \dfrac{16}{48} = 2 - \dfrac{4 \cdot 48 + 16 \cdot 4}{4 \cdot 48} = 2 - \dfrac{192 + 64}{192} = \dfrac{128}{192}$ と桁数が増える[5]．

4 列目について見ると，約分して計算すれば $u_{44} = a_{44} - l_{41}u_{14} - l_{42}u_{24} - l_{43}u_{34} = 1 - \dfrac{1}{4} \cdot 1 - \dfrac{1}{3} \cdot \dfrac{1}{4} - \dfrac{1}{2} \cdot \dfrac{1}{3}$ であるが，例えば第 4 項の $\dfrac{1}{2}$ は $\dfrac{12288}{24576}$ になっているので，分母・分子ともに 9 桁の数になる．なお $905969664 = 2^{25}3^3$ である．

フランク行列は行列式が 1 になる特殊な行列なので，桁数の増加が小さいが，ヒルベルト行列の場合は桁数がより大きくなる．次数 4 のヒルベルト行列を分解すると，次のようになる．

$$\begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix} = \begin{pmatrix} \frac{1}{1} & & & \\ \frac{1}{2} & 1 & & \\ \frac{1}{3} & \frac{1}{1} & 1 & \\ \frac{1}{4} & \frac{9}{10} & \frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ & \frac{1}{12} & \frac{1}{12} & \frac{3}{40} \\ & & \frac{1}{180} & \frac{1}{120} \\ & & & \frac{1}{2800} \end{pmatrix} \tag{12}$$

[5] $\left(\dfrac{4}{4} + \dfrac{16}{48}\right)$ の計算では，筆算では分母は 48 にするが，有理算術演算で約分しない場合は，式 (2) によって計算するので，分母は $4 \cdot 48$ より 192 になる．

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 の対角項を記述する．等式の左辺が約分した計算，右辺が約分しない場合を示す．

$$\begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ & \frac{1}{12} & \frac{1}{1} & \frac{9}{10} \\ & & \frac{1}{180} & \frac{3}{2} \\ & & & \frac{1}{2800} \end{pmatrix} = \begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ & \frac{1}{12} & \frac{24}{24} & \frac{36}{40} \\ & & \frac{144}{25920} & \frac{14929920}{9953280} \\ & & & \frac{44030125670400}{123284351877120000} \end{pmatrix}$$

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分子が 14 桁の 44030125670400 となり，分母はその 2800 倍で 18 桁になる．

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，解ベクトルは「分母 = 分子」つまり 1 であるが，その数は x_1 （の分子，分母）が 191 桁，x_2 が 97 桁，x_3 が 55 桁，x_1 が 38 桁になる．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に**「ループ計算の中に対角項で割る操作を含む」行列計算アルゴリズムを，GCD 計算を省略して行うと，桁数の膨張で計算不能に陥る**．ループ内の除算は「分母 2 べき」を保存せず，これが他の行列要素の乗数として用いられるので，元の行列要素の分母が 2 べきでも，中間結果は分母 2 べきではないので，GCD 計算に時間を要する．

3.1.2 整数性と桁数激減

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[6]．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分数であるが，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 のヒルベルト行列を LDL 分解すると，式 (12) より $d_1 = 1$，$d_2 = \frac{1}{12}$，$d_3 = \frac{1}{180}$，$d_4 = \frac{1}{2800}$ となり，$\det(A) = \frac{1}{6048000} = 0.000000165\cdots$ 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，$\frac{1}{3}$ が丸められるので，行列式は分子が 58 桁，分母が 65 桁の分数になる．これは途中の d_3 や d_4 が桁数の多い分数になっているからである．

有理数行列の要素の分母の 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[7]．

$$\begin{aligned} d_1 &= 36028797018963968 \\ d_2 &= 3002399751580330 \\ d_3 &= \frac{270648451357573893329822394739598562875328769 9}{135216064024344439468984159436 80} \\ d_4 &= \frac{348254932766439364688124334918317072376904048372928722086 64}{270648451357573893329822394739598562875328769 9} \\ \det(A) &= 27860394621315149175049946793465365790152323869834297766931 2 \end{aligned}$$

行列 A を軸選択なしで LU 分解すると，対角項は $\det(A_0) = 1$ として，

$$u_{ii} = \frac{\det(A_{i,i})}{\det(A_{i-1,i-1})} \tag{13}$$

[6] 2×2 の場合を確かめると，$\det(A) = a_{11}a_{22} - a_{12}a_{21}$ であるが，$a_{11} = \frac{n_{11}}{d_{11}}$ と分数の形で書き直すと，$\det(A) = \frac{n_{11}}{d_{11}} \cdot \frac{n_{22}}{d_{22}} - \frac{n_{12}}{d_{12}} \cdot \frac{n_{21}}{d_{21}}$ であり，これの分母は $d_{11}d_{22}d_{12}d_{21}$ なので，整数行列の場合は 1，つまり行列式は整数になる．

[7] 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．

と，首小行列の行列式の比が現れる [6, p. 203]．したがって

$$\prod_{i=1}^{n} u_{ii} = \prod_{i=1}^{n} \frac{\det(A_i)}{\det(A_{i-1})} = \det(A_n) \tag{14}$$

である．$A = LDL^T$ と分解した場合は $u_{ii} = d_i$ である（フランク行列の例 (11) で確認されたい）．

この例の場合は，d_3 の分子は d_4 の分母に等しくなっている[8]．したがって，総積を計算する過程で約分されて，行列式は整数になる．

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．行列 A の固有値 λ を求める場合，特性多項式 $p(\lambda) = \det(A - \lambda I)$ の零点を計算するが，A が整数行列の場合，特性多項式 $p(\lambda)$ の係数も，行列式から得られるので整数になる．したがってこの A をフロベニウス標準形に変換する過程でも，行列式の計算と同じような桁数の激減する文脈が現れる．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

連分数の計算は，有理算術演算の特長をよく表す．反復計算の依存性が強いにも関わらず，展開された連分数を，昇順にも降順にも計算できる．ここではじめに，有理数を正則連分数（regular continued fraction）に展開すると有限回で展開しきることを，1000 桁の π の近似値を用いた例題で示す．次に規則性をもつ一般の連分数で π を求める計算で GCD 計算を間引いたほうが速い例を示す．

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[9]．

$$x_0 = a_0 + \frac{1}{x_1} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{x_2}} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{x_3}}} = \cdots \tag{15}$$

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．

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 [7]．

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「ユークリッドの互除法」の BASIC プログラムが掲載されていた[10]．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．“$a = bq + r$” を “$a = d_{-2}$” と “$b = d_{-1}$” と “$r = d_0$” とおき漸化式

$$d_{i-2} = d_{i-1}q + d_i \tag{16}$$

[8] $d_1 d_2 \div 8$ は d_3 の分母に等しい．

[9] $0 \le x < a_0 + 1$ となる整数 a_0 を見つけ，$x = a_0 + (x - a_0)$ と書き $0 \le x - a_0 < 1$ ならば $x_1 = \dfrac{1}{x - a_0}$ とおけば $x = a_0 + \dfrac{1}{x_1}$ になる．これを $x_i = 0$ になるまで繰り返す．

[10] 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．

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 (16) の両辺を d_{i-1} で割り，分数 $\dfrac{d_{i-2}}{d_{i-1}} = x_i$ とおくと連分数展開の漸化式 $x_i = a_i + \dfrac{1}{x_{i+1}}$ 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[11]．

```
int main(void){
    rational pi,p,r,x,y;
    uint32_t i,n,m;
    std::cout << "Enter n" << std::endl;
    std::cin >> n;
    longint pin("31415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679 略 ");
    longint pid("10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 略 ");
    rational pi1000(pin,pid);
    rat::vector<rational> a(n+1);   // rational 型の配列
    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;                                  //  残りの
        if(y==rational::ZERO){m=i-1;break;}
        x = y.inv();                            //  逆数をとる
        p = x.rational::floor(); // x_i の整数部分を取り出し
        a[i] = p;                               // それを a_i として
    }
    r=a[m];
    for(i=m-1;i>0;i--)r=a[i]+rational::ONE/r;
    pi=a[0]+rational::ONE/r;
    std::cout << "pi="<< pi.format(n+10,n)<< std::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によって式 (15) の a_i 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[12]，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 a_n から降順（$i = n, n-1, \cdots, 2, 1$）に連分数 a_i から第 n 近似分数 k_n を得る．$\pi - 3$ の正規連分数展開は $[a_1, a_2, a_3, a_4, a_5, \cdots] = [7, 15, 1, 292, 1, \cdots]$ となり，最良近似分数（best approximation fraction）は $\dfrac{1}{7}$, $\dfrac{15}{106}$, $\dfrac{16}{113}$, $\dfrac{4687}{33102}$, $\dfrac{4703}{33215}$, $\cdots$ となる．

`pi` の表示は `format` 関数に桁数を指定して書くが[13]，ここでは小数点以下 `n` 桁とした．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．`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．第 n 近似分数 $\dfrac{p_n}{q_n}$ の誤差は，$\dfrac{1}{q_n^2}$ より小さく，分母の大きさの割に高精度である．

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 [7]．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．

[11] $\pi \fallingdotseq 3.14159$ とすれば，$3.14159 = \dfrac{314159}{100000}$ と表す．

[12] 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．

[13] `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．

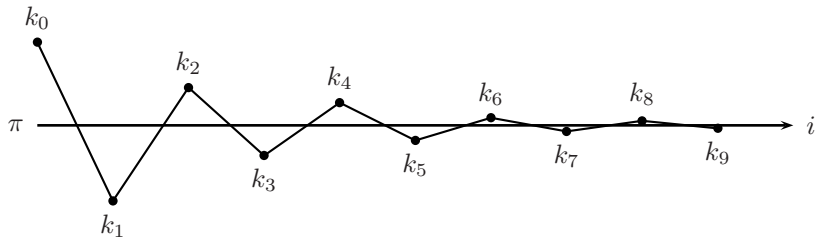

図 1: 連続する k_i による包含

x_{n+1} を無視して a_n までとると**第 n 近似分数**といい，ここでは $k_n = \dfrac{p_n}{q_n}$ と表記する．

$$k_0 = \frac{a_0}{1} = \frac{p_0}{q_0}$$

$$k_1 = a_0 + \frac{1}{a_1} = \frac{a_0a_1 + 1}{a_1} = \frac{p_1}{q_1}$$

$$k_2 = a_0 + \frac{1}{a_1 + \dfrac{1}{a_2}} = \frac{a_0a_1a_2 + a_0 + a_2}{a_1a_2 + 1} = \frac{p_2}{q_2}$$

$$k_3 = a_0 + \frac{1}{a_1 + \dfrac{1}{a_2 + \dfrac{1}{a_3}}} = \frac{a_0a_1a_2a_3 + a_0a_1 + a_0a_3 + a_2a_3 + 1}{a_1a_2a_3 + a_1 + a_3} = \frac{p_3}{q_3}$$

したがって

$$p_0 = a_0, \quad q_0 = 1$$

$$p_1 = a_0a_1 + 1, \quad q_1 = a_1$$

$$p_2 = a_0a_1a_2 + a_0 + a_2 = a_2(a_0a_1 + 1) + a_0 = a_2p_1 + p_0, \quad q_2 = a_1a_2 + 1 = a_2q_1 + q_0$$

$$p_3 = a_3(a_0a_1a_2 + a_0 + a_2) + a_0a_1 + 1 = a_3p_2 + p_1, \quad q_3 = a_3(a_1a_2 + 1) + a_1 = a_3q_2 + q_1$$

これを続けると次の 3 項漸化式が得られる．

$$p_i = a_ip_{i-1} + p_{i-2}, \quad q_i = a_iq_{i-1} + q_{i-2} \tag{17}$$

証明は帰納法で行う [8, p. 45]．式 (17) によって計算すれば，降順の反復を昇順で計算できる．

正則連分数はユークリッドの互除法と同じなので，連続する 2 つの近似分数 k_{i-1} と k_i によって，被近似数を包含（inclusion）できる．図 1 にこれを示す．これは次のように証明される [8, p. 46]．

Proof. 連続する 2 つの近似分数 $k_{i-1} = \dfrac{p_{i-1}}{q_{i-1}}$, $k_i = \dfrac{p_i}{q_i}$ の分子と分母について

$$p_iq_{i-1} - p_{i-1}q_i = (-1)^{i-1} \tag{18}$$

が成立する．これは，式 (18) の左辺の p_i，q_i に式 (17) を代入すると，下線部

$$p_iq_{i-1} - p_{i-1}q_i = (\underline{a_ip_{i-1}} + p_{i-2})\underline{q_{i-1}} - \underline{p_{i-1}}(\underline{a_iq_{i-1}} + q_{i-2}) = -(p_iq_{i-2} - p_{i-2}q_i) \tag{19}$$

がキャンセルして，添字がひとつ小さくなり，符号反転した式になる．したがってこれを続けると $(-1)^{i-1}$ にゆきつくからである．

式 (18) の両辺を $q_{i-1}q_i$ で割ると

$$k_i - k_{i-1} = \frac{(-1)^{i-1}}{q_{i-1}q_i} \tag{20}$$

が得られ，**解（この場合は π）を交互に挟み込んで収束に向かう**性質が証明される．

式 (19) の右辺の p_i と q_i に式 (17) を代入して添字をもうひとつ小さくする．

$$-((a_ip_{i-1} + \underline{p_{i-2}})\underline{q_{i-2}} - \underline{p_{i-2}}(a_iq_{i-1} + \underline{q_{i-2}})) = -a_i(p_{i-1}q_{i-2} - p_{i-2}q_{i-1}) = (-1)^i a_i \tag{21}$$

最後の等式は式 (18) を代入した．そこで両辺を $q_{i-2}q_i$ で割って，

$$k_i - k_{i-2} = \frac{(-1)^i a_i}{q_{i-2}q_i} \tag{22}$$

を得る．これから，**偶数項または奇数項のみに着目すれば，それぞれが単調に収束に向かう**ことも証明される．したがって不等式

$$k_0 < k_2 < k_4 < \cdots < \pi < \cdots < k_5 < k_3 < k_1 \tag{23}$$

が成立する． □

これによって「連続する 2 項による区間 (k_{i-1}, k_i) または (k_i, k_{i-1}) が π を包含する」ことが言える．

3.2.2 円周率の計算

$\arctan x$ の連分数展開は次式で表される [8, p. 51]．

$$\arctan x = \cfrac{x}{1 + \cfrac{x^2}{3 + \cfrac{(2x)^2}{5 + \cfrac{(3x)^2}{\ddots \cfrac{}{2n-1 + \cfrac{(nx)^2}{\ddots}}}}}} \tag{24}$$

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（この公式の導出の概略を付録に記した．詳しくは参考文献 [9] を参照されたい）．$x = 1$ とおくと $\dfrac{\pi}{4}$ が得られ，

$$\frac{\pi}{4} = \cfrac{1}{1 + \cfrac{1^2}{3 + \cfrac{2^2}{5 + \cfrac{3^2}{7 + \cfrac{4^2}{\underbrace{9 + \underbrace{\ddots}_{v_5}}_{v_4}}}}}} \tag{25}$$

4 倍すると π が得られる．

なお，この形の連分数

$$a_0+\cfrac{b_1}{a_1+\cfrac{b_2}{a_2+\cfrac{b_3}{a_3+\cfrac{b_4}{a_4+\cfrac{b_5}{a_5+\ddots}}}}} \tag{26}$$

を一般の連分数という．$b_i=1$ の場合が正則連分数になる．

式 (25) の具体的な計算方法を考える．$a_i=2i-1$，$b_i=(i-1)^2$ と置いて $i=2,\cdots,n$，ただし $b_1=1$ で考える．

はじめに末端の分数を v_n と置いて，**降順** $i=n,n-1,\cdots,2$ **に反復計算**する方法を示す．式 (25) で，$v_5=\ddots$，$v_4=\cfrac{4^2}{9+\ddots}$ とおけば，第 n 近似分数を求める計算は

$$v_{i-1}=\frac{i^2}{(2i+1)+v_i} \tag{27}$$

を i が $n-1$ から 2 まで反復して，最後に 4 倍の処理を入れればよいことがわかる．

具体的に第 2 近似分数の計算例を示す．

$$k_2=\cfrac{4}{a_1+\cfrac{1^2}{a_2+\cfrac{2^2}{5}}}=\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5}}}=\cfrac{4}{1+\cfrac{1^2\cdot5}{3\cdot5+2^2}}=\frac{4\cdot19}{19+5}=\frac{76}{24}=3.16666\cdots$$

第 3 近似分数の計算例を示す．

$$k_3=\cfrac{4}{a_1+\cfrac{1^2}{a_2+\cfrac{2^2}{a_3+\cfrac{3^2}{7}}}}=\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7}}}}=\cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2\cdot7}{35+3^2}}}=\cfrac{4}{1+\cfrac{1^2\cdot44}{3\cdot44+28}}=\frac{160}{51}=3.1372549\cdots$$

第 2 近似分数 k_2 は π より大きく，第 3 近似分数 k_3 は π より小さい．$k_0=4$, $k_1=3$ とおくと，k_i は i が偶数のとき π より大きく，奇数のとき π より小さい．なお，第 n 近似分数，例えば $\dfrac{76}{24}$ も，途中に現れる分数，例えば $\dfrac{28}{44}$ なども既約でないところが正則連分数と異なる．

正則連分数のように，第 i 近似分数 $k_i=\dfrac{p_i}{q_i}$ と第 $(i-1)$ 近似分数 $k_{i-1}=\dfrac{p_{i-1}}{q_{i-1}}$ による区間が π を挟むことは，次のように証明される（連分数に関する標準的な整数論の教科書 [10] は，正則連分数について詳しいが，一般の連分数については記載されていない．各等式の導き方は正則連分数の場合と同じ）．

Proof. 正則連分数の 3 項漸化式 (17) に対応する式を示す．

$$p_i=a_ip_{i-1}+b_ip_{i-2},\quad q_i=a_iq_{i-1}+b_iq_{i-2} \tag{28}$$

連続する 2 つの近似分数に対する分子と分母の関係式 (18) は次のようになる．

$$p_iq_{i-1}-p_{i-1}q_i=(-1)^{n-1}b_ib_{i-1}\cdots b_1 \tag{29}$$

解を交互に挟み込んで収束に向かう性質を保証する式 (20) は次のようになる．

$$k_i-k_{i-1}=\frac{(-1)^{i-1}b_ib_{i-1}\cdots b_1}{q_{i-1}q_i} \tag{30}$$

偶数項または奇数項のみに着目すれば，それぞれが単調に収束に向かうことを証明する式 (22) は次式に変わる．

$$k_i - k_{i-2} = \frac{(-1)^i a_i b_{i-1} b_{i-2} \cdots b_1}{q_{i-2} q_i} \tag{31}$$

b_i は正の整数なので，π を連続する 2 つの近似分数が包含する不等式 (23) が成立する． □

式 (30) を $i = 3$ で確かめると，$k_2 - k_3 = \frac{76}{24} - \frac{160}{51} = \frac{1}{34}$ であり，$\frac{b_3 b_2 b_1}{q_2 q_3} = \frac{(-1)^2 9 \cdot 4}{24 \cdot 51} = \frac{1}{34}$ である．この式は，連続する 2 項による区間の幅を与えることが分かる．

3 項漸化式による昇順反復計算　分母と分子を独立に計算すると，それぞれは自然数なので，rational 型の変数を使っても，GCD 計算は行わない．

```
int main() {
   uint32_t n;
   std::cout << "input n=" << std::endl;
   std::cin >> n ;
   rational p,q,p0,q0,p1,q1,u,v,x;
   interval pi;
   long long tstart0=gettime();
#ifdef LGCD1
      longint::lgcd_p = longint::lgcd1;
#endif
   p0=rational::FOUR; q0=rational::ONE;
   p1=rational::TEN+rational::TWO; q1=rational::FOUR;
   for(uint32_t i=2; i<=n; i++){
      x=i;
      p=(rational::TWO*x+rational::ONE)*p1+(x*x)*p0;
      q=(rational::TWO*x+rational::ONE)*q1+(x*x)*q0;
      p0=p1; q0=q1;
      p1=p;  q1=q;
   }
   std::cout << "loop time=" << (double)(gettime()-tstart0)/1000000. << "(sec)" << std::endl;
// longint::lgcd_p = longint::lgcd2;
   u=p0/q0; v=p1/q1;
   if(u<v){
      pi=interval(u,v);
   }else{
      pi=interval(v,u);
   }
   std::cout << " pi="; formatprn(pi);
   return 1;
}
```

`formatprn` 関数によって，$n = 2000$ を与えた場合の結果を示す[14]．

2000 項で計算した結果

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

$n = 2000$ はすぐに計算が終了するが，10 倍の $n = 20000$ 項で計算すると，パソコンでは反復に 10 秒ほどかかり，上記の 1500 digits が 15281 digits と表示される．`-DLGCD1` を指定してコンパイルすると，反復後の 2 つの除算 `u=p0/q0` と `v=p1/q1` で約分しないので，桁数が大きいままで，その後の formatprn で

[14] `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．

計算時間を要する[15]．2000 の場合は，p も q も 2^{32} 進数で 677 桁の，合計 1354 桁が，約分された分数では 400 桁に落ちる．20000 の場合は，8825 桁の，合計 17650 桁が，約分された分数では 4014 桁に落ちる．この桁数の削減をしないで formatprn で 10 進変換すると遅くなるため，反復計算で GCD を省略しても，2 つの除算では GCD 計算は省略しないほうがよい．

降順反復計算　n を与えられたら，第 n 近似分数の初項に最初の反復を行った中間解と，第 $(n-1)$ 近似分数の初項を interval 型の変数に作り，ループ添字 i を $n-2$ から 1 まで降順に反復する．

第 n 近似分数 k_n の初項は $v_n = \dfrac{n^2}{2n+1}$ であるから，反復計算式 (27) から，

$$v_{n-1} = \frac{(n-1)^2}{(2n-1)+\dfrac{n^2}{2n+1}} = \frac{(n-1)^2(2n+1)}{n^2+(2n+1)(2n-1)}$$

になる．一方，第 $(n-1)$ 近似分数の初項は $v_{n-1} = \dfrac{(n-1)^2}{2(n-1)+1}$ である．両者を有理数型の変数 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，$(n-2)$ から 1 まで降順に反復し，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= ··· と v= ··· 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．プログラム PiCfracAtanRint.cpp を示すが，この計算も，毎回 GCD を計算すると遅くなる．連分数から近似分数を求める計算は，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．

```
        uint32_t n;
        double 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,xx,tmp;
        rational pit,piu;
        long long tstart0=gettime();
    #ifdef LGCD1
        std::cout << "LGCD1 defined.  USE lgcd1 for lgcd_p" << std::endl;
        longint::lgcd_p = longint::lgcd1;
    #endif
        p=n*n; q=2*n+1; tmp=2u*(n-1u)+1u; t=p+q*tmp;           //  n^2+(2n+1)*(2n-1)
        tmp=(n-1u)*(n-1u);
        p=q*tmp;         q=t;            t=p/q; // t = (2n+1)*(n-1)^2 / (n^2+(2n+1)*(2n-1))
        r=(n-1u)*(n-1u); s=2u*(n-1u)+1u; u=r/s; // u = (n-1)^2 / 2(n-1)+1
        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);       //  set number of digits in lastdgts
        for(uint32_t i=n-2; i>0; i--){
            x=i; xx=x*x;
            interval z=v+(rational::TWO*x+rational::ONE);
            interval w=z.inv();
            rational vlow=v.lower();  //  lower number picked for getdgts();
            if((double)getdgts(vlow)/(double)lastdgts >= k){   // check digits increase
                longint::lgcd_p = longint::lgcd2;
                v=w*xx;
                longint::lgcd_p = longint::lgcd1;
                vlow=v.lower(); lastdgts=getdgts(vlow); // set number of digits in lastdgts
```

[15] formatprn では内部で lgcd_p=lgcd1 を指定しているが，引数の桁数が大きくなっているので遅くなる．

```
        }else{
            v=w*xx;
        }
    }
    std::cout << "loop finish. time=" << (double)(gettime()-tstart0)/1000000.<< "(sec)"<<std::endl;
    vlow=v.lower(); rational vup=v.upper();
    std::cout << "numdgts lower=" << getdgts(vlow) << " and  upper=" << getdgts(vup) << std::endl;
    v=v+rational::ONE;
    interval y=v.inv();
    longint::lgcd_p = longint::lgcd2;  //    FORCE lgcd2 for *4 operation
    y=y*rational::FOUR;
    std::cout << " pi="; formatprn(y);
#ifdef LGCD1
    std::cout << "LGCD1 defined.  USE lgcd1 for lgcd_p" << std::endl;
    longint::lgcd_p = longint::lgcd1;
#endif
    rational vl=y.lower();
    std::cout << " num dgts of pi (after lgcd2) =" << getdgts(vl) << std::endl;
    std::cout << "total time=" << (double)(gettime()-tstart0)/1000000.0 << "(sec)" << std::endl;
    return 1;
}
```

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 関数で変数 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 倍になったら約分する．

実測した範囲では $k = 1.8$ の近傍が適当であるようだ．$n = 20000$ の例では，$k = 1.8$ の場合は 20000 回中 21 回 GCD を求めて約分する．

区間幅を指定した計算　π は無理数なので，π を含む区間の幅を指定して，反復がその区間幅まで狭められたらその区間を返すように実装するのが便利である．この場合は式 (30) を使うと，反復中に $\frac{p}{q}$ を計算するよりも速い．プログラム PiCfracAtanRintX.cpp を示す．

```
    uint32_t i,m;
    rational bprod,spint,pint,p,q,p0,q0,p1,q1,u,v,x,xx;
    interval pi;
#ifdef BYDIV
    rational temp0,temp=rational::ONE;
#endif
    std::cout << "input m=" << std::endl;
    std::cin >> m;
    spint=rational::ONE;
    for(i=1; i<m; i++) spint = spint/rational::TEN;
    long long tstart0=gettime();
    longint::lgcd_p = longint::lgcd1;
    bprod=rational::ONE;
    p0=rational::FOUR; q0=rational::ONE;
    p1=rational::TEN+rational::TWO; q1=rational::FOUR;
    i=1;
    for(;;){
        i++; x=i; xx=x*x; bprod*=xx;
        p=(rational::TWO*x+rational::ONE)*p1+xx*p0;
        q=(rational::TWO*x+rational::ONE)*q1+xx*q0;
        p0=p1; q0=q1; p1=p;  q1=q;
#ifdef BYDIV
        temp0=temp; temp=p/q;
        if(i%2==1){
            pint=temp0-temp;
        }else{
            pint=temp-temp0;
        }
```

```
#else
        pint=bprod/(q0*q1);
#endif
        if(pint < spint) break;
    }
    std::cout << "loop finish. time=" << (double)(gettime()-tstart0)/1000000.<< "(sec)"<<std::endl;
    std::cout << "i=" << i << std::endl;
    std::cout << "digits of p=" << getdgts(p) <<  "  digits of q=" << getdgts(q) <<  std::endl;
#ifdef LGCD1
    longint::lgcd_p = longint::lgcd2;
#endif
    u=p0/q0; v=p1/q1;
    std::cout << "digits of v=" << getdgts(v) <<  std::endl;
    if(u<v){
        pi=interval(u,v);
    }else{
        pi=interval(v,u);
    }
    std::cout << " pi="; formatprn(pi);
```

まとめ　ガウス超幾何関数によって表された arctan を一般連分数に展開した式によって π を，昇順と降順のループ構成で計算してみた．連分数計算は有理算術演算の特長をよく表す計算である．昇順では自然数である分母 q_n と分子 p_n を独立に反復計算するので，GCD は最後に $\frac{p_n}{q_n}$ を求めるところで使用するだけである．降順では反復計算式 (27) の変数 v_i が有理数なので，これを既約にするタイミングを最適化することで高速化できる．

有理算術演算で初等関数を計算する場合，無理数解をユーザ指定の精度（区間幅）で求めて返す形で実装したい．三角関数は引数を基本周期に還元する計算（argument reduction）が要求されるが，これには高精度の π が必要である．したがって，三角関数の開発段階では，指定された区間幅を満たす π を含む区間を返す関数が必要である．この要件には，昇順に計算すれば，区間幅が要求精度を満たしたところで反復を打ち切るように実装できる．一方，降順では精度は最後にしかわからないので，要件を満たす実装はやさしくない．このようなループ計算の特長は，浮動小数点演算では現れない有理算術演算に独特の性質である．有理算術演算では，桁数の制御に GCD 計算が必要で，この計算が本計算よりも重い場合に顕著になる．

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．浮動小数点演算では，縮小反復には「2 分法」や「挟み撃ち法」を用いることが多い．2 分法は，解の存在する区間 $(a,\ b)$ の中点 $c = \frac{a+b}{2}$ で関数値を求め，$p(a) \cdot p(c) < 0$ 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，2 分法の中点の代わりに x 切片

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} \tag{32}$$

を次の反復での新たな a または b として採用する．

有理算術演算で両者を試してみると，浮動小数点演算の場合とは計算時間の比が全く異なり，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 $a = \frac{a_n}{a_d}$ の

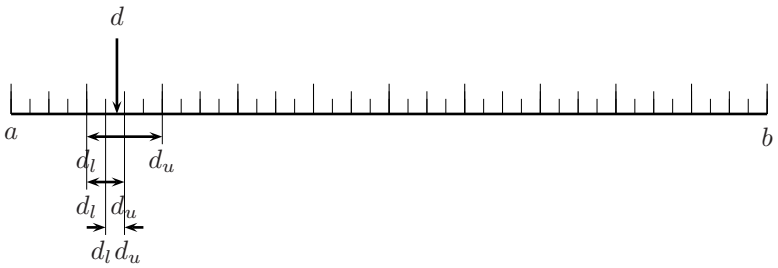

図 2: d を含む精度による 3 通りの区間 (d_l, d_u)

ように記述すると，$c = \dfrac{a_n b_d + a_d b_n}{2a_d b_d}$ 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．しかし挟み撃ち法は $f(b) - f(a)$ で割るので，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．「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．有理数 $x = \dfrac{x_n}{x_d}$ と分母の 2 のべき k を与えると，y の分子を区間

$$y_n = \left(\lfloor \frac{2^k x_n}{x_d} \rfloor, \lceil \frac{2^k x_n}{x_d} \rceil \right) = (y_{nl}, y_{nu}) \tag{33}$$

で求めて，$y = \left(\dfrac{y_{nl}}{2^k}, \dfrac{y_{nu}}{2^k} \right)$ 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 (32) による座標値を d とする．roundrat 関数は 2 進数の精度を指定されると，d を挟む 2 つの 2 進数を interval 型の変数の上下限として返す．2 進数 $\dfrac{a}{2^k}$ のスクリーン幅は，2 のべき k が 1 つ大きくなると半分になる．図 2 に，区間 (a, b) 間に d が得られたとき，これを含む 2 進数のスクリーンを 3 通りの精度で示した．roundrat 関数に指定する精度によって，返される区間 (d_l, d_u) も 3 通りになる．上の区間 (d_l, d_u) に対して，中の区間は 2 倍の精度を指定した場合，下の区間は 4 倍の精度を指定した場合である．d は任意の有理数であるが， d_l, d_u は分母が 2 のべき乗，つまり 2 進数の有理数である．

得られた区間を使って，2 分法と挟み撃ち法を折衷した方法を採用した．精度は $d - a$ と $b - d$ の小さいほうの小数点以下のビット数を基準に与え[16]，(d_l, d_u) を得たら，d が区間 (a, b) で a に近い場合は，$h = d_u - a$ として，$a + h,\ a + 2h.\ a + 4h.\ a + 8h. \cdots$ と h を倍々に増やして $f(a) \cdot f(a + 2^k h) < 0$ となる k を探す．d が区間 (a, b) で b に近い場合は，$h = b - d_u$ として，$b - h,\ b - 2h.\ b - 4h.\ b - 8h. \cdots$ と h を倍々に増やして $f(b) \cdot f(b - 2^k h) < 0$ となる k を探す．包含が成立した $a + 2^k h$ または $b - 2^k h$ を次の反復解 c とする．

具体的な**数値例**を紹介する．式 (32) で得られる x 切片の座標値は，特性多項式 $f(x)$ の係数が 300 桁を超える整数になっており，分子 $af(b) - bf(a)$ の桁数は大きく，さらに分母に $f(b) - f(a)$ があるため，分母・分子が 1435 桁になっている．

$$d = -\frac{66361(1425 \text{ digits })59873}{28355(1425 \text{ digits })69856} = -2.340375387850571 \cdots \tag{34}$$

[16]roundrat が $2^{10} = 1024$ の倍数を引数で指定するので，ビット数を 10 で割ることで，10 の 3 乗と 2 進と 10 進の違いを丸めて合わせている．後述する例では，$d - a = 8.5 \times 10^{-18}$ のとき $\log_2(8.5 \times 10^{-18}) = -56$ 程度になるので，`nprec` $= 11$ になり，$(2^{10})^{11}$ が分母になる．

これを roundrat 関数で丸めて次の区間を得る．

$$\begin{aligned}(d_l, d_u) &= \left(-\frac{30379(24\text{ digits })84523}{12980(24\text{ digits })05024}, -\frac{15189(23\text{ digits })92261}{64903(22\text{ digits })52512}\right)\\ &= \left(-\frac{30379(24\text{ digits })84523}{2^{110}}, -\frac{15189(23\text{ digits })92261}{2^{109}}\right)\\ &= (-2.340375387850571\cdots, -2.340375387850571\cdots)\end{aligned} \tag{35}$$

倍精度の 17 桁では差異は現れない．

区間 (a, b) で，式 (34) の d と式 (35) の d_l と d_u の間には

$$a < d_l < d < d_u < b \tag{36}$$

の関係が守られれば (d_l, d_u) は高速化に都合のよいように選ぶべきである．ここでは「分母 2 べき」を選び，桁数も 1435 から 34 桁に減らした．

この 2 分法と挟み撃ち法を折衷した方法により，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[17]．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 2 べき」である）．特性多項式の値は加減算と乗算だけで評価できるので，分母 2 べきは保存される．近似解でも包含が成立すれば，2 分法やここで述べた方法で必要なだけ高精度化することが可能であり，**整数性**から因数分解の代替となる因子探しができる [11]．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，ここで述べた方法ではこれを 800 回程度に減らすことができ，精度改良の時間を半分以下（43 秒から 20 秒）にすることができた．

まとめ　計算式が加減算と乗算だけ（除算なし）で，入力データが倍精度浮動小数点数を有理数変換した数の場合は，有理数が「分母 2 べき」を保持すると計算速度を高められる．そのためには有理数から区間を得るとき，区間端を「分母 2 べき」に丸めることが高速化に効果がある．

5　まとめと今後の課題

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，小さな例で桁数膨張を紹介した．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．

無理数解を追究する場合，区間算術演算を使用するが，数式が加減算と乗算だけで構成される場合は「分母 2 べき」は保存される．倍精度浮動小数点計算で得られた近似値は 2 進数なので，有理数変換すると「分母 2 べき」である．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，区間端点の値を適当な精度の 2 進数に丸めて，2 進 GCD アルゴリズムの高速性を利用すると効果があることを確かめた．

今後の課題として，初等関数の実装を試してみたい．

[17] 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（最適化オプションの影響で重根が消滅して）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 12 次と 13 次の因子 $s_{12}(\lambda)$ と $t_{13}(\lambda)$ を整数性を用いて探すところに用いた．

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．

付　録

A　ガウス超幾何関数による逆正接関数の連分数展開

ガウスによる超幾何級数の連分数展開を用いて得られる公式について補足する．実数 α と自然数 n に対して，セミコロンを用いた表記で，

$$(\alpha;n)=\alpha(\alpha+1)\cdots(\alpha+n-1)$$

とおく．さらに，$(\alpha;0)=1$ とする．$(1;n)=n!$ である．$\alpha,\beta,\gamma\neq 0,-1,-2,\ldots$ に対して，べき級数

$$F(\alpha,\beta,\gamma;x)=\sum_{n=0}^{\infty}\frac{(\alpha;n)(\beta;n)}{(\gamma;n)n!}x^n \tag{37}$$

は $|\,x\,|<1$ において収束する．

Proof. $a_n=\dfrac{(\alpha;n)(\beta;n)}{(\gamma;n)n!}$ とおけば，

$$\frac{a_n}{a_{n+1}}=\frac{(\alpha;n)(\beta;n)(\gamma:n+1)(n+1)!}{(\alpha;n)(\beta;n+1)(\gamma;n)n!}=\frac{(\gamma+n)(n+1)}{(\alpha+1)(\beta+1)}=\frac{\left(\dfrac{\gamma}{n}+1\right)\left(1+\dfrac{1}{n}\right)}{\left(\dfrac{\alpha}{n}+1\right)\left(\dfrac{\beta}{n}+1\right)}$$

より，収束半径は $\lim\limits_{n\to\infty}\dfrac{a_n}{a_{n+1}}=1$ である． □

このべき級数によって定義される関数 $F(\alpha,\beta,\gamma;x)$ を**ガウスの超幾何関数**と呼ぶ．パラメーター α,β,γ に特別な値を与えることによって，いろいろな初等関数が超幾何関数を用いて表せる．例えば，

$$\begin{aligned}
F(\alpha,0,\alpha;x)&=\sum_{n=0}^{\infty}\frac{(0;n)}{n!}x^n=1,\\
F(\alpha,\beta,\beta;x)&=\sum_{n=0}^{\infty}\frac{(\alpha;n)}{n!}x^n=\sum_{n=0}^{\infty}\binom{-\alpha}{n}(-x)^n=(1-x)^{-\alpha}\\
xF(1,1,2;x)&=\sum_{n=0}^{\infty}\frac{n!n!}{(n+1)!n!}x^{n+1}=\sum_{n=0}^{\infty}\frac{1}{n+1}x^{n+1}=-\log(1-x)\\
xF\left(\frac{1}{2},1,\frac{3}{2};-x^2\right)&=\sum_{n=0}^{\infty}\frac{1}{2n+1}(-1)^n x^{2n+1}=\arctan x
\end{aligned} \tag{38}$$

また，$\alpha=\gamma=1$，$\beta\to\infty$ の極限をとれば

$$e^x=\lim_{\beta\to\infty}F\left(1,\beta,1;\frac{x}{\beta}\right)$$

式 (38) は次のように導くことができる．

$$\begin{aligned} xF\left(\frac{1}{2},1,\frac{3}{2};-x^2\right) &= x\sum_{n=0}^{\infty}\frac{\left(\frac{1}{2};n\right)(1;n)}{\left(\frac{3}{2};n\right)n!}(-x^2)^n \\ &= x\sum_{n=0}^{\infty}\frac{\left(\frac{1}{2}\cdot\frac{3}{2}\cdots\frac{2n-1}{2}\right)(1\cdot 2\cdots n)}{\left(\frac{3}{2}\cdot\frac{5}{2}\cdots\frac{2n+1}{2}\right)(1\cdot 2\cdots n)}(-1)^n x^{2n} \\ &= x\sum_{n=0}^{\infty}\frac{1}{2n+1}(-1)^n\cdot x^{2n} = \sum_{n=0}^{\infty}\frac{1}{2n+1}(-1)^n\cdot x^{2n+1} \end{aligned}$$

一方，$\tan^{-1}x$ のテイラー展開は，$\tan^{-1}x$ を微分すると $\dfrac{1}{1+x^2} = 1-x^2+x^4-x^6+\cdots = \displaystyle\sum_{n=0}^{\infty}(-1)^n x^{2n}$ であることから，これを積分して $x-\dfrac{x^3}{3}+\dfrac{x^5}{5}-\cdots = \displaystyle\sum_{n=0}^{\infty}\frac{1}{2n+1}(-1)^n\cdot x^{2n+1}$ である．

超幾何関数の定義から，次の関係式を導くことができる．

命題 1　：ガウスの隣接関係式（contiguity relation）

$$F(\alpha,\beta,\gamma;x) = F(\alpha,\beta+1,\gamma+1;x) - \frac{\alpha(\gamma-\beta)}{\gamma(\gamma+1)}xF(\alpha+1;\beta+1,\gamma+2;x) \tag{39}$$

$$F(\alpha,\beta,\gamma;x) = F(\alpha+1,\beta,\gamma+1;x) - \frac{\beta(\gamma-\alpha)}{\gamma(\gamma+1)}xF(\alpha+1;\beta+1,\gamma+2;x) \tag{40}$$

Proof. 式 (39) の右辺は

$$\sum_{n=0}^{\infty}\frac{(\alpha;n)(\beta+1;n)}{\gamma;n)n!}x^n - \frac{\alpha(\gamma-\beta)}{\gamma(\gamma+1)}x\sum_{n=0}^{\infty}\frac{(\alpha+1;n)(\beta+1;n)}{(\gamma+2;n)n!}x^n$$

初項をそろえるため $(\beta+1;n) = \dfrac{(\beta;n+1)}{\beta} = \dfrac{(\beta;n)(\beta+n)}{\beta}$ や $(\gamma+2;n) = \dfrac{(\gamma+n+1)(\gamma;n+1)}{\gamma(\gamma+1)}$ などの置き換えを行うと

$$= \sum_{n=0}^{\infty}\frac{\gamma(\beta+n)(\alpha;n)(\beta;n)}{\beta(\gamma+n)(\gamma;n)n!}x^n - \sum_{n=0}^{\infty}\frac{(\gamma-\beta)(\alpha;n+1)(\beta;n+1)}{\beta(\gamma+1+n)(\gamma;n+1)n!}x^{n+1}$$

となるが，第 2 項の $n+1$ を n にすると，総和を $n=0$ からを $n=1$ からにできる．

$$= \sum_{n=0}^{\infty}\frac{\gamma(\beta+n)(\alpha;n)(\beta;n)}{\beta(\gamma+n)(\gamma;n)n!}x^n - \sum_{n=1}^{\infty}\frac{(\gamma-\beta)n(\alpha;n)(\beta;n)}{\beta(\gamma+n)(\gamma;n)n!}x^n$$

第 1 項の総和の $n=0$ の定数項は 1 なので

$$\begin{aligned} &= 1 + \sum_{n=1}^{\infty}\left(\frac{\gamma(\beta+n)-(\gamma-\beta)n}{\beta(\gamma+n)}\right)\frac{(\alpha;n)(\beta;n)}{(\gamma;n)n!}x^n \\ &= 1 + \sum_{n=1}^{\infty}\frac{(\alpha;n)(\beta;n)}{(\gamma;n)n!}x^n = F(\alpha,\beta,\gamma;x) \end{aligned}$$

$F(\alpha,\beta,\gamma;x) = F(\beta,\alpha,\gamma;x)$ に注意すれば，式 (39) で α と β を交換することによって式 (40) を得る．　□

(39) の両辺を $F(\alpha,\beta+1,\gamma+1;x)$ で割れば，

$$\frac{F(\alpha,\beta,\gamma;x)}{F(\alpha,\beta+1,\gamma+1;x)} = 1 - \frac{\alpha(\gamma-\beta)}{\gamma(\gamma+1)}x\frac{1}{\dfrac{F(\alpha,\beta+1,\gamma+1;x)}{F(\alpha+1,\beta+1,\gamma+2;x)}}$$

同様に (40) で (α,β,γ) を $(\alpha,\beta+1,\gamma+1)$ で置き換えて,

$$F(\alpha,\beta+1,\gamma+1)=F(\alpha+1,\beta+1,\gamma+2;x)-\frac{(\beta+1)(\gamma+1-\alpha)}{(\gamma+1)(\gamma+2)}xF(\alpha+1,\beta+2,\gamma+3;x)$$

この両辺を $F(\alpha+1,\beta+1,\gamma+2;x)$ で割れば,

$$\frac{F(\alpha,\beta+1,\gamma+1;x)}{F(\alpha+1,\beta+1,\gamma+2;x)}=1-\frac{(\beta+1)(\gamma+1-\alpha)}{(\gamma+1)(\gamma+2)}x\frac{1}{\dfrac{F(\alpha+1,\beta+1,\gamma+2;x)}{F(\alpha+1,\beta+2,\gamma+3;x)}}$$

したがって,

$$\frac{(\alpha,\beta,\gamma;x)}{F(\alpha,\beta+1,\gamma+1;x)}=1-\cfrac{\dfrac{\alpha(\gamma-\beta)}{\gamma(\gamma+1)}x}{1-\cfrac{\dfrac{(\beta+1)(\gamma+1-\alpha)}{(\gamma+1)(\gamma+2)}x}{\dfrac{F(\alpha+1,\beta+1,\gamma+2;x)}{F(\alpha+1,\beta+2,\gamma+3;x)}}}$$

(α,β,γ) を $(\alpha+1,\beta+1,\gamma+2)$ で置き換えることを繰り返せば,

$$\frac{F(\alpha,\beta,\gamma;x)}{F(\alpha,\beta+1,\gamma+1;x)}=1+\cfrac{a_1}{1+\cfrac{a_2}{\ddots\cfrac{}{1+\cfrac{a_{2n}x}{\dfrac{F(\alpha+n,\beta+n,\gamma+2n;x)}{F(\alpha+n,\beta+n+1,\gamma+2n+1;x)}}}}}\tag{41}$$

ここで

$$a_1=-\frac{\alpha(\gamma-\beta)}{\gamma(\gamma+1)},\quad a_2=-\frac{(\beta+1)(\gamma+1-\alpha)}{(\gamma+1)(\gamma+2)}$$

であり，(α,β,γ) を $(\alpha+n-1,\beta+n-1,\gamma+2n-2)$ で置き換えることによって，$n\geq 1$ に対して

$$a_{2n-1}=-\frac{(\alpha+n-1)(\gamma+n-1-\beta)}{(\gamma+2n-n)(\gamma+2n-1)}\tag{42}$$

$$a_{2n}=-\frac{(\beta+n)(\gamma+n-\alpha)}{(\gamma+2n-1)(\gamma+2n)}\tag{43}$$

定理 2　：超幾何関数の比 $F(\alpha,\beta,\gamma;x)/F(\alpha,\beta+1,\gamma+1;x)$ は，$|\,x\,|<1$ における有理型関数として，連分数展開

$$\frac{F(\alpha,\beta,\gamma;x)}{F(\alpha,\beta+1,\gamma+1;x)}=1+\cfrac{a_1x}{1+\cfrac{a_2x}{1+\cfrac{a_3x}{1+\cfrac{a_4x}{1+\ddots}}}}$$

を持つ．ここで，係数 a_{2n-1},a_{2n} は (43) によって与えられる．特に $\alpha,\gamma=\dfrac{1}{2}$，$\beta=0$ とすれば,

$$F\left(\frac{1}{2},0,\frac{1}{2};-x^2\right)=1,\qquad xF\left(\frac{1}{2},1,\frac{3}{2};-x^2\right)=\arctan x$$

であるから，定理 2 より，

$$\frac{1}{x^{-1}\arctan x} = 1 + \cfrac{-a_1x^2}{1 + \cfrac{-a_2x^2}{\ddots + \cfrac{\ddots}{1 + \cfrac{-a_{2n}x^2}{1 + \ddots}}}}$$

この逆数をとってから[18]，両辺に x をかければ，

$$\arctan x = \cfrac{x}{1 + \cfrac{-a_2x^2}{\ddots + \cfrac{\ddots}{1 + \cfrac{-a_{2n}x^2}{1 + \ddots}}}}$$

ここで，

$$a_{2n-1} = -\frac{\left(\frac{1}{2}+n-1\right)\left(\frac{1}{2}+n-1\right)}{\left(\frac{1}{2}+2n-2\right)\left(\frac{1}{2}+2n-1\right)} = -\frac{(2n-1)^2}{(4n-3)(4n-1)}$$

$$a_{2n} = -\frac{n\left(\frac{1}{2}+n-\frac{1}{2}\right)}{\left(\frac{1}{2}+2n-1\right)\left(\frac{1}{2}+2n\right)} = -\frac{(2n)^2}{(4n-1)(4n+1)}, \quad n \le 1$$

したがって，

$$-a_n = \frac{n^2}{(2n-1)(2n+1)}, n \le 1$$

とまとめられる．以上によって，$|\,x\,| < 1$ における正則関数として，$\arctan x$ の連分数展開

$$\arctan x = \cfrac{x}{1 + \cfrac{\frac{x^2}{1\cdot 3}}{1 + \cfrac{\frac{(2x)^2}{3\cdot 5}}{1 + \cfrac{\frac{(3x)^2}{5\cdot 7}}{\ddots + \cfrac{\ddots}{1 + \cfrac{\frac{(nx)^2}{(2n-1)(2n+1)}}{\ddots}}}}}} = \cfrac{x}{1 + \cfrac{x^2}{3 + \cfrac{(2x)^2}{5 + \cfrac{(3x)^2}{\ddots + \cfrac{\ddots}{2n-1 + \cfrac{(nx)^2}{\ddots}}}}}} \tag{44}$$

をえる．この右辺の連分数は，実数の閉区間 $[0,\,1]$ において，連続関数の一様収束極限であることがわかるから，$[0,1]$ における連続関数を表す．したがって，式 (44) の両辺で $x \to 1-0$ とすれば，$\arctan 1 = \dfrac{\pi}{4}$

[18] $1 + \frac{-a_1x^2}{Y}$ とおいて，通分して $\frac{Y - a_1x^2}{Y}$ の逆数は $\frac{Y}{Y - a_1x^2}$ で，分母，分子を Y で割ると $\frac{1}{1 - \frac{a_1x^2}{Y}}$

であるから，π の連分数展開

$$\pi = \cfrac{4}{1+\cfrac{1^2}{3+\cfrac{2^2}{5+\cfrac{3^2}{7+\cfrac{4^2}{9+\ddots}}}}} \tag{45}$$

を得る．

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. In *HPCS2013* **論文集**, pages 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. In *HPCS2014* **論文集**, pages 57–64, 2014.

[5] 寒川 光, 藤野清次, 長嶋利夫, and 高橋大介. *HPC* **プログラミング**—*ITText* **シリーズ**. オーム社, 2009.

[6] J. H. Wilkinson. *Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[7] 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.

[8] 小林昭七. **円の数学**. 裳華房, 1999.

[9] 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.

[10] 高木貞治. **初等整数論講義（第 *2* 版）**. 共立出版, 1971.

[11] 寒川 光. 講座　第 4 回「有理数計算」. **シミュレーション**, 34(4):46–55, 2015.