

有理数演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

寒川 光

概要：有理数計算は、浮動小数点演算や多倍長演算とは異なり、正確な計算結果を提供できるので数学的な価値が高い。しかし有理数計算の問題は、固有値計算のように数値の範囲が無理数に及ぶ計算ができないので、計算対象が制限される点にある。固有値問題の精度保証は、これまでは数学的、プログラミング的に高度な知識を必要とする手法を用いなければ困難であった。本論文では、平方根を陰的に処理する（開平を避ける）ことで正確な3重対角化を実現し、任意の高精度で精度保証された固有値を求める方法を、プログラムの概要を含めて報告する。有理数計算は、問題の規模が小さければ、十進 BASIC の有理数モードを用いて計算することが可能である。しかし通常の数値計算で扱う規模の問題では、かなり大規模なメモリと、並列化をとともう高速化が必要になる。また、有理数と浮動小数点数を混在させられれば、浮動小数点演算、区間演算、有理数演算を繋げた計算環境を実現できる。筆者は、有理数計算の機能を、多桁整数演算と有理数演算機能を C++ で実装し、浮動小数点演算と混在させるプログラミング環境を開発し、有理数計算を研究している。現在では、汎用サーバーでも 8 Tflop/s の理論ピーク性能と 8 TB のメモリ容量を実装するものもある [1]。有理数計算は、1 ビットの誤差に対しても敏感に反応する特徴があるので、計算機の検収で行われるベンチマークプログラムに適している。

キーワード：有理数計算、精度保証計算、対称行列固有値、平方根の陰的定式化

High precision symmetric eigenvalue computation through exact tridiagonalization by using rational number arithmetic

HIKARU SAMUKAWA

Abstract: Since rational number arithmetic, which is different from floating-point arithmetic or multiple-precision arithmetic, can provide exact computations, it is valuable from mathematical point of view. However its application area is limited because of its lack of ability to compute irrational numbers, such as eigenvalue analysis. A self-validation technique for eigenvalue analysis is difficult because it requires deep mathematical and programming knowledge. In this paper, we propose a method of exact reduction from symmetric matrix to tridiagonal matrix by using implicit square root formulation.

A rational number arithmetic is possible in decimal BASIC for small problems. However for the normal size problems, a huge memory and tuning including parallelization are required. We have developed a C++ programming environment capable of using rational number variables mixed with other types of variables upon multi-digit integers and rational number arithmetics layers. Currently a general purpose server provides 8 Tflop/s computing power and 8 TB memory capacity[1]. Since the rational number arithmetic is sensitive to a bit error, it is suitable to be used for benchmark programs.

Keywords: rational number arithmetic, self-validation computation, symmetric eigenvalue problem, implicit square root formulation

1. 有理数計算の特徴

有理数演算は、分母、分子を多桁の整数で保持し、四則演算を (a, b, c, d を多桁整数として) 次のように計算する。

- 加減算: $\frac{b}{a} \pm \frac{d}{c} = \frac{bc \pm ad}{ac}$
- 乗算: $\frac{b}{a} \cdot \frac{d}{c} = \frac{bd}{ac}$
- 除算: $\frac{b}{a} \div \frac{d}{c} = \frac{bc}{ad}$

n 桁と m 桁の整数の積は nm 桁になるので、演算のたびに分母、分子の桁数は増えるが、分母、分子の最大公約数で割り、既約分数にする [2][3]。有理数計算を実装したプログラミング言語に、教育目的で開発された十進 BASIC がある [4]。高校の数学 B の教科書の「数値計算とコンピュータ」の章では、十進 BASIC で例題を記述したものもある [5]。十進 BASIC では、数値は 10 進 15 桁、10 進 1000 桁、2 進数 (Intel x86 アーキテクチャの倍精度浮動小数点数)、実部と虚部をそれぞれ 10 進 15 桁で保持する複素数、有理数の 5 つのモードを選択できる。モードを選択すると、数値はすべてその型となるので、有理数と浮動小数点数を混在することはできない^{*1}。

浮動小数点数は有限のビット数で表される数なので、数学的には有理数である。IEEE 標準の**浮動小数点数から有理数への変換**を考える。倍精度浮動小数点数 a を、 e を指数、 d_i を 0 または 1 として 2 進数で次のように表す。

$$a = \left(\pm \sum_{i=0}^{52} d_i 2^{-i} \right) \cdot 2^e = A \cdot 2^e \quad (1)$$

$B = A \cdot 2^{52}$ は 2^{54} よりも小さい整数なので 17 桁以下で表すことができる。 B を用いると式 (1) の数は、 $a = B \cdot 2^{e-52}$ である。 $C = 52 - e$ とおくと $a = B \div 2^C$ 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。

$|a| \leq 0.5$ の場合、分母、分子は 10 進数では 17 桁に収まる。IEEE 標準の倍精度浮動小数点数の場合には、 $e_{max} = 1023$ なので、絶対値の大きい数の分母

は $2^{1023-52} = 2 \cdot 10^{292}$ となる。

例えば 2 進数で循環小数になる 0.4 とそれに 1 ビットの誤差を加えた $0.4 \underbrace{0000000000000000}_{14} 1$ はそれぞれ次のようになる。

$$0.4 = \frac{3,602,879,701,896,397}{9,007,199,254,740,992} \quad (2)$$

$$0.400 \dots 01 = \frac{7,205,759,403,792,795}{18,014,398,509,481,984} \quad (3)$$

1 ビットの誤差を含む数の分母は、0.4 の分母の 2 倍であり、分子は 0.4 の分子の 2 倍プラス 1 である。分母、分子は 16 または 17 桁の整数である。倍精度浮動小数点数が、特別に絶対値の大きな数でなければ、有理数では分母、分子がこの程度の桁数の整数に収まる。

1.1 有理数計算による正確な計算

十進 BASIC の有理数モードでは分母、分子を 2000 桁の整数まで扱える。これらのモードで、連立 1 次方程式をガウス消去法で解くと、有理数計算の特長が確認できる。ヒルベルト行列

$$a_{ij} = \frac{1}{i+j-1} \quad (4)$$

を係数行列として、解ベクトルの全要素が 1 になる問題を解くと、2 進数や 10 進 15 桁のモードによる浮動小数点計算では、演算のたびに生ずる丸め誤差の影響で、 $n = 12$ の問題に対して 30% の、 $n = 13$ では 100% 以上の誤差が現れる。これを有理数モードで実行すると、正確な解が得られる。すなわち、**計算機パワーとメモリ容量が許せば、計算が有理数の範囲で実行可能なアルゴリズムは、浮動小数点演算に伴う丸め誤差の影響を完全に排除できる。**

1.2 有理数の丸め

浮動小数点計算では、(機械語レベルの) 演算命令がその数値を丸める。したがって数値を表現するためのメモリは増加しないが精度は失われる (丸め誤差を含む)。反対に有理数計算では、桁数を増加させながら計算するので、精度は失われない (丸め誤差は入らない) が、数値を格納するためのメモリは増大する。この両極端の性質は、非線形方程式を同じアルゴリズムで解くと理解しやすい。非線形問題を解くための収束反復の計算で、浮動小数点計算のアルゴリズムをそのまま有理数計算に用いると、不必要に桁数の多い計算を行い、計算が続けられなくな

¹ 芝浦工業大学
Sibaura Institute of Technology, Minuma, Saitama
337-8570, Japan

^{*1} 有理数計算で得られた値を浮動小数点数に丸めて、10 進数で計算を続ける場合は、有理数計算の結果をファイルに書き出し、10 進モードのプログラムで読み込む必要がある。

ることがある*2.

非線形問題の例として、2分法で多項式 $f(x)$ の零点を求める問題をあげる. 解が1つだけ存在する区間 (a, b) が得られたとき、区間の中点 $c = \frac{a+b}{2}$ の座標値の桁数は、浮動小数点計算では区間端の a や b と変わらないが、有理数計算では増加してゆく. 4次多項式

$$f(x) = x^4 - 10x^3 + 15x^2 - 7x + 1 \quad (5)$$

は、 $f(x) = (x-1)(x^3 - 9x^2 + 6x - 1)$ と因数分解できるので、 $x = 1$ の解をもつ. 区間 $(a, b) = (0.9, 1.2)$ で2分法反復を開始すると、反復解は $\frac{21}{20}, \frac{39}{40}, \frac{81}{80}, \frac{159}{160}, \frac{321}{320}, \dots$ と推移する*3. 分母の偶数と分子の奇数は変わらないので、反復解は1に到達しない. 反復のたびに中点 c の分母は2倍になり、 k 回の反復によって $10 \cdot 2^k$ になる*4. ここでは c は区間内の数値でさえあればよく、中点の位置を高精度に求める必要はない. したがって、 c の近傍で分母分子の桁数が少ない適当な有理数に置き換え (丸め) たい.

有理数計算でも、有理数と倍精度浮動小数点を混在させられれば、型変換によって有理数を倍精度に丸め、改めて有理数に変換すれば、浮動小数点計算と同様の桁数の c を選ぶことができる. しかし倍精度の桁数では、高精度計算には不十分なため、有理数を、精度指定して別の有理数に丸める一般的な関数を用意した. この有理数として、元の有理数に近い数の最良近似分数 (best approximating fraction) を使う.

関数 $\text{ratround}(x, m)$ は、有理数 x をスケール倍した大きな数を2乗し、その数の整数部分の平方根を連分数展開することで最良近似分数を得て、それをスケールで割った値を返す. 引数に与える m は、変換された有理数と x との相対誤差が 10^{-m} 以下を指定する [6]. この関数を用いることにより、区間幅が広い間は、 c の座標値を必要以上高い精度で求めることなく反復計算する.

*2 有理数計算プログラミングは、浮動小数点計算プログラミングに比較すると、一般的には、桁溢れを意識しないので簡単である (例えば、LU 分解で行列式の値も計算するには、浮動小数点演算では桁溢れを考慮するため、やや厄介なプログラミングが要求され、大学初年度ではなかなか難しい).

*3 最初の反復解は $x_1 = \left(\frac{9}{10} + \frac{6}{5}\right) \div 2 = \frac{21}{20}$ となる. これは1より大きいので、 $x_2 = \left(\frac{9}{10} + \frac{21}{20}\right) \div 2 = \frac{39}{40}$ となる.

*4 分母と分子は互いに素なので、約分されることもない.

2. 平方根の陰的定式化と計算可能性

直交行列 Q の定義は $Q^T Q = I$ である. $m \times n$ の行列 A から Q を生成するシュミットの直交化アルゴリズムを示す (直交化された行列 Q の第 j 列目の列ベクトルを q_j とする).

$$\begin{aligned} & \text{for } j = 1, n \\ & \quad p_j = a_j - \sum_{k=1}^{j-1} (q_k^T a_j) q_k \\ & \quad q_j = \frac{1}{\|p_j\|} p_j \\ & \text{loop} \end{aligned}$$

ここに $\|p_j\| = \sqrt{p_j^T p_j}$ なので、このアルゴリズムを素朴にプログラミングすると、開平演算が必要である. 直交分解の応用として、線形最小2乗法の問題 $Ax \approx b$ を考える. $m \times n$ の長方形行列 A を、ここでは $n = 3$ として示す. A を QR 分解し、

$$A = QR = (q_1 q_2 q_3) \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{pmatrix} \quad (6)$$

両辺に Q^T を掛けることで $Q^T Ax = Q^T b$ となる. 左辺の係数行列は上三角行列 R になり、後進代入で解を得られる. 一見、開平演算必須のように見えるが⁵, $q_k = \frac{1}{\sqrt{p_k^T p_k}} p_k$ の係数 $\frac{1}{\sqrt{p_k^T p_k}} = \frac{1}{r_{kk}}$ を分離して書くと、直交化アルゴリズムの $(q_k^T a_j) q_k$ は、

$$(q_k^T a_j) q_k = \frac{1}{r_{kk}} (p_k^T a_j) \frac{1}{r_{kk}} p_k = \frac{1}{r_{kk}^2} (p_k^T a_j) p_k$$

と計算でき、開平演算は不要になる. 線形最小2乗解を求める場合は、直交行列 Q を経由する必要はなく、両辺に掛ける行列の各列が、互いに直交していれば十分である*5. つまり、 n 以下の任意の i と j で $i \neq j$ に対して、 $p_i^T p_j = 0$ が成立していれば十分である. このとき、式 (6) は次のように書き表せる.

$$A = PR = (p_1 p_2 p_3) \begin{pmatrix} 1 & \frac{r_{12}}{r_{11}} & \frac{r_{13}}{r_{11}} \\ 0 & 1 & \frac{r_{23}}{r_{11}} \\ 0 & 0 & 1 \end{pmatrix} \quad (7)$$

この例のように、ベクトル q の要素 q_i に平方根がかかり、 $q_i = \sqrt{r} p_i$ のように表されるとき、無理数 $\sqrt{r}$ と有理数 p_i を分離して、平方根は2乗した r を格納し、後続の計算でその2乗 $r p_i^2$ を正確に計算す

*5 $Ax \approx b$ の問題では、 $P^T Ax = P^T b$ となるので、右辺を割る操作が入る.

る方法を**平方根の陰的定式化** (implicit square root formulation) と呼ぶことにする*6。

この方法は、平方根を使用するコレスキー分解 $A = CC^T$ に対する、修正コレスキー分解 LDL^T と同等と考えられる*7。修正コレスキー分解は連立1次方程式の解法として有効であるが、係数行列が正定値の場合の $Ax = \lambda Bx$ の形の一般化固有値問題で、 $B = CC^T$ とコレスキー分解して、 $(C^{-1}AC^{-T})(C^Tx) = \lambda(C^Tx)$ の形の通常の固有値問題に変換する場合には利用できない。このように、平方根の陰的な処理が有効か無効かは、コレスキー分解をどのように使うかの応用に依存する。

本稿で「できる」か「できない」かの議論は、プログラムが扱う**計算式の形を変えないで計算できる範囲**で考える。数式処理プログラムを用いて、計算式を生成し、展開済みの計算式に数値を代入して計算を進める範囲には拡大しないものとする。つまり、有理数計算で計算する数式は、通常の数値計算と同様に考えるが、異なる点は、扱う数値の型が浮動小数点数ではなく有理数（そのために必要となるメモリー領域は動的に拡大する）になる。

3. 実対称行列の有理数計算による固有値の高精度計算（精度保証付き）

固有値問題 $Ax = \lambda x$ は右辺に未知数の積があるので非線形問題である。したがって計算方法は、要求された解の精度に固有値が収束するまで反復する。解法は、収束反復のループで係数行列に対して直接演算を行う方法と、相似変換によって3重対角行列に変換し、収束反復のループでは3重対角行列に対して演算を行う方法に分けられる。前者には、ヤコビ法、行列式探索法、べき乗法、逆べき乗法、シフト付き逆べき乗法、サブスペース法などがあり、後者の3重対角化には、ハウスホルダー法（鏡映変換）、ギブンス法（面内回転）、ランチョス法などがある。

有理数計算では LDL^T 分解を正確に計算すること

ができるので、行列 $A - \omega I$ を LDL^T 分解し、行列式の値 $\det(A - \omega I)$ と、対角行列 D の項の符号から A の ω よりも小さい固有値の数を知ることができる。これらの計算はすべて正確な有理数計算の範囲内で可能なので、この過程を、要求された解の精度に固有値が収束するまでの反復すれば、目的を達することができる。ただしこの方法は、**行列 A の固有値が十分に分離して n 個存在する場合**には稼働するが、固有値が重根を持つ、あるいは非常に近い固有値をもつ場合にはうまく稼働しない。

一方、3重対角化を経由する方法の多くは、3重対角行列への消去の過程に三角関数や開平演算があるので、有理数の範囲では、正確な3重対角化はできないと考えられがちである。

本章で、**平方根の陰的定式化による正確な3重対角化**を行い、対称3重対角行列に対する2分法とニュートン法反復により、任意の高精度で固有値を求める方法を述べる。3重対角化を経由する利点は、重根をもつ問題に対処できることである。はじめに、多重度のある問題で、3重対角行列が数学的にどうなるかを、Wilkinsonの教科書より引用する。

対称行列が多重度 k の固有値を持つ場合、ギブンス法やハウスホルダー法によって得られる3重対角行列は、少なくとも $k-1$ 個の零副対角要素をもつ [8], p. 300.

この引用は、正確な3重対角化ができれば、縮退の問題は自動的に解決されることを示唆している*8。しかしギブンス法やハウスホルダー法は有理数計算では実現できないので、これを計算機で確かめるには、数式処理を利用するなどの方法しか考えられなかった。

対称行列の3重対角化には、ランチョス法があるが、この方法も、そのままでは開平演算を避けられない。本章でははじめに、前述した直交分解に現れる開平演算を陰的に処理する方法が、ランチョス法に適用できない理由を示す。次に正確な3重対角化を、共役勾配法 (conjugate gradient: CG 法) から再構成する方法を示す。

3.1 ランチョス法

対称行列 A を3重対角化 $Q^TAQ = T$ する正規直交行列 Q が存在する。 $AQ = QT$ を列ごとに比較

*6 シュミットの直交化アルゴリズムを浮動小数点演算によってプログラミングする場合、古典的な計算順序による Classical Gram-Schmidt (CGS) 法、計算誤差を少なく抑える目的から計算順序を変更した Modified Gram-Schmidt (MGS) 法、計算速度の観点から CGS を2回実行するダブル CGS 法の3つが存在する [7]。有理数演算では丸め誤差が現れないので、CGS 法と MGS 法に差異はなく、ダブル CGS 法の意味はない。

*7 LDL^T 分解の対角項 $D = \text{diag}(d_i)$ を、 $s_i = \sqrt{d_i}$ として、 s_i を対角項にもつ対角行列を S とすると、 $LDL^T = LSSL^T$ なので、 $C = LS$ である。 LDL^T 分解は、 CC^T 分解の平方根の陰的定式化と言える。

*8 縮退があるかもしれない対称行列の固有値を求める問題は複雑な問題であるが、正確な3重対角行列へ変換する問題と、3重対角行列の固有値を求める問題に分割できると、それぞれは元の問題よりも難易度の低い2つの問題と考えられる。

する.

$$A[\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n] \\ = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n] \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & & & \\ & & \ddots & & \\ & & & \ddots & \beta_{n-1} \\ & & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

第1列目は $A\mathbf{q}_1 = \alpha_1\mathbf{q}_1 + \beta_1\mathbf{q}_2$ となるので, この式の左から $\mathbf{q}_1^T$ を掛け, 直交条件 $\mathbf{q}_1^T\mathbf{q}_2 = 0$ より $\alpha_1 = \mathbf{q}_1^T A\mathbf{q}_1$ を得る. ここで $\mathbf{r}_1 = (A - \alpha_1 I)\mathbf{q}_1$ がゼロベクトルでなければ, そのノルム $\beta_1 = \|\mathbf{r}_1\|$ を求めて正規化 $\mathbf{q}_2 = \frac{\mathbf{r}_1}{\beta_1}$ する.

2列目以降は第 j 列目について記述すると,

$$A\mathbf{q}_j = \beta_{j-1}\mathbf{q}_{j-1} + \alpha_j\mathbf{q}_j + \beta_j\mathbf{q}_{j+1} \quad (8)$$

なので, 直交性より $\alpha_j = \mathbf{q}_j^T A\mathbf{q}_j$ が得られる. $\mathbf{q}_{j+1}$ は

$$\mathbf{r}_j = (A - \alpha_j I)\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1} \quad (9)$$

がゼロベクトルでなければ, ノルム $\beta_j = \|\mathbf{r}_j\|$ を求めて正規化 $\mathbf{q}_{j+1} = \frac{\mathbf{r}_j}{\beta_j}$ する. ここでは $\mathbf{q}_j$ の直交条件から α_j が, $\mathbf{q}_j$ の正規化条件から β_j が決められる. このアルゴリズムでは, 式 (8) が, $A\mathbf{q}_j = \sqrt{s_{j-1}}\mathbf{q}_{j-1} + \alpha_j\mathbf{q}_j + \sqrt{s_j}\mathbf{q}_{j+1}$ の3項の和で, 平方根がそのうちの2項にかかっている. これはシュミットの直交化の $\mathbf{q} = s\mathbf{p}$ のような係数倍の関係ではない. したがって, 計算式の項数を増加させずに計算を進めるには, 平方根を陽に求めなければならず, ここで計算誤差が入る.

3.2 CG法を経由する3重対角化

対称正定値行列 A を係数行列とする線形方程式 $A\mathbf{x} = \mathbf{b}$ の解法に用いられるCG法は, その反復過程はランチョス法と同等である. CG法は有理数計算で正確に計算することが可能である [9]. 各反復で得られる3つのスカラー量を保存することで, ランチョス法による3重対角化を再構成できる [10], p. 528.

$$\gamma_k = \mathbf{p}_k^T A\mathbf{p}_k, \quad \sqrt{\delta_k} = \|\mathbf{r}_k\|^{-1}, \quad \beta_k \quad (10)$$

ランチョスベクトル $\mathbf{q}_j$ は, 残差ベクトル $\mathbf{r}_k$ に対応する*9. なお, 本節の β_k は, 前節のランチョス法の β_k とは異なる.

*9 ランチョス法の $\mathbf{q}_1$ をCG法の $\mathbf{r}_0$ と置いた場合が, 数学的に同一の3重対角化になる.

対角項が1で副対角項が $-\beta_k$ の2重対角行列を

$$B_k = \begin{pmatrix} 1 & -\beta_2 & & \\ & 1 & \ddots & \\ & & \ddots & -\beta_k \\ & & & 1 \end{pmatrix}, \quad (11)$$

また, 残差ノルムを対角項とする対角行列を Δ とすると, 3重対角行列 T_k は次のように得られる.

$$T_k = \Delta^{-1} B_k^T P_k^T A P_k B_k \Delta^{-1} \\ = \Delta^{-1} B_k^T \begin{pmatrix} \gamma_1 & & \\ & \ddots & \\ & & \gamma_k \end{pmatrix} B_k \Delta^{-1} \quad (12)$$

ここで内側の行列の積 $B_k^T P_k^T A P_k B_k$ は対称3重対角行列になるので, その対角項を d_i , 副対角項を s_i とおくと, T_k は次のように変形される.

$$T_k = \Delta^{-1} \begin{pmatrix} d_1 & s_1 & & \\ s_1 & d_2 & s_2 & \\ & s_2 & \ddots & \\ & & & d_k \end{pmatrix} \Delta^{-1} \\ = \begin{pmatrix} \delta_1 d_1 & \sqrt{\delta_1 \delta_2} s_1 & & \\ \sqrt{\delta_1 \delta_2} s_1 & \delta_2 d_2 & \sqrt{\delta_2 \delta_3} s_2 & \\ & \sqrt{\delta_2 \delta_3} s_2 & \ddots & \\ & & & \delta_k d_k \end{pmatrix} \quad (13)$$

この形を**陰的な3重対角化**と呼ぶ. 「陰的」は, 残差ノルムの逆数の2乗 δ_k をベクトルとして別途保持することを意味する*10.

対称3重対角行列 T_k の行列式の計算では, 非対角項は, 対称の位置にある項同士の積 $(\sqrt{\delta_i \delta_{i+1}} s_i)^2$ によって計算されるので, 平方根を陽に計算する必要がない. したがって, 誤差を含まない3重対角化を(陰的ではあるが)実現できる. これを用いて, 任意の有理数のシフト点 ω に対する行列式の値 $|T - \omega I| = |A - \omega I|$ を正確に求めることができる. また負の固有値の数も, スツルム列の性質から得られる.

*10 ランチョス法ではランチョスベクトル $\mathbf{q}_k$ を求めるために, 残差ベクトル $\mathbf{r}_k$ を正規化したので, 開平演算が避けられなかった. CG法では残差ベクトル $\mathbf{r}_k$ は, 2回の反復の探索方向ベクトルの1次結合 $\mathbf{r}_{k-1} = \mathbf{p}_k - \beta_k \mathbf{p}_{k-1}$ で計算される. ここに, β_k は, 残差ノルムの2乗の減少率である. このため, 反復は残差ノルムそのものではなく, その2乗 $\mathbf{r}_k^T \mathbf{r}_k$ で計算を進めることができる.

4. 有理数計算プログラム

本章で有理数計算のプログラムを解説する。はじめに下位の階層の多桁整数演算と有理数演算について説明する。次に、連立1次方程式の直接解法を例に、有理数を形成する分母、分子の桁数の増加と計算時間の振舞いについて解説をする。その次に、対称行列の固有値問題を計算するプログラムを解説する。最後に、対称行列の固有値問題を、多重度の高い問題を例に、1ビットの誤差がある場合を含めて示す。

4.1 有理数演算

多桁の整数は、基数を 2^{32} として 32 ビット整数型の配列に格納する。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 31 万桁)。

四則演算は筆算で行うのと同様のアルゴリズムで計算するが、乗算は桁数が増えると、FFT を用いることで高速化した。FFT は基数 2 のみを使用し、32 ビット整数を、基数が 2^{24} の単精度浮動小数点数に変換し、多倍長演算を、各桁では倍精度で計算する。通常の乗算と FFT 乗算の境界は、 2^{24} 進数で $2^8 = 256$ 桁 (10 進数で約 1800 桁) としている。

有理数は、多桁の整数を分母、分子に用い、有理数同士の四則演算のたびに、2 進 GCD アルゴリズムで GCD を求め、既約分数とする。これらの演算は教科書 [2][3] の多倍長演算と有理数演算にほぼ忠実にを行った [9]。

プログラムは、変数の型として、多桁整数 (longint) 型と有理数 (rational) 型を定義して、数値計算アルゴリズムはこれらの型の変数に対して記述できるようにした。有理数計算や多倍長計算などの高精度計算は、浮動小数点計算で近似解が得られ、その精度に疑問が持たれる場合に、問題分析を目的に行われることが多い。したがって、通常の倍精度、あるいは 4 倍精度計算との親和性が重要と考えた。そこで、C++ のオペレータオーバーロードの機能を用いて、四則演算を、longint 型や rational 型の変数に対して記述できるようにした。

4.2 有理数計算による対称行列の LDL 分解

有理数計算では、有理数の四則演算回数が、桁数を通して下位の階層で稼働する多桁演算の計算量に関係するため、浮動小数点計算の計算時間の振舞とは大きく異なる。この事実を、対称行列の LDL^T 分解を例に示

す。行列 A が対称の場合、上三角行列の要素 u_{ij} と、下三角行列の要素 l_{ij} の間では、 $l_{ji} = u_{ij}/u_{ii}$ の関係がある。この関係を行列によって表すと、 $A = LDL^T$ となる (D は対角行列で、 $U = DL^T$)。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。

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} t_{ki} u_{kj}, \quad (i \leq j) \quad (14)$$

$$t_{ij} = \frac{u_{ij}}{u_{ii}} \quad (i < j) \quad (15)$$

プログラムは付録に示したが、通常の Fortran や C 言語による数値計算プログラムとよく似ている。違いは、**変数が有理数型**にある。

係数行列を、次数 n のフランク行列^{*11}

$$a_{ij} = n - \max(i, j) + 1 \quad (16)$$

式 (4) のヒルベルト行列、浮動小数点数に丸めたヒルベルト行列、 $|a_{ij}| \leq 0.5$ の倍精度浮動小数点乱数の 4 つの場合の、次数 $n = 100$ での比較を示す。

係数行列	時間 (秒)
フランク行列	0.7
ヒルベルト行列	3.8
浮動小数点ヒルベルト	92.6
$ a_{ij} \leq 0.5$ の乱数	989.3

計算機は Intel の Core i5 (2.67GHz) を搭載したノートパソコンで、1 スレッドのみを使用した。

フランク行列は、要素が整数である。また、行列式は 1 で、 D の対角項 d_i は、1 行目以外は $\frac{n-i+1}{n-i+2}$ になる。したがって分解計算の過程で、行列要素の有理数を構成する分母、分子の桁数がほとんど増加しない特殊な行列である。桁数の増加がほとんどないため、計算時間は $O(n^3)$ である。

ヒルベルト行列は、フランク行列と同様の整数が分母に存在する。分解された行列 D の対角項 d_i は、1 行目以外は分子が 1 か -1 になる。一方、分母の桁数は、1 行の分解において 1 桁または 2 桁ずつ増加する^{*12}。したがって、フランク行列の場合よりも多桁

^{*11} $n = 5$ の行列と LDL^T 分解の L を示す。

$$\begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 \\ 3 & 3 & 1 & 0 & 0 \\ 2 & 4 & 2 & 1 & 0 \\ 1 & 5 & 4 & 3 & 1 \end{pmatrix}, D$$

 は $\text{diag}(5, \frac{4}{5}, \frac{3}{4}, \frac{2}{3}, \frac{1}{2})$ である。

^{*12} 行列式 $|A|$ は $\prod_{i=1}^n d_i$ なので、ヒルベルト行列は非常に小さな行列式の値をもち、これが行列の条件数を大きく (ill-condition に) する。

整数演算の計算量は多く、計算時間は $O(n^4)$ になる。

ヒルベルト行列の行列要素を倍精度浮動小数点数に丸めてから有理数変換すると、行列要素の桁数が多くなっているため、本来のヒルベルト行列よりも計算時間ははるかに長くなる。

一般の行列に近い、 $|a_{ij}| \leq 0.5$ の乱数を用いると、分母、分子の桁数の増加はさらに大きくなり、 d_i の分母と分子の桁数の和は、1 行の分解において約 33 桁ずつ増加し、 d_{300} では 10,000 桁を超える。三角行列の要素は、式 (14) の右辺にある内積で生成される。その要素の分母と分子の桁数の和が線形に増加する理由は、桁数が行位置に対して線形に増加する 2 本のベクトルの内積によって作られる有理数の桁数が、ベクトル長に比例して増加するからと考えられる。このため、計算時間は $O(n^4)$ と $O(n^5)$ の間にある。 $O(n^5)$ よりも小さくなる理由は、FFT によるものと考えられる。

このように、同じ次数の行列の LDL^T 分解でも、計算時間の振舞いは浮動小数点演算の場合とは大きく異なる。データによって演算桁数が大きく異なるため、 LDL^T 分解プログラムから読み取れる有理数演算回数からでは、計算時間を予測することができない。次数 100 で比較すると 1000 倍以上の差が表れ、計算時間では $O(n^3)$ から $O(n^5)$ に変化する。

4.3 対称行列の固有値計算

対称行列の固有値問題を計算するプログラムを、正確な 3 重対角化、区間の絞り込み、ニュートン法による加速によって精度保証された桁数を求めるプログラムについて述べる。

4.3.1 陰的な 3 重対角化

前節の式 (13) の 3 重対角行列の d_i, s_i, δ_i を求めるプログラムを示す。ループ部分は、解ベクトルを計算しない CG 法反復である。

```

k = 0;  $\mathbf{r}_0$  = initialize;  $\delta_1 = \|\mathbf{r}_0\|^2$ 
 $\beta_1 = 1$ ;  $d_1 = \frac{\gamma_1}{\delta_1}$ ;  $s_1 = -\gamma_1$ 
do
  k = k + 1
  if(k = 1)
     $\mathbf{p}_1 = \mathbf{r}_0$ 
  else
     $\beta_k = \frac{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{r}_{k-2}^T \mathbf{r}_{k-2}}$ 
     $\mathbf{p}_k = \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}$ 
  end
   $\alpha_k = \frac{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{p}_k^T A \mathbf{p}_k}$ 
   $\gamma_k = \mathbf{p}_k^T A \mathbf{p}_k$ 

```

```

if(k > 1)
   $d_k = \frac{\gamma_k + \gamma_{k-1} \beta_{k-1}^2}{\delta_k}$ 
   $s_k = -\gamma_k \beta_k$ 
end
 $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k A \mathbf{p}_k$ 
 $\delta_{k+1} = \|\mathbf{r}_k\|^2$ 
if( $\delta_{k+1} = 0$ ) exit do
loop

```

係数行列が縮退していると CG 法反復は n 回よりも少ない反復で終了するので、この場合の 3 重対角行列の次数は、もとの対称行列の次数よりも小さい。

次数 n の対称 3 重対角行列 T を次のように記述したとき ($b_i \neq 0$ とする),

$$T = \begin{pmatrix} a_1 & b_2 & & \\ b_2 & a_2 & b_3 & \\ & b_3 & a_3 & \\ & & & \ddots \end{pmatrix}$$

スツルム列は、 T_i を T の次数 i の主小行列としたとき、 $T_i - \omega I$ の行列式を求める多項式 $p_i(\omega) = \det(T_i - \omega I)$ を、 i が 1 から n まで漸化式の形で計算したときの $p_i(\omega)$ の符号変化の回数によって得られる。

```

 $p_0(\omega) = 1$ 
 $p_1(\omega) = a_1 - \omega$ 
do i = 2, n
   $p_i(\omega) = (a_i - \omega)p_{i-1}(\omega) - b_i^2 p_{i-2}(\omega)$ 
loop

```

副対角項 b_i^2 を、 $\delta_i \delta_{i+1} s_i^2$ によって計算する。

4.3.2 区間の絞り込み

固有値が存在する区間 $[\omega_l, \omega_u]$ が与えられた場合、次のループでその区間を縮小する。

```

 $d\omega = \frac{\omega_u - \omega_l}{n}$ 
k = 0;  $d\omega_l = \omega_l$ ;  $\omega = \omega_l$ 
do
   $d\omega_u = d\omega_l + d\omega$ 
  call TDCdet(A,  $d\omega_u$ , f, no)
  if(no - k = 1) then
    call Bisect(A,  $d\omega_l$ ,  $d\omega_u$ ,  $\omega$ )
    k = k + 1
     $d\omega_l = d\omega_u$ 
     $d\omega = \frac{\omega_u - \omega_l}{n}$ 
  else if(no - k = 0) then
     $d\omega_l = d\omega_u$ 
  else
     $d\omega = \frac{d\omega}{2}$ 
  end if

```

```

if(k = n) then exit loop
loop

```

サブルーチン TDCdet は、引数に与えられた 3 重対角行列 T と ω から、行列式の値 $f(\omega) = |T - \omega I|$ を引数 f に、また ω よりも小さい固有値の数を、スツルム列の性質から得て、引数 no に返す。したがって、反復により区間 $(d\omega_l, d\omega_u)$ に固有値が 1 つだけ存在するような区間を得られる。浮動小数点演算では、隣接する固有値が非常に近いとき、計算誤差の影響で絞り込みを誤る場合があるが、有理数演算では**重根がなければ**、この方法で確実に絞り込みが達成できる。

区間 (a, b) 内に固有値が 1 つに絞り込まれた時点では、サブルーチン Bisect によって、その区間の固有値を、2 分法により指定された精度で求める。

```

SUB Bisect(A, a, b, ω)
call TDCdet(A, a, fa, no)
call TDCdet(A, b, fb, no)
do
  c = ratround(a - (fa * (b - a)) / (fb - fa), ord)
  call TDCdet(A, c, fc, no)
  if(fc = 0) then exit loop
  if(fa * fc < 0) then
    b = c; fb = fc
  else
    a = c; fa = fc
  end if
  if(|fa - fb| < ε) then exit loop
loop
ω = c

```

2 分法は、区間の中点を用いるのではなく、2 点 $(a, f(a))$ と $(b, f(b))$ を結ぶ直線と x 軸との交点の x 座標値

$$c = a - \frac{f(a) * (b - a)}{f(b) - f(a)} \quad (17)$$

を使用した。この方法では、区間端の数値の分母、分子の桁数を、中点を用いる場合よりも急速に増加させる。したがって `ratround` による丸めの効果は大きい。

4.3.3 ニュートン法による収束の加速

2 分法の収束は遅いので、2 次収束するニュートン法を用いる。 ω_0 から ω_n までの $n + 1$ 点で行列式の値 $\det(A - \omega_i I)$ を求め、特性多項式 (characteristic polynomial) の $n + 1$ 個の多項式の係数を、 $n + 1$ 次元の連立 1 次方程式を解くことで得られる。特性多項式の係数は、デザイン行列を係数行列とする次の

連立 1 次方程式を解くことで得られる。 $n = 3$ の場合の係数を求める連立 1 次方程式を示す。

$$\begin{pmatrix} 27 & 9 & 3 & 1 \\ 8 & 4 & 2 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \det(A - 3I) \\ \det(A - 2I) \\ \det(A - I) \\ \det(A) \end{pmatrix}$$

係数行列を消去する際、桁数が増加しないように、 ω_0 から ω_n には 0 から n を使用した。

特性多項式の係数 a_0 から a_n が得られれば、その導関数も得られるので、ニュートン法反復により、2 分法の収束を加速できる^{*13}。

$$f(\omega) = a_0 \omega^n + a_1 \omega^{n-1} + \dots + a_n = 0$$

ニュートン法で反復解が区間外に出た場合は、2 分法に戻す。

2 分法だけでは、有理数を `ratround` で丸めても、解が有理数の場合でも正解に収束することはほとんどないが、ニュートン法では、解が有理数の場合には、正解に収束することがある。例えば n が 3 の倍数プラス 1 の場合のフランク行列の固有値の 1 つは 1 である^{*14}。 $n = 4$ の場合は、特性多項式が式 (5) になる。ニュートン法反復では、収束状況に応じた丸め (`ratround` の引数 m を、 $|a - b|$ と $|f(a) - f(b)|$ に連動) を行う。

4.3.4 固有ベクトルの計算

固有ベクトルの計算は、高精度で固有値が求まっているので、逆べき乗法の反復

$$\mathbf{x}_j = (A - \lambda_i I)^{-1} \mathbf{x}_{j-1} \quad (18)$$

により、 λ_i に対応する固有ベクトルを得ることができる。重根に対応する固有ベクトルも、初期ベクトルの変更と直交化により得られる。

固有値が高い精度で求まっている場合、3 重対角行列を捻り三角分解し、高精度の固有値を用いて、3 重対角行列の固有ベクトルを求めることができる [11]。これらの計算は、固有値が有理数ではないので、通常の浮動小数点演算で行う。

現実の浮動小数点計算で、3 重対角化ができていれば、3 重対角行列 T を正確な行列と考えて、それから Wilkinson の後退誤差解析の考え方で導かれる対称行列 A' を考える [12]。陰的 3 重対角化は

^{*13} 組立て除法を 2 重に行うホーナー法を用いる。

^{*14} $n = 4$ の場合、ベクトル $\mathbf{x} = (-1, 0, 1, 1)^T$ が固有方程式 $A\mathbf{x} = 1 \cdot \mathbf{x}$ を満たす。

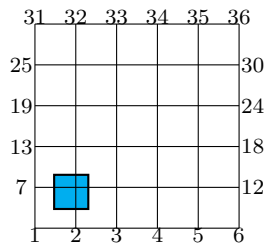


図 1 四角柱の内部の温度分布

$$T = \Delta^{-1} B_k^T P_k^T A' P_k B_k \Delta^{-1}$$

であるが、 A' は想定するだけで、陽に求める必要はない。 A' は、元の行列 A とは異なるが、 T は A' に対する正確な 3 重対角行列なので、 T から有理数計算によって、 A' の高精度な固有値を求めることができる。このようにして得られた固有値は、必要な精度だけ限りなく高精度計算できるので、捻り三角分解も有理数計算すれば、固有ベクトルを求める際、いっさい直交化を行うことなく計算を進められる可能性がある。最終的に得られる固有値は、 A' に対する精度保証された固有値なので、元の浮動小数点計算で問題が生じた場合の解析に、3 重対角化が問題なのか、固有値を求めるところに問題があるのかの判定に有効になる。また、浮動小数点演算によるシュミットの直交化は、並列性の観点からは難題なので、3 重対角行列に対する高精度な固有値を並列計算できれば、高速化の観点からも期待される。

4.4 有理数計算による対称行列の固有値の計算

多重度の高い固有値問題を、浮動小数点計算の場合と比較し、陰的定式化による有理数計算が、多重度を検出できることを示す。また、この問題に、意図的に 1 ビットの誤差を混入させた係数行列を作成し、有理数計算から何が得られるかを示す。

4.4.1 重複がある場合の固有値

一辺の長さが l の正方形の断面をもつ四角柱を $m \times m$ 分割した領域で、5 点差分で熱伝導問題を考える [7]。図 1 に 4×4 に分割した解析領域を示す。熱伝導係数を均一とすると、係数行列には対称性があるため、縮退がある。縮退の程度は、連立 1 次方程式の問題では境界条件（右辺ベクトル）に依存するが、3 重対角化の問題では初期ベクトルに依存する。熱伝導係数を 1 とした場合、 2×2 分割では係数行列は次のようになる。

表 1 固有値

i	λ_i	浮動小数点演算による λ_i
1	0.763932...	
2,3	1.763932...	...73, ...95
4	2.763932...	
5,6	3.	...0027, ...0044
7,8,9,10	4.	...9982, ...0044
11,12	5.	...9991, ...0000
13	5.236068...	
14,15	6.236068...	...863, ...889
16	7.236068...	

$$A = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}$$

4×4 の分割の問題の固有値は表 1 の第 2 列のように分布している。 $m \times m$ 分割では、4 が多重度 m の固有値である。

このような問題に対する 3 重対角化では、出発ベクトル（ランチョス法では q_1 ）の選び方によって縮退が異なる^{*15}。

- 出発ベクトルとして、すべての要素を 1 にすると、CG 法反復は 3 回で収束し、 $\lambda_1, \lambda_5, \lambda_{13}$ が得られる。
- 出発ベクトルの i 番目の要素を i にすると、CG 法反復は 6 回で収束し、すべての要素が 1 の場合に加えて $\lambda_2, \lambda_7, \lambda_{14}$ が得られる。
- 出発ベクトルの i 番目の要素を $\text{MOD}(i^2, 16)$ にすると、CG 法反復は 9 回で収束し、すべての異なる固有値が得られる^{*16}。

係数行列に縮退がある場合、固有値は重複するが、CG 法反復に基づく陰的な 3 重対角化を経由すると、3 重対角行列 T_k に変換する過程で、多重度の高い解は除外される。したがって、係数行列に直接演算を行う方法では正確に求めることが困難であった多重度の高い解も、単根となるので、安定して求めることができる。3 重対角化は有理数演算によっているので正確であり、2 分法に基づく反復は、解の上界と下界を挟んで区間縮小してゆくので、得られた固有値の精度は保証される。

^{*15} クリロフ部分空間は、初期ベクトルに依存する。

^{*16} このケースでは、有理数を構成する多桁整数の桁数は、配列長が 128 までで収まっており、計算時間も数 10 秒である。

4.4.2 係数行列に誤差を含ませた場合の固有値

表1の3列目に、倍精度浮動小数点計算による、固有値の差異の生じる下位の桁を示した。この表示は、最下位の桁が16桁を指定した^{*17}。2列目は、桁数を指定しないでprintコマンド表示されたもので、この範囲では多重度の高い固有値は、重根であるか、近似根であるかは分からない。

浮動小数点計算では、いろいろな理由で、本来、同じ値になるべき項が、丸め誤差の影響を受ける。特にコンパイラの最適化機能などのために計算順序が変更されて生じる誤差は、原因究明に労力を要することが多い^{*18}。問題の熱伝導係数を、0.1に変更すると、浮動小数点計算では、係数行列の非零項は循環小数になるので、誤差の影響を調べやすい。この問題に対して、 4×4 の問題を、誤差のない係数行列の対角項 $a_{ii} = 0.4$ の場合と、1ビットの誤差を A の1行1列成分 a_{11} だけに加えた係数行列を比較した。 a_{11} の値は、有理数計算では、式(2)と式(3)の違いになる。誤差なしの場合、固有値は表1の分布(ただし値は10分の1)になるが、誤差を載せた問題で20桁まで計算すると、 $i = 5, 6$ の重根が.29999999999999988048と.300000000000000015837に分離した。違いが現れるのが16桁目なので、倍精度浮動小数点計算ではこの違いを確かめられない^{*19}。これまでは、固有値問題の多重度を調べるには、数式処理によって、特性多項式を因数分解して、 $(\lambda - m)^k$ の項が現われれば、多重度 k と判定する必要があった^{*20}。本論文で述べた、平方根の陰的定式化を用いた有理数計算を用いることによって、この判定を数値計算の延長上で調べることができる。

数値計算の現場で、解析結果が疑問視されたとき、数値計算プログラムに、有理数計算を組み込むことができれば、問題分析(プログラムのバグか、コンパイラの計算順序によって生じる問題かなど)に適用できる。この目的では、FortranやC言語で書かれ

たプログラムから呼出し可能なライブラリの形で有理数計算が使用可能なことが望ましい。このライブラリは、並列化などの高速化技術を適用することで、数式処理プログラムよりもはるかに高速に計算できることが求められる。

5. まとめ

固有値問題 $Ax = \lambda x$ で、 A に縮退がある場合、あるいは縮退に近い状態である場合、これを浮動小数点演算で解くことは大変難しい。この困難は、浮動小数点演算では、丸め誤差をマシンイプシロンと比較して行わなくてはならないことに由来する。引用したように、「対称行列が多重度 k の固有値を持つ場合、...3重対角行列は、少なくとも $k-1$ 個の零副対角要素をもつ」が、四則演算を行う機械である計算機で、有理数演算を行っても、無理数を正確に計算することはできないので、これを数値計算によって確かめることはできなかった。本稿では、平方根の陰的定式化により、正確な3重対角化を得て、固有値を任意の高精度で求める方法を述べた。

有理数計算は、計算の原理は筆算の延長線上にあり素朴であるが、無理数を扱えないことと、計算機に対する負荷が大きいことにより、これまでは教育用の十進BASICなど、限られた実装しか存在しなかった。精度や証明を目的とするプログラミング環境には、多倍長演算、数式処理、区間演算に基礎をおく精度保証計算が用いられるが、有理数演算を用いると、計算誤差を気にすることのない素朴なプログラミングにより、数式が正しいか否かを調べることもできる。数学教育を目的とした場合、演習で扱う数値計算の例題は小規模なものに限られるので、有理数計算が可能なものは、積極的に有理数計算を使用していくべきである。

さらに規模の大きな有理数計算を行えば、浮動小数点演算で問題が生じた場合に、その原因究明に利用できる解析ツールとして期待できる。この例を、意図的に係数行列に1ビットの誤差を混入させた問題によって紹介した。固有値固有ベクトルの計算は、複数のプロセスに分解できる計算なので、そのうちの1つのプロセスだけでも有理数計算を適用できれば、問題解決の糸口を見つけられる。このアプローチを、固有ベクトルの計算の項で述べた。このような目的に有理数計算を生かすには、FortranやCで書かれた既存のプログラムに、有理数計算を組み込めることが必須である。

^{*17} 表の計算はプログラミング環境Rを用いて行い、下位の桁の表示はprintコマンドのdigitオプションによって16桁を指定した。

^{*18} 浮動小数点演算では、結合則が成立しない $(a+b)+c \neq a+(b+c)$ 。

^{*19} この計算では、倍精度浮動小数点計算で求めた固有値の近似値を使用した。十進BASICでは浮動小数点数と有理数を1つのプログラムで同時に使用できない(変数の型は「数」か「文字」に限定されるプログラミング言語の仕様による)。また、有理数モードで、桁数の多い整数を読み込むことも簡単にはできない。

^{*20} $(\lambda - m)^k$ の m は、係数行列の数値項から計算される数で、数式処理では式展開によって正確に求めることができる。

しかし、有理数計算のもっとも顕著な特徴である、1 ビットの誤差も見逃さない性格は、計算機の検収で行われるベンチマークにあると考える^{*21}。スーパーコンピュータの計算性能は、1976 年の Cray-1 の 160M flop/s から、2012 年のセコイア (Blue Gene/Q) の 16P flop/s の比較で 1 億倍になった [13]。このような大規模なシステムでは、納入時に検収のためのベンチマークを行うのが一般的である。現在よく行われる、擬似乱数で作成された係数行列に対する LU 分解では、行列サイズが 1000 万に達し、直接解法で得られた解の精度は上位数桁と言われている。これを 1 回だけ反復改良して、残差ベクトルのノルムが許容範囲に入るかどうか調べている^{*22}。このような方法では、下位の桁にわずかな計算機のエラーが現れても分からない。つまり、計算機システムに対する検収としては、エラーの検出能力が高くない。このような浮動小数点演算による連立 1 次方程式の直接解法よりも、有理数計算による直接解法や CG 法のほうが、エラーの検出能力は高いので、次世代ベンチマークとして有力である。特に、式 (16) とその脚注に示したフランク行列の LDL^T 分解のように、分解された三角行列の要素位置 i, j で、 d_i や l_{ij} に入るべき数値が分かっているならば、障害を検出したらただちに停止してダンプをとれるプログラムが作成できるので、問題の追及が短時間で行える。浮動小数点演算で分散メモリ型並列計算を行うと、メッセージの到着順序で丸め誤差の現れ方が変化するので再現性がなく、正確な値を決めることができない。

付 録

A.1 対称行列の LDL 分解プログラム

次に示すプログラムは、数値線形代数の対称行列の三角分解を、Fortran で記述した例を機械的に変換したものである [7], p. 128. Fortran では、2 次元配列を引数に渡す場合、先導次元を用いるが、C++ 言語で、変数型として rational 型、またそのマトリクスとベクトルを扱えるようにしたので、先導次元を引数に入れる必要はない。

```
void LDL(rational_matrix& a, int n){
    rational s,t;
```

^{*21} 「検収」は、納入品が要求仕様に合っていることを確かめるために行う検査であり、これをパスすると、その品の管理責任が、受注者から発注者に移る。

^{*22} 入らなければ、fail と表示されるだけで、そこから問題分析のきびしい追跡計算が始められる。

```
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;
}
```

参考文献

- [1] B. Sinharoy, R. N. Kalla, and et.al., IBM Power7 Multicore Server Processor, *IBM J. Res. Develop.*, 55(3):1/1-1/29, 2011.
- [2] D. Knuth, *The Art of Computer Programming, Volume 2*, Addison-Wesley, 1969. 渋谷正昭訳：準数値算法/乱数, サイエンス社, 1981.
- [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] <http://hp.vector.co.jp/authors/VA008683/>.
- [5] 飯高 茂, 松本幸夫 (他), **高等学校 数学 B**, 東京書籍, 2008.
- [6] Hikaru Samukawa, Rational Number Arithmetic Including Square Root Handling, In *31th JSST Annual Conference (JSST 2012) International Conference in Modeling and Simulation Technology*, 2012.
- [7] 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.
- [8] J. H. Wilkinson, *Algebraic Eigenvalue Problem*, Oxford University Press, 1965.
- [9] Hikaru Samukawa, A Study of Rational Number Arithmetic, In *30th JSST Annual Conference (JSST 2011) International Conference in Modeling and Simulation Technology*, 2011.
- [10] G. H. Golub and C. F. Van Loan, *Matrix Computations — third edition*, Johns Hopkins University Press, 1996.
- [11] I. S. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, 1989.
- [12] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Her Britannic Majesty's Stationery Office, 1963, 一松 信, 四条忠雄訳：丸め誤差解析, 培風館, 1974.

- [13] <https://www.llnl.gov/news/newsreleases/2012/Jun/NR-12-06-07.html>.