数理解析研究所講究録
第2054巻 2017年 1-13

安定化手法に基づく計算履歴法と LLLアルゴリズムへの適用

Application of the Log Method Based on Stabilization Techniques to the LLL Algorithm

永嶋 裕樹
HIROKI NAGASHIMA
東邦大学大学院 理学研究科
GRADUATE SCHOOL OF SCIENCE, TOHO UNIVERSITY *

白柳 潔
KIYOSHI SHIRAYANAGI
東邦大学 理学部
FACULTY OF SCIENCE, TOHO UNIVERSITY †

Abstract

LLL アルゴリズム (以下，アルゴリズム LLL) によりガウス既約基底を求めることは，最短ベクトル問題への基本的なアプローチである．最短ベクトル問題とは，n 個の線形独立なベクトル $a_1, \ldots, a_n \in \mathbb{K}^n$ が与えられ，これらのベクトルで生成される格子の中で，ユークリッドノルムで $\mathbf{0}$ 以外の最短なベクトルを求める問題である．また，ガウス既約基底はユークリッドの互除法に類似した操作で計算できる．しかし，その計算に浮動小数点近似を用いると，不安定性の問題がしばしば生じる．すなわち，入力行列の各係数の精度桁を上げても，その出力の行列は，正確なガウス既約基底に収束するとは限らない．本論文では，K. Shirayanagi と M. Sweedler が提案した安定化手法と安定化手法に基づく計算履歴法（以下，ISCZ 法）をアルゴリズム LLL に適用し，ガウス既約基底を浮動小数点で計算するための安定なアルゴリズムを実現する．さらに，いくつかの例に対して計算機実験を行い，そのアルゴリズムの安定性を実証する．これによって，安定化手法の有効性及び ISCZ 法の有効性を示す．また，白柳を中心に従来の ISCZ 法に対する New idea を提案した．この New idea の有効性についても示す．

1 はじめに

情報セキュリティ分野では，次世代暗号として有力な格子暗号が注目されている．格子暗号とは，格子の最短ベクトル問題など，格子理論における難しい問題を安全性の根拠とする暗号方式である．最短ベクトル問題は，格子暗号の安全性の根拠となっている．このとき，ガウス既約基底は最短ベクトル問題を考える上で重要な役割を果たす．また，数論との結びつきもある．具体的には，Simultaneous Diophantine approximation[10] を考える上で重要な役割を果たす．本論では，このガウス既約基底に焦点を当てる．

*6514005n@nc.toho-u.ac.jp
†kiyoshi.shirayanagi@nc.toho-u.ac.jp

ガウス既約基底は，ユークリッドの互除法に類似した方法によって計算できる．ところが，厳密計算でそれを行うと，特に行列のサイズが大きいときは，膨大な時間と記憶容量を必要とする．そこで，浮動小数点近似計算によって，その負荷が軽減できればよい．しかしながら，上述の計算法は厳密計算用のアルゴリズムであり，それをそのまま愚直に適用すると，入力値の精度桁がどれほど大きくなっても，出力が正確なガウス既約基底に収束するとは限らない．以下の英文は，参考文献 [11]p.89 の本文 (一部抜粋) であり，近似計算では本来辿るべきアルゴリズム LLL の経路を正しく辿れず，正確な出力から非常にかけ離れてしまう旨を警告している．

> If the Gram matrix does not necessarily have rational coefficients, the $\mu_{i,j}$ and ... must be represented approximately using floating point arithmetic. ... The main problem with this approach is that roundoff errors may prevent the final basis to be LLL reduced. In many cases, this is not really important since the basis is not far from being LLL reduced. It may happen however that the roundoff errors cause catastrophic divergence from the LLL algorithm, and consequently give a basis which is very far from being reduced in any sense. Hence we must be careful. ...

本論では，入力の精度を一定ずつ上げて実行を繰り返したとき，その各出力が正確なガウス既約基底に収束することを保証するような安定なアルゴリズムを与える．さらに，実際にそれを計算機で実行して，その有効性を示す．提案する方法は，単純に従来のアルゴリズムに浮動小数点計算を導入したものではなく，文献 [1] の安定化手法のテクニックを導入する．この安定化手法は，アルゴリズムがある条件を満たせば，そのアルゴリズムを新しいアルゴリズムに変換し，その新しいアルゴリズムをある値以上の精度桁で実行すれば，正確な出力に近い答を返すものである．本論では，アルゴリズム *LLL* がその適用条件を満たすことを確認し，実際に安定化手法及び ISCZ 法を適用する．

まず 2 章では，復習として安定化手法の概略を述べる．3 章では，安定化手法からヒントを得て提案された ISCZ 法の概略及び New Idea を述べる．4 章ではアルゴリズム *LLL* について述べる．最短ベクトル問題へアプローチするアルゴリズムはいくつか存在するが，本論では，安定性に重点を置き，不安定性の原因を明確に説明するなどの理由で，最も素朴と思われるアルゴリズムを選んだ．4.2 節では，そのアルゴリズムの不安定性の原因について詳しく述べる．5.1 節では，そのアルゴリズムに安定化手法と ISCZ 法を適用する．5.2 節では，いくつかの行列についての計算機実験を報告し，本提案手法の有効性を示す．

2 復習 (アルゴリズムの安定化手法)

文献 [1] に基づき，次のクラスのアルゴリズムについて安定化手法を説明する．

- 入力，中間ステップ，出力の任意のデータは，多変数多項式環 $R[x_1,\ldots,x_m]$ に属する．R は実数体の部分体である．
- アルゴリズムで行われる操作は，$R[x_1,\ldots,x_m]$ における加算，減算，乗算，剰余計算のみである．
- 述語の不連続点は，あるとすれば 0 のみである．

次に述語の不連続点について説明する．述語は多項式の集合から

$$\{\mathrm{TRUE},\mathrm{FALSE}\}$$

への写像である．述語 p が $f \in R[x_1,\ldots,x_m]$ で不連続であるとは，$f_i \to f$ となる $R[x_1,\ldots,x_m]$ の元の列 $\{f_i\}_i$ が存在して，$p(f_i) \not\to p(f)$ (すなわち，$p(f_i) \neq p(f)$) となることをいう．ここに，$f_i \to f$ とは f_i が

係数ごとに f に収束することを示す．係数の収束は普通の実数集合における収束である．従って，0 が述語 p の不連続点であるとは，0 多項式 (任意の係数が 0 である多項式) において述語 p が不連続であることである．

上述の 3 つの条件を満たすアルゴリズムを不連続点 0 の代数的アルゴリズムと定義する．多項式を扱う大抵の数式処理のアルゴリズムは，不連続点 0 の代数的アルゴリズムであるか，またはそれに変換できるものである．例えば，$X = 9?$ という述語は，$Y \leftarrow X - 9$ という変数置換えと $Y = 0?$ という不連続点 0 の述語に変換できる．

簡単のため，アルゴリズムに現れる操作は，多項式演算と剰余演算しかないとしているが，文献 [1] では平方根や微分などの他の多くの関数についても議論している．

$\mathcal{A}$ は不連続点 0 の代数的アルゴリズムであると仮定する．$\mathcal{A}$ は近似入力に対して，不安定となることがある．例えば，次のようなアルゴリズム REPEART_SUB を考える．

```
function REPEAT_SUB(())
    X ← 1
    while (X > 0)
        X ← X − 1/3
    endwhile
    return (X)
endfunction
```

アルゴリズム REPEAT_SUB は，入力は空 () で，最初に X に 1 を代入した後，$X > 0$ である間，X から 1/3 を減算し続け，最後に X を出力する．もちろん，REPEAT_SUB(()) は 0 の値を返す．ところが，任意の精度桁 k に対し，浮動小数近似 $(1/3)_k = 0.333\ldots3(k$ 桁) となってしまい，REPEAT_SUB$((\,)_k)$ は常に $-0.333\ldots32(k$ 桁) を返してしまう．これが不安定性である．言い換えれば，$(\,)_k \to ()$ (より正確には，$\forall k, (\,)_k = ()$) であるが，REPEAT_SUB$((\,)_k) \to -1/3 \neq$ REPEAT_SUB(()) である．REPEAT_SUB は，不連続点として $\{0\}$ を持つ述語 $X > 0$ を持っているので，不連続点 0 の代数的アルゴリズムである．この不連続点があるため，述語の評価 (条件分岐) が正しく行われず，アルゴリズムの正しい実行過程 (元のアルゴリズムを厳密計算で実行したときの過程) を辿ることができない．これが不安定性の原因である [1)]．

アルゴリズムの安定化は，アルゴリズムの構造を変えずにデータ集合を変えることで行われる．データ集合は，元の係数から区間係数に変えられる．区間係数は，2 つの浮動小数点数のペア $[\alpha, \beta]$ で表され，$[\alpha, \beta] = \{\gamma \mid \alpha \leq \gamma \leq \beta\}$ である．区間には，中心 A，半径 α で囲う形式 (円形区間) なるブラケット係数 $[A, \alpha]$ もあるが，本論では下界と上界による矩形区間を採用する．ブラケット係数の詳細については，文献 [1] を参照されたい．元のアルゴリズムにおいて係数の間で演算が行われる部分は，区間係数の間で区間演算が行われる．重要なポイントは，述語を評価する直前に，「ゼロ書換え」を行うことである．具体的には，区間係数が 0 を含むとき，その区間係数を 0 区間 (下界 0 上界 0 の区間) に書き換える．それ以外のときは，何も変えず，そのままとする．ゼロ書換えを行った後，その区間係数の第 1 要素，すなわち区間の下界において述語を評価する (第 2 要素の上界で評価しても差し支えない)．ゼロ書換えの特筆すべき特徴は，区間係数の列の収束性を保存すること，そして，区間係数の列が 0 に" 近似的に" 収束する (定義は後述) ならば，ゼロ書換えされた区間係数の列は有限回のステップで 0 に「到達する」という点である．従って，述語

[1)] ただし，この具体例はアルゴリズムに潜む不安定性を説明するだけのためである．実際の数値計算による 0 判定にはいろいろな工夫がなされている．例えば，適当なガード桁を用意して，その桁数以下に浮動小数の値が収まれば，その数を 0 と見なすという便法がある．しかしながら，ガード桁をどの程度にすればよいかという難しい問題があり，数値計算の分野では誤差解析などで盛んに研究が行われている．

がまさに不連続点の 0 で評価できるようになる．もしゼロ書換えを行わなかったとしたら，述語は 0 の値で評価されない可能性があり，もとより 0 はその述語の不連続点であったがために，アルゴリズムは元のアルゴリズムと同じ実行過程を辿らなくなる．

アルゴリズムの安定化手法についてまとめると，次のようになる．$\mathcal{A}$ を不連続点 0 の代数的アルゴリズム，$Stab(\mathcal{A})$ を文献 [1] のテクニックによって安定化されたアルゴリズムとする．アルゴリズム $Stab(\mathcal{A})$ は次の特徴を持つ．

1. **区間領域**　データ領域は区間を係数に持つ多項式の集合である．区間係数は $[\alpha, \beta]$ の形をとる．ここで，$\alpha, \beta \in R, \alpha \leq \beta$ である．$[\alpha, \beta]$ は，集合 $\{\gamma \in R \mid \alpha \leq \gamma \leq \beta\}$ を表現する．

2. **区間演算**　区間係数の加減乗除に対しては，区間演算を用いる．区間係数間の二項演算子 $\circ \in \{+, -, \times, /\}$ に対して，

$$[\alpha, \beta] \circ [\theta, \eta] \stackrel{\text{def}}{=} [\min(\alpha \circ \theta, \alpha \circ \eta, \beta \circ \theta, \beta \circ \eta), \max(\alpha \circ \theta, \alpha \circ \eta, \beta \circ \theta, \beta \circ \eta)]$$

　である．

3. **ゼロ書換え**　不連続点 0 を持つ述語が評価される直前に，ゼロ書換えを行う．すなわち，各区間係数 $[\alpha, \beta]$ に対して，$\alpha \leq 0 \leq \beta$ ならば $[\alpha, \beta]$ を $[0, 0]$ に書き換える．そうでなければ，$[\alpha, \beta]$ のままとする．

$Stab(\mathcal{A})$ は区間係数多項式 (の集合) を入力として受け入れ，区間係数多項式 (の集合) を出力として返す．$\mathcal{A}$ に対する入力 $f \in R[x_1, \ldots, x_m]$ は，$Stab(\mathcal{A})$ に対する入力のために，区間係数多項式の列 $\{Stab(f)_j\}_j$ に近似される．具体的には，f を $\sum_{i_1, \ldots, i_m} a_{i_1 \ldots i_m} x_1^{i_1} \ldots x_m^{i_m}$ と表すと，$Stab(f)_j$ は次のように定義される．

$$\sum_{i_1, \ldots, i_m} [(\alpha_{i_1 \ldots i_m})_j, (\beta_{i_1 \ldots i_m})_j] x_1^{i_1} \ldots x_m^{i_m} \quad \text{where} \quad (\alpha_{i_1 \ldots i_m})_j \leq (\beta_{i_1 \ldots i_m})_j$$

ここに，ある l があって，$j \geq l$ ならば，

$$(\alpha_{i_1 \ldots i_m})_j \leq a_{i_1 \ldots i_m} \leq (\beta_{i_1 \ldots i_m})_j$$

であり，$j \to \infty$ のとき，任意の $i_1 \ldots i_m$ に対して $(\alpha_{i_1 \ldots i_m})_j - (\beta_{i_1 \ldots i_m})_j \to 0$ である．このとき，$Stab(f)_j$ は f に近似的に収束すると呼ぶ．各 j に対し，$Stab(\mathcal{A})$ を入力 $Stab(f)_j$ で実行したとき，その出力を $Stab(\mathcal{A})(Stab(f)_j)$ と書く．

次の定理は $Stab(\mathcal{A})$ の特徴を主張している．

定理 1 (係数収束 [1])

アルゴリズム $\mathcal{A}$ は入力 $f \in R[x_1, \ldots, x_m]$ に対して正常終了し，f に近似的に収束する区間係数多項式の列 $\{Stab(f)_j\}_j$ が与えられていると仮定する．このとき，ある n が存在して，$j \geq n$ ならば，$Stab(\mathcal{A})$ は入力 $Stab(f)_j$ に対して正常終了し，かつ，$j \to \infty$ のとき，

$$Stab(\mathcal{A})(Stab(f)_j) \to \mathcal{A}(f)$$

となる．

係数収束の実用例として，一般逆行列を求めるグレビルのアルゴリズムなどがある．その他の例は，文献 [2] を参照されたい．

3 アルゴリズムのISCZ法

3.1 準備

ISCZ 法を説明するための準備を行う．Interval with Symbol を次のように定義する．

定義 Interval with Symbol (IS)
区間係数 $[\alpha, \beta]$ $(\alpha, \beta \in \mathbb{F},\ \alpha \leq \beta)$ と形式的なシンボル *Symbol* に対して，次のペア

$$\big[[\alpha, \beta], \mathit{Symbol}\big]$$

を Interval with Symbol (IS または IS 係数) という．また，IS 係数の第 2 要素 (*Symbol*) をシンボル部分という．IS 係数からなる集合を $\mathbb{IS}$ とする．

シンボルは，アルゴリズム実行中に現れる係数を記録するために使用される．IS 係数のシンボル部分の膨張を防ぐための一つの工夫について述べる．基本的な方針は，シンボルの代わりに自然数を用いることである．すなわち，IS 係数の代わりに次のペア

$$\big[[a_1, a_2], L\big] \ \text{where}\ ([a_1, a_2] \text{は区間係数}) \wedge ((L \text{はシンボル}) \vee L \in \mathbb{N})$$

を用いる．さらに，各ペアが他のペアからどのように構成されたかを示すリストを用意する．これをシンボルリスト (Symbol List) と呼ぶ．

3.2 ISCZ法

3.2.1 理論

不連続点 0 の代数的アルゴリズムに対し，正確係数の出力を得るために厳密計算を軽減するために提案された手法 [9] を復習する．この手法の基本的なアイディアは，係数を記録するために IS 係数を使用する．重要なことは，各ゼロ書換えにおいてシンボルを実数値に復元して真にゼロ書換えが正しいかどうかを確認する．ISCZ 法の手続きは次の通りである．

1. R-to-IS: 各入力係数 a をペア
$$\big[[a_1, a_2], \mathit{Symbol}_a\big]$$
に変換する．ここで，$[a_1, a_2]$ $(a_1 \leq a_2)$ は a の予め定められた精度の区間係数，Symbol_a は a を表すシンボルである．

2. IS 演算: IS 演算を次のように定義する．
$$\begin{aligned}&\text{For } \big[[a_1, a_2], \mathit{Symbol}_a\big], \big[[b_1, b_2], \mathit{Symbol}_b\big] \in \mathbb{IS},\\ &\big[[a_1, a_2], \mathit{Symbol}_a\big] \circ \big[[b_1, b_2], \mathit{Symbol}_b\big] \stackrel{\text{def}}{=} \big[[a_1, a_2] \circ [b_1, b_2], (\hat{\circ}, \mathit{Symbol}_a, \mathit{Symbol}_b)\big]\\ &\text{where } \circ \in \{+, -, \times, /\}\end{aligned}$$

3. Symbol List: IS 係数のシンボル部分の膨張を防ぐために前述で定義したシンボルリストを使用する．

4. *Correct* Zero Rewriting (正しいゼロ書換え): *Correct* Zero Rewriting を次のように定義する.

$$\forall\left[[a_1,a_2],s\right]\in\mathbb{IS},\ a_1\le 0\le a_2\text{のとき}$$
$$s\text{をそれに対応する}r(s)\text{に復元する}$$
$$\begin{cases} r(s)=0 & \Rightarrow \text{次のステップに進む}\\ r(s)\neq 0 & \Rightarrow \text{精度を上げてR-to-ISに戻る}\end{cases}$$

効率化のために，後の再利用を考え s の実数値 $r(s)$ を記憶しておく.

5. IS-to-R: 出力のシンボル部分の中の各入力シンボルにそれぞれ対応する入力係数を代入し，演算シンボルに演算の意味を与えて実数値に復元する.

以上の手法を ISCZ 法 (IS method with *Correct* Zero rewriting) と呼ぶ.

さて，ある精度 j で $Stab(f)_j$ を $Stab(\mathcal{A})$ に入力したときの実行過程は，もしアルゴリズム中のすべてのゼロ書換えが正しいならば，真の入力 f を $\mathcal{A}$ に入力したときの実行過程と完全に一致する (i.e., アルゴリズムの正しい実行過程を辿ることが出来る). ISCZ 法では，各ゼロ書換えにおいて，それが正しいかどうかを確認する．さらに，安定化定理により，すべてのゼロ書換えが正しくなる精度が存在する．故に，ISCZ 法は有限ステップで終了し，その出力の各 IS 係数のシンボルは正しい実数値を与える.

以上より，次の定理が成り立つ.

定理 2 (ISCZ 法の停止性と正当性-[9])

$\mathcal{A}$ が入力 I で正常終了すると仮定せよ．このとき，$\mathcal{A}$ に対する *ISCZ* 法は，常に有限ステップで終了し，正しい結果 $\mathcal{A}(I)$ の出力と同じ結果を与える.

ISCZ 法のメリットは次の通りである.

1. 実数値に復元された出力が真に正しい出力かどうかを確認するという正当性の確認が不要である.
2. 任意の $\left[[a_1,a_2],s\right]\in\mathbb{IS}$ に対して，$a_1\le 0\le a_2$ でない限り，厳密計算をスキップすることができ，浮動小数点計算だけで済む．すなわち，ゼロでない係数についての厳密計算を省略することが出来る．従って，ISCZ 法は，$a_2<0\vee 0<a_1$ の場合が $a_1\le 0\le a_2$ の場合よりも多いとき，厳密計算をより多くスキップ出来るため有効である.

3.2.2 New Idea

本論では，ISCZ 法に対する New Idea[2] を提案する[3]．基本的な方針は，なるべくアルゴリズムの最初に戻らずに効率的に計算を進めるというものである.

•New Idea (*Correct* Zero Rewriting の工夫): *Correct* Zero Rewriting を次のように工夫する．復元した実数値 $r(s)\neq 0$ のとき，精度を上げる．このとき，単にアルゴリズムの最初に戻って R-to-IS を行うのではなく，その時点でのシンボルの状況から適切な精度 μ' を探す．すなわち，対象の IS 係数中の区間係数 $[a_1,a_2]$ に対して，$a_1\le 0\le a_2$ を満たさなくなる ($a_2<0\vee 0<a_1$を満たす) 精度 μ' を見つけるまで精度を上げ続ける．その後，精度 μ' でアルゴリズムの最初に戻って R-to-IS を行う.

アルゴリズム実行過程の終盤に，ゼロ書換えが多く発生する場合に上記の New Idea の恩恵を受けやすくなる．なぜならば，前述の通りなるべくアルゴリズムの最初に戻らないように工夫しているからである.

[2] 白柳 潔が中心となって白柳研究室にて 2015 年に編み出された.

[3] 更なる New Idea を既に考案しているが，今回は省略する.

4 ガウス既約基底

4.1 アルゴリズム

はじめに，ガウス既約基底を求めるアルゴリズム LLL[4] のための準備を行う．基底 $b_1,\ldots,b_n$ に対して，$b_i^*, \mu_{i,j}, b_i(j)$ を次のように定義する[5]．

$$b_i^* \stackrel{\text{def}}{=} \begin{cases} b_1 & \text{for } i=1 \\ b_i - \sum_{j=1}^{i-1} \frac{b_i \cdot b_j^*}{\|b_j^*\|^2} b_j^* & \text{for } i=2,\ldots,n \end{cases}$$

各 $1 \le j < i \le n$ に対して，グラムシュミット係数 $\mu_{i,j}$ を次のように定義する．

$$\mu_{i,j} \stackrel{\text{def}}{=} \frac{b_i \cdot b_j^*}{\|b_j^*\|^2}$$

ここで，

$$\mu_{i,i} \stackrel{\text{def}}{=} 1$$

である．また，各 $j \le i$ に対して，$b_1,\ldots,b_{j-1}$ に直交する b_i の成分を $b_i(j)$ と定義する．すると，$b_i(j)$ は $b_1^*,\ldots,b_{j-1}^*$ に直交するので，

$$b_i(j) = \mu_{i,j} b_j^* + \mu_{i,j+1} b_{j+1}^* + \cdots + b_i^*$$

と書ける．

次に，ガウス既約基底の定義を説明する．基底 $b_1,\ldots,b_n$ が，各 $1 \le i \le n-1$ に対して，

1. $\|b_i(i)\| - \frac{2}{\sqrt{3}}\|b_{i+1}(i)\| \le 0$
2. $|\mu_{i+1,i}| - \frac{1}{2} \le 0$

を満たすとき，$b_1,\ldots,b_n$ はガウス既約であると呼ばれる．ガウス既約基底を求め，その中で最小のノルムのものをとれば，近似の最短ベクトルが求まる．

以上で，アルゴリズム LLL のための準備が整った．次に，アルゴリズムを述べる．なお，LLL の具体例については，[7] を参照されたい．

アルゴリズム LLL
入力: 基底 $(b_1,\ldots,b_n)^T$　　//n × n 行列
出力: ガウス既約基底

Step1:while 基底 $b_1,\ldots,b_n$ がガウス既約でない
　StepA:各 $i(1 \le i \le n-1)$ に対して，

---

[4] Lenstra-Lenstra-Lovász によって考案されたアルゴリズムである．アルゴリズムの証明に関する詳細は，文献 [5] や [10] を参照されたい．

[5] この定義は，まさにグラムシュミットの直交化法 (Gram-Schmidt orthogonalization, or GSO) そのものである．GSO では，ベクトルを正規化する立場をとるものもあるが，本論文では正規化しない．

```
            if |μ_{i+1,i}| − 1/2 > 0
               m ← μ_{i+1,i} に最も近い整数 m
               b_{i+1} ← b_{i+1} − m b_i
            endif
   StepB:if ‖b_i(i)‖ − 2/√3 ‖b_{i+1}(i)‖ > 0
               b_i と b_{i+1} を交換する
         endif
Step2:(b_1, ..., b_n)^T を出力する    //n × n 行列
```

4.2 不安定性

アルゴリズム LLL の不安定性の原因について復習する．LLL の Step1 の StepA の if 文と StepB の if 文に注目すると，それぞれ

$$|\mu_{i+1,i}| - \frac{1}{2} > 0\ ,\ \|b_i(i)\| - \frac{2}{\sqrt{3}}\|b_{i+1}(i)\| > 0$$

であるかどうかの判定が頻繁に起きることがわかる．この述語は不連続点 0 を持ち，これがアルゴリズム LLL に不安定性を生じさせる可能性がある．具体的には，それぞれ

$$|\mu_{i+1,i}| \approx \frac{1}{2}\ ,\ \|b_i(i)\| \approx \frac{2}{\sqrt{3}}\|b_{i+1}(i)\|$$

であれば，厳密計算と浮動小数点近似計算とでは，アルゴリズムの過程がこの時点で異なってしまう．また，最も近い整数を選ぶ際にも不安定になり得ることにも注意されたい ([8] を参照されたい)．

アルゴリズム中の演算は，加減乗除 (及び絶対値，平方根) のみであることがわかるから，結論として，LLL は不連続点 0 の代数的アルゴリズムであることがわかる．

5 アルゴリズム LLL の安定化と ISCZ 法

5.1 *Stab(LLL),ISCZ(LLL)*

2 章で説明したアルゴリズムの安定化手法及び ISCZ 法を古典的なアルゴリズム LLL に適用する．

$Stab(LLL)$ について説明しよう．まず $Stab(LLL)$ のデータ領域は，区間係数を成分とする行列の集合である．区間演算が行われるのは，StepA 及び StepB の述語の左辺を計算するときである．ゼロ書換えが行われるのは，StepB の述語を評価する直前である．

定理 1 より，以下が成り立つ．

定理 3 (LLL **の安定化 [7]**)
行列 $A = [a_{kl}] \in \mathbb{K}^{n\times n}$ に対して，区間係数の行列の列 $Stab(A)_j = [Stab(a_{kl})_j]$ で，成分ごとに A に近似的に収束するものを考える．すなわち各 (k,l) について，$Stab(a_{kl})_j$ が a_{kl} に近似的に収束する行列の列を考える．このとき，$Stab(LLL)(Stab(A)_j) \to LLL(A)$ が行列成分ごとに成り立つ．

さらに，定理 2 より，以下が成り立つ．

定理 4 (*LLL* の ISCZ 法)

LLL に対する入力を I とし，$LLL(I)$ は正常終了すると仮定せよ．*LLL* に *ISCZ* 法を施したものを ISCZ(LLL) とする [6]．このとき，

$$\exists N \text{ such that } k \geq N \Rightarrow \mathrm{ISCZ(LLL)(ISCZ(I)}_k) = LLL(I)$$

5.2 実験結果

本実験で使用する PC 及び実行環境は次の通りである．

1. 使用コンピュータ
   OS: Windows 7 Home Premium
   CPU: Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz
   実装メモリ (RAM): 8.00 GB
2. 使用ソフト
   数式処理ソフト Maple14

実験

LLL の入力として次の 5×5 行列を仮定する．なお，行をベクトルと見なせば，これらは基底であることに注意せよ．

入力:
$$\begin{bmatrix} 116 & 29 & 77 & 39 & 15 \\ 1509 & 380 & 1008 & 517 & 206 \\ -41589 & -86427456 & 2103801 & -68729 & 10524652 \\ -5780859 & -12013416379 & 292428339 & -9553326 & 1462926632 \\ 1000000000000000 & 100000101010 & 1000111110 & 10000010110 & 101010101110110 \end{bmatrix}$$

厳密計算の結果は次の通りである．

厳密計算
$$\begin{bmatrix} 1 & 3 & 7 & 10 & 11 \\ 12 & 5 & 0 & 5 & 4 \\ 20 & -11 & 77 & -1 & -17 \\ -3258869 & -84657952 & -10282727 & 92135 & 13259340 \\ 259483651263947 & -14076040084392 & 36338913119143 & -313662592872765 & -196524401222718 \end{bmatrix}$$

続いて，浮動小数点近似計算による結果を述べる．まずは，精度桁 30 桁のときについてである．

近似計算 (精度桁 30 桁)
$$\begin{bmatrix} 1.0 & 3.0 & 7.0 & 10.0 & 11.0 \\ 12.0 & 5.0 & 0.0 & 5.0 & 4.0 \\ 20.0 & -11.0 & 77.0 & -1.0 & -17.0 \\ -3258869.0 & -84657952.0 & -10282727.0 & 92135.0 & 13259340.0 \\ 259483651300000.0 & -14076040080000.0 & 36338913120000.0 & -313662592900000.0 & -196524401200000.0 \end{bmatrix}$$

この結果は，厳密計算の結果と出力がほとんど同じで，アルゴリズム *LLL* はあたかも安定しているように見える．ところが，精度桁を上げて精度桁 70 桁で浮動小数点近似計算してみると，次のような 5×5 行列

[6] データ領域は，IS 係数を成分とする行列の集合である

が出力された．

近似計算 (精度桁 70 桁)

$$\begin{bmatrix} 12.0 & 5.0 & 0.0 & 5.0 & 4.0 \\ -11.0 & -2.0 & 7.0 & 5.0 & 7.0 \\ 127.0 & 31.0 & 70\ 0 & 34.0 & 8.0 \\ -20471317.0 & -91414240.0 & -9156679.0 & -5538105.0 & 9237740.0 \\ 320310359800000.0 & 9799864197000.0 & 323595595740000.0 & -293766006000000.0 & -182312553400000.0 \end{bmatrix}$$

これは，厳密計算の 5×5 行列と結果が全く異なっていることがわかる．この結果からいえることは，精度桁を上げれば解が収束するわけではないということである．従って，このアルゴリズムには「不安定性」が認められる．一方，安定化手法では，精度桁 26 桁から安定化したと考えられる．

計算時間及びメモリ: 計算時間については，次の表 1 の通りである．近似計算と安定化手法の精度桁は 70 桁で固定しており，単位は second である．また，ISCZ 法は初期精度は 1 で，精度の増加幅も 1 としている．ただし，#ZR はゼロ書換えの回数を表し，#SL は Symbol List の要素数を表している．以降も同様である．

表 1. 計算時間

	厳密計算	近似計算	安定化手法	ISCZ 法	(#ZR/#SL)
$\mathbb{Z}^{5\times5}$	0.546	0.374	3.260	34.133	(1543/143432)
$\mathbb{K}^{5\times5}$	45.007	0.359	2.917	14.086	(9846/123441)
$\mathbb{K}^{6\times6}$	1 週間以上	2.324	16.879	27.316	(6027/541325)
$\mathbb{K}^{7\times7}$	1 週間以上	15.709	95.723	146.563	(10483/1113341)

$\mathbb{K}$ 上の係数をもつ 5×5 行列を対象とした実験結果は次の表 2 の通りである．ex.1,2,3 の近似計算と安定化手法の精度桁は 70 桁，ex.4 の近似計算と安定化手法の精度桁は 120 桁である．

表 2. 計算時間

$\mathbb{K}^{5\times5}$	厳密計算	近似計算	安定化手法	ISCZ 法	(#ZR/#SL)
ex.1	1 週間以上	11.497	61.792	19292.331	(10483/1113341)
ex.2	1 週間以上	0.874	4.711	15.601	(7/143422)
ex.3	2 週間以上	0.609	3.183	7.722	(6/123441)
ex.4	1 週間以上	1.638	10.187	11246.268	(577/266168)

表 1 及び表 2 からわかる通り，成分が整数のみのときは安定化手法の有効性は認められなかった．ところが，成分を一般の代数体に拡げたときは，厳密計算が 1 週間以上かかってしまうのに対して，安定化手法は 2 分以内と安定化手法の方が圧倒的に早い結果となった．また，ISCZ 法についても，厳密計算よりも早い時間 (20000 秒以内) に正しい結果を出力している事がわかる．従って，一般の代数体 $\mathbb{K}$ 上では，安定化手法と ISCZ 法が有効であるということが確認できた．

続いて，従来の ISCZ 法と今回我々が新たに提案した New Idea の比較実験を次の表 3 及び表 4 に示す [7]．

表 3. 従来の方法 (左) と New Idea (右)

		従来の方法		New Idea	
		計算時間	メモリ	計算時間	メモリ
$\mathbb{Z}^{5\times5}$	ex.1	37.72s	12.51GiB	33.92s	7.35GiB
$\mathbb{Z}^{5\times5}$	ex.2	71.11s	14.29GiB	50.26s	8.81GiB
$\mathbb{K}^{5\times5}$	ex.1	13.04s	2.92GiB	13.00s	2.68GiB
$\mathbb{K}^{5\times5}$	ex.2	3.72m	99.11GiB	7.24s	1.49GiB
$\mathbb{K}^{5\times5}$	ex.3	26.86s	6.79GiB	17.64s	3.86GiB

表 4. 従来の方法 (左) と New Idea (右)

		従来の方法		New Idea	
		計算時間	メモリ	計算時間	メモリ
$\mathbb{Q}^{6\times6}$	ex.1	2.53m	24.20GiB	65.94s	11.19GiB
$\mathbb{K}^{6\times6}$	ex.1	24.88s	3.71GiB	27.24s	3.57GiB
$\mathbb{K}^{6\times6}$	ex.2	21.38m	113.76GiB	10.46m	56.46GiB
$\mathbb{K}^{7\times7}$	ex.1	4.09m	25.48GiB	100.93s	10.22GiB
$\mathbb{K}^{7\times7}$	ex.2	4.13m	25.27GiB	2.43m	16.29GiB

表 3 及び表 4 からわかる通り，従来の ISCZ 法に比べて計算時間，総使用メモリ量共に New Idea の方が有効であることがわかる．顕著な例として，例えば表 4 中の $\mathbb{K}^{6\times6}$ の ex.2 を見てみると，従来の方法では計算時間が 21.38m，総使用メモリ量が 113.76GiB である一方で，New Idea ではそれぞれ 10.46m(−10.92m)，56.46GiB(−57.30GiB) であり，有効性が見られる．

[7] メモリは総使用メモリ量を表し，単位は s=second，m=minute，GiB=GibiByte=2^{30}byte である

考察: 前述のように，近似計算による不安定性の原因は，無限小数に対して丸める操作による誤差が生じて，不等式の真偽が全く異なってしまう点にあると考えられる．安定化手法及びISCZ法では，その部分を上手にカバーしている．

アルゴリズム LLL には，今回の実験結果より，不安定性が認められる．そして，一般の代数体 $\mathbb{K}$ 上では，安定化手法のメリットを十分に活かせることが確認できた．さらに，厳密計算よりもISCZ法が有効であることも確認できた．ISCZ法について，総演算回数に対してゼロ書換えの回数の割合が少なければ少ないほど計算時間がかからないということを確認できた．これは，厳密計算を実行する過程が減少するためである．また，計算時間及び総使用メモリ量共に従来の方法よりもNew Ideaの方が有効であることを示すことができた．

6 おわりに

ガウス既約基底を，有限精度の浮動小数点計算を用いても安定に計算できるための手法を提案し，実際にその効果を実証した．また本論文では，安定化手法からヒントを得て提案されたISCZ法について，今回我々が新たに提案したNew Ideaの著しい有効性を主に主張した．ただし，これは数値計算で行われている便法を否定するものではないことに注意されたい．本手法をそれと比較しながら研究を進めるべきであろう．

今後の課題としては，行列(基底)が与えられたとき，本手法が正確な答を与えるのにどれだけの桁数が必要であるかを理論的に見積もることが挙げられる．また，最短ベクトル問題へのアプローチとして，よりモダンなアルゴリズム，例えばδLLL簡約アルゴリズム[8]，MLLLアルゴリズム[9]やBKZアルゴリズムを調査することも重要である．

参 考 文 献

[1] K. Shirayanagi, M. Sweedler: A Theory of Stabilizing Algebraic Algorithms, Technical Report 95-28, Mathematical Sciences Institute, Cornell University (1995)

[2] 白柳 潔: アルゴリズムの安定化理論, 数式処理, Vol.5, No.2, pp.2-21 (1997)

[3] 白柳 潔: 代数的アルゴリズムの安定化理論, コンピュータソフトウェア, **Vol.19 No.3**, (2002), 49–65

[4] 白柳 潔, 新妻 弘崇: 実多項式行列スミス標準形の安定な浮動小数点計算法, 情報処理学会論文誌 第40巻, (1999)

[5] V.V. ヴァジラーニ, (訳) 浅野 孝夫: 近似アルゴリズム, 丸善出版, (2012)

[6] K. Shirayanagi: Floating point Gröbner bases, Mathematics and Computers in Simulation 42, (1996), 509–528

[7] 永嶋 裕樹, 白柳 潔: 安定化手法の最短ベクトルアルゴリズムへの適用について, 数理解析研究所講究録 1955 数式処理とその周辺分野の研究, pp.1-12, 京都大学数理解析研究所, (2015)

[8] 永嶋 裕樹: 安定化手法に基づく計算履歴法とδLLL簡約アルゴリズムへの適用, 東邦大学理学研究科修士論文, (2016)

[8] ガウス既約基底の一般化であるδLLL簡約基底を求める．詳細は，[10]を参照されたい．

[9] 現在，筆者が研究している途中である．δLLLとMLLLの決定的な違いは，入力のベクトルが線形独立でなくてもよいという点にある．これにより，MLLLアルゴリズム中には不等式よりも強い条件である「ゼロかゼロでないか」の判定が多く含まれている．従って，δLLLアルゴリズムよりも著しく不安定になり得ると思われる．MLLLの詳細については[12]を参照されたい．

[9] K. Shirayanagi, H. Sekigawa: Reducing Exact Computations to Obtain Exact Results Based on Stabilization Techniques, Symbolic-Numeric Computation, pp.191-197, (2009)

[10] M.R. Bremner: Lattice Basis Reduction, CRC Press, (2012)

[11] H. Cohen: A Course in Computational Algebraic Number Theory, Springer-Verlag, (1993)

[12] M. Pohst: A Modification of the LLL Reduction Algorithm , J. Symbolic Computation, (1987)

[13] D. ミッチアンチオ, S. ゴールドヴァッサー, (訳) 林 彬: 暗号理論のための格子の数学, 丸善出版株式会社, (2012)

[14] 齋藤 宣一: 数値解析入門, 東京大学出版会, (2012)

[15] 一松 信: 数値解析, 朝倉書店, (1982)