数理解析研究所講究録
第1955巻 2015年 1-12

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

Application of the Stabilization Techniques to the Shortest Vector 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

最短ベクトルアルゴリズム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を最短ベクトルアルゴリズムに適用し、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が示される。

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。本論では、このガウス既約基底に焦点を当てる。

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

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

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。

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、最短ベクトルアルゴリズムがその適用条件を満たすことを確認し、実際に安定化手法を適用する。

まず 2 章では、安定化手法の概略を述べる。3 章では、安定化手法を適用する対象となるアルゴリズムについて述べる。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。3.3 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。4.1 節では、そのアルゴリズムに安定化手法を適用する。4.2 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 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。具体例を挙げよう。例えば、次のようなアルゴリズム REPEAT_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 $\circ \in \{+, -, \times, \div\}$ に対して、

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

[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。

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、$\alpha \le 0 \le \beta$ 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。

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

ここに、ある l があって、$j \ge l$ ならば、

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

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)

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、$j \ge n$ 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。

さて、本アルゴリズムにあまり恩恵を与えないが、安定化手法を論じる際に重要である「台収束」についても説明しよう。台収束は、係数収束よりも強い収束を実現する。なお、最後に出力をゼロ書換えし、実係数多項式への変換を行うアルゴリズムを以下 $Stab(\mathcal{A})_R$ と書く。台収束について説明するために多項式の台を定義する。多項式

$$f = \sum_{i_1,\ldots,i_m} a_{i_1\ldots i_m} x_1^{i_1}\ldots x_m^{i_m}$$

の台とは、0 でない係数を持つ変数の冪積 (係数 1 の単項式) の集合

$$\{x_1^{i_1}\cdots x_m^{i_m} \mid a_{i_1\ldots i_m} \neq 0\}$$

であり、これを $Supp(f)$ と書く。同様に区間係数多項式 F に対しても $Supp(F)$ は、$[0,0]$ でない区間係数を持つ冪積の集合である。直感的にいえば、台は「多項式の形」である。$Stab(\mathcal{A})_R(Stab(f)_j)$ の形は有限回のステップで (ある有限の j で)、$\mathcal{A}(j)$ の形と同じになる。言い換えれば、$Stab(\mathcal{A})_R(Stab(f)_j)$ の係数で、0 に収束するものは、有限ステップで 0 に到達する。これを台収束と定義する。

次の定理が、$Stab(A)_R$ の安定性を述べるものである。

定理 2 (台収束)

定理 1 と同じ仮定を置く。このとき、ある n が存在して、$j \ge n$ のとき、$Stab(\mathcal{A})_R$ は入力 $Stab(f)_j$ に対して正常終了し、かつ、次の 2 つを満たす。

1. $Stab(\mathcal{A})_R(Stab(f)_j) \to \mathcal{A}(f)\,(j \to \infty)$
2. ある N が存在して、$j \geq N$ ならば、

$$Supp(Stab(\mathcal{A})_R(Stab(f)_j)) = Supp(\mathcal{A}(f))$$

係数収束や台収束の実用例として、ブッフバーガーアルゴリズムやスツルム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 ガウス既約基底

3.1 アルゴリズム

はじめに、最短ベクトルアルゴリズム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。

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

各 $1 \leq j < i \leq n$ に対して、

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

さらに、

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

また、各 $j \leq i$ 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 $1 \leq i \leq n-1$ に対して、

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

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、最短ベクトルアルゴリズムのための準備が整った。次に、アルゴリズムを述べる。

最短ベクトルアルゴリズム
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 $i(1 \leq i \leq n-1)$ に対して、
　　if $|\mu_{i+1,i}| - \frac{1}{2} > 0$
　　$m \leftarrow \mu_{i+1,i}$ に最も近い整数 m
　　$b_{i+1} \leftarrow b_{i+1} - mb_i$
　　endif
　StepB:if $\|b_i(i)\| - \frac{2}{\sqrt{3}}\|b_{i+1}(i)\| > 0$
　　b_i と b_{i+1} を交換する
　endif
Step2:$(b_1, \ldots, b_n)^T$ を出力する　//n × n 行列

上記の Lenstra-Lenstra-Lovász によるアルゴリズムを、それぞれの頭文字をとることにより、以下 *LLL* で表す。また、アルゴリズムの証明に関する詳細は、文献 [5] を参照されたい。

3.2 具体例

ここでは、アルゴリズム *LLL* の具体例を挙げる。次の 3 つの 3 次元ベクトルを考えよう。

$$b_1 = (14, 5, 7)$$
$$b_2 = (4, 17, 8)$$
$$b_3 = (3, 1, 15)$$

以上の b_1, b_2, b_3 は基底であるため、*LLL* の入力の条件を満たす。従って、入力は次のように書ける。

$$\text{入力:} \begin{bmatrix} 14 & 5 & 7 \\ 4 & 17 & 8 \\ 3 & 1 & 15 \end{bmatrix}$$

続いて、Step1 の 1 回目の反復処理により b_1, b_2, b_3 はそれぞれ次のようになる。

$$b_1 = (14, 5, 7), \quad b_2 = (3, 1, 15), \quad b_3 = (-10, 12, 1)$$

Step1 の 2 回目の反復処理により、

$$b_1 = (-11, -4, 8), \quad b_2 = (14, 5, 7), \quad b_3 = (-10, 12, 1)$$

となり、Step1 の 3 回目の反復処理により、

$$b_1 = (-11, -4, 8), \quad b_2 = (3, 1, 15), \quad b_3 = (-10, 12, 1)$$

となる。ここで、b_1, b_2, b_3 はガウス既約の定義を満たすため、Step2 として次の 3×3 行列が出力される。

$$\text{出力:} \begin{bmatrix} -11 & -4 & 8 \\ 3 & 1 & 15 \\ -10 & 12 & 1 \end{bmatrix}$$

3.3 不安定性

アルゴリズム LLL の不安定性の原因について議論する。LLL の `Step1` 中の `StepB` に注目すると、$\|b_i(i)\| - \frac{2}{\sqrt{3}}\|b_{i+1}(i)\| > 0$ 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、$\|b_i(i)\| \approx \frac{2}{\sqrt{3}}\|b_{i+1}(i)\|$ 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。

アルゴリズム中の演算は、加減乗除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。

4 最短ベクトルアルゴリズムの安定化

4.1 $Stab(LLL)$

2 章で説明したアルゴリズムの安定化手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)
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。

4.2 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 1

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)
$$\begin{bmatrix} 12.0 & 5.0 & 0.0 & 5.0 & 4.0 \\ -11.0 & -2.0 & 7.0 & 5.0 & 7.0 \\ 127.0 & 31.0 & 70.0 & 34.0 & 8.0 \\ -20471317.0 & -91414240.0 & -9156679.0 & -5538105.0 & 9237740.0 \\ 320310359800000.0 & 9799864197000.0 & 32359595740000.0 & -293766006000000.0 & -182312553400000.0 \end{bmatrix}$$

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。

実験 2

LLL の入力として次の 6×6 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
$$\begin{bmatrix} \frac{5234611161119}{5328795531605} & \frac{27751382652}{25341106103} & \frac{88660895770}{235243225523} & \frac{511035722683}{844887403917} & \frac{957086086711}{268148812305} & \frac{93352455747}{269639578895} \\ \frac{29}{19} & \frac{11}{29} & \frac{5}{11} & \frac{23}{13} & \frac{2}{5} & \frac{5}{13} \\ \frac{520846782179}{71936486556} & \frac{504675355147}{64127188429 0} & \frac{395434900123}{41763311281 3} & \frac{22253211857}{18852353807 4} & \frac{433319885771}{46314460336 2} & \frac{86472740614}{7774816624 07} \\ \frac{938707368057}{224630307242} & \frac{871618127793}{601213403174} & \frac{778620105762}{578965114615} & \frac{648093863358}{208724787311} & \frac{189737925613}{127082697726} & \frac{114027765715}{101912645384} \\ \frac{1004}{35815} & \frac{4231}{393965} & \frac{1575}{157586} & \frac{927}{78793} & \frac{80349}{157586} & \frac{327823}{157586} \\ \frac{1632457762 51}{19248115120 6} & \frac{33639537198 1}{15573809747 9} & \frac{68879383805 8}{35191731370 9} & \frac{7694333586 3}{46997483243 2} & \frac{992738262 1}{10453017486} & \frac{12589447692 1}{15958201643 0} \end{bmatrix}$$

厳密計算の結果は次の通りである。係数膨張が発生してしまっているため、一部結果を省略している。

厳密計算

$$\begin{bmatrix} \frac{29}{19} & \frac{11}{29} & \cdots \\ \frac{1004}{35815} & \frac{4231}{393965} & \cdots \\ \frac{3739429888803447853 36}{3817016084886615} & -\frac{8226851143592818}{907591715078945} & \cdots \\ -\frac{2343290259358471}{1366793244564} & \frac{13839895145235968}{1859688464441} & \cdots \\ \frac{1197762546276183151 1968024}{3488899846167060981051} & -\frac{2160395715671199430066597432}{559034815275001947767867} & \cdots \\ \frac{8829859567531093122170422851080659855800072}{6329334799497935524870579684691494 05} & -\frac{5194306681836335712441432901664674603306 4152}{1182751107075829161681061543355090575405} & \cdots \end{bmatrix}$$

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)

$$\begin{bmatrix} 1.52631\ldots78947 & 0.37931\ldots24138 & 0.45454\ldots54545 & 1.76923\ldots76923 & \cdots \\ 0.02803\ldots53651 & 0.01073\ldots16651 & 0.00999\ldots42104 & 0.01176\ldots27813 & \cdots \\ 9796.73599\ldots64084 & -9.06448\ldots05570 & 3759.44841\ldots88973 & 593.72690\ldots44258 & \cdots \\ -1714.44384\ldots49971 & 7442.05032\ldots39191 & 8955.74847\ldots06130 & -815.29796\ldots91818 & \cdots \\ 3433.06657\ldots55853 & -386.45101\ldots29803 & -4463.01644\ldots86998 & 1633.70093\ldots85286 & \cdots \\ 139506.9126 & -43917.15764\ldots61558 & 1617258.706 & 231824.3765 & \cdots \end{bmatrix}$$

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 140 桁で浮動小数点近似計算してみると、次のような 6×6 行列が出力された。

近似計算 (精度桁 140 桁)

$$\begin{bmatrix} 0.02803\ldots51277 & 0.01073\ldots16847 & 0.00999\ldots31089 & 0.01176\ldots67545 & \cdots \\ 1.52631\ldots94737 & 0.37931\ldots17241 & 0.45454\ldots54545 & 1.76923\ldots92308 & \cdots \\ 5309.93937\ldots16974 & -1120.52557\ldots89062 & 2424.81234\ldots69532 & -4626.75878\ldots21784 & \cdots \\ -1714.44384\ldots26265 & 7442.05032\ldots43361 & 8955.74847\ldots86129 & -815.29796\ldots64816 & \cdots \\ 3433.06657\ldots69800 & -386.45101\ldots08459 & -4463.01644\ldots77351 & 1633.70093\ldots76038 & \cdots \\ 139506.9126 & -43917.15764\ldots08696 & 1617258.706 & 231824.3765 & \cdots \end{bmatrix}$$

これは、厳密計算の 6×6 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、この実験結果からも LLL には「不安定性」が認められる。

最後に、安定化手法を適用したときの実験結果について述べる。次の結果は、精度桁が 20 桁のときの計算結果である。一部結果を省略している。

安定化手法 (精度桁 20 桁)

$$\begin{bmatrix} [1.5263157894736842100, 1.5263157894736842110] & [0.37931034482758620685, 0.37931034482758620695] & \cdots \\ [0.028032947089208432216, 0.028032947089208432226] & [0.010739532699605294882, 0.010739532699605294892] & \cdots \\ [9796.7359971069098193, 9796.7359971069098203] & [-9.0644846211241827685, -9.0644846211241827575] & \cdots \\ [-1714.4438404844241933, -1714.4438404844241922] & [7442.0503271745970269, 7442.0503271745970280] & \cdots \\ [3433.0665799766554941, 3433.0665799766554963] & [-386.45101461318 6774, -386.45101461318 6762] & \cdots \\ [139506.9126, 139506.9126] & [-43917.157640025067767, -43917.157640025066904] & \cdots \end{bmatrix}$$

次に、厳密計算での結果を精度桁 10 桁で近似したものを参考のために記載する。

厳密計算の近似 (精度桁 10 桁)

$$\begin{bmatrix} 1.526315789 & 0.3793103448 & 0.4545454545 & 1.769230769 & 0.4000000000 & 0.3846153846 \\ 0.02803294709 & 0.01073953270 & 0.009994542662 & 0.01176500451 & 0.5098739736 & 2.080279974 \\ 9796.735997 & -9.064484621 & 3759.448415 & 593.7269059 & -478.7715442 & 1494.175065 \\ -1714.443840 & 7442.050327 & 8955.748478 & -815.2979610 & -450.2643961 & 678.3696680 \\ 3433.066580 & -386.4510146 & -4463.016449 & 1633.700938 & 15830.80089 & -1355.620459 \\ 139506.9126 & -43917.15764 & 1617258.706 & 231824.3765 & 163304.8909 & 196204.4464 \end{bmatrix}$$

以上の実験結果からわかるように、安定化手法の結果を各成分ごとに見てみると、厳密計算の結果である値をそれぞれ含んでいることがわかる。安定化手法の精度桁を 10 桁から徐々に上げていっても、同様な結果を得ることができた。従って、安定化手法では、精度桁 10 桁から安定化したと考えられる。

実験 3

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

$$\begin{bmatrix} -\frac{2}{5} - \frac{9}{5}\sqrt{2} & -2 & -\frac{3}{14} & 0 & \frac{10}{17}\sqrt{2} \\ \frac{1}{2380}\sqrt{-\frac{14734958137}{144} + \frac{218740802}{3}\sqrt{2}} & -\frac{3}{17} - 3/2\sqrt{2} & -\frac{12}{5} + \frac{13}{5}\sqrt{2} & -\frac{2}{3} + \frac{7}{20}\sqrt{2} & \frac{5}{16} + \sqrt{2} \\ -\frac{69}{82} + \frac{27}{61}\sqrt{2} & 1201 + \frac{33519}{19}\sqrt{2} & \frac{9023}{23} + \frac{73079}{79}\sqrt{2} & -\frac{77}{37} + \frac{9}{23}\sqrt{2} & -599 + \frac{45029}{29}\sqrt{2} \\ -\frac{8}{25} + \frac{5}{97}\sqrt{2} & 1001 - \frac{1591}{9}\sqrt{2} & -\frac{94}{29} - \frac{11}{35}\sqrt{2} & \frac{77087}{87} + \frac{6531}{31}\sqrt{2} & \frac{3079}{79} - 19\sqrt{2} \\ -\frac{89}{88} - \frac{9}{5}\sqrt{2} & -\frac{16}{23} - \frac{11}{37}\sqrt{2} & -\frac{2123}{2} + \frac{31513}{13}\sqrt{2} & \frac{14}{17} - \frac{39}{50}\sqrt{2} & \frac{1059}{59} + \frac{9109}{9}\sqrt{2} \end{bmatrix}$$

厳密計算の結果は次の通りである。係数膨張が発生してしまっているため、一部結果を省略している。

厳密計算

$$\begin{bmatrix} -\frac{2}{5} - \frac{9}{5}\sqrt{2} & -2 & -\frac{3}{14} & \cdots \\ \frac{1}{28560}\sqrt{-14734958137 + 10499558496\sqrt{2}} & -\frac{3}{17} - \frac{3}{2}\sqrt{2} & -\frac{12}{5} + \frac{13}{5}\sqrt{2} & \cdots \\ -\frac{8}{25} + \frac{5}{97}\sqrt{2} + \frac{11}{1904}\sqrt{-14734958137 + 10499558496\sqrt{2}} & \frac{16522}{17} - \frac{7637}{18}\sqrt{2} & -\frac{11578}{29} + \frac{15004}{35}\sqrt{2} & \cdots \\ -\frac{1069}{2050} + \frac{2314}{5917}\sqrt{2} - \frac{1}{4080}\sqrt{-14734958137 + 10499558496\sqrt{2}} & \frac{3421}{17} + \frac{667391}{342}\sqrt{2} & \frac{1375173}{3335} + \frac{2508311}{2765}\sqrt{2} & \cdots \\ -\frac{89}{88} - \frac{9}{5}\sqrt{2} & -\frac{16}{23} - \frac{11}{37}\sqrt{2} & -\frac{2123}{2} + \frac{31513}{13}\sqrt{2} & \cdots \end{bmatrix}$$

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)

$$\begin{bmatrix} -2.9455\ldots70358 & -2.0 & -0.21428\ldots14286 & 0.0 & 0.83189\ldots08359 \\ 0.37328\ldots08822 & -2.2977\ldots90985 & 1.2769\ldots68295 & -0.17169\ldots13193 & 1.7267\ldots72421 \\ 61.345\ldots72455 & 371.86\ldots94850 & 207.01\ldots78251 & 1155.6\ldots03254 & 297.01\ldots84115 \\ -64.902\ldots05854 & -372.97\ldots93495 & 2159.6\ldots49606 & -1155.9\ldots40857 & 1152.2\ldots84026 \\ 188.56\ldots36855 & 4078.8\ldots68349 & -2417.0\ldots34547 & 2283.2\ldots2389 & -434.85\ldots94614 \end{bmatrix}$$

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 行列が出力された。

近似計算 (精度桁 70 桁)

$$\begin{bmatrix} 0.37328\ldots89299 & -2.2977\ldots03745 & 1.2769\ldots75903 & -0.17169\ldots99105 & 1.7267\ldots90732 \\ -2.9455\ldots83318 & -2.0 & -0.21428\ldots57143 & 0.0 & 0.83189\ldots27835 \\ -159.30\ldots37306 & 642.99\ldots18606 & -15.257\ldots61482 & 1184.0\ldots26111 & 57.026\ldots02615 \\ 155.75\ldots89836 & -644.11\ldots71152 & 2381.9\ldots07610 & -1184.2\ldots15861 & 1392.2\ldots74959 \\ -252.73\ldots25759 & 4621.0\ldots77277 & -2861.5\ldots09404 & 2339.9\ldots34365 & -914.82\ldots78946 \end{bmatrix}$$

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。この結果からも、精度桁を上げれば解が収束するわけではないことがわかる。従って、この実験結果からも LLL には「不安定性」が認められる。

最後に、安定化手法を適用したときの実験結果について述べる。次の結果は、精度桁が 10 桁のときの計算結果である。一部結果を省略している。

安定化手法 (精度桁 10 桁)

$$\begin{bmatrix} [-2.945584417, -2.945584407] & [-2.000000005, -1.999999995] & [-0.2142857148, -0.2142857138] & \cdots \\ [0.3732891886, 0.3732891896] & [-2.297790936, -2.297790926] & [1.276955256, 1.276955266] & \cdots \\ [61.34561373, 61.34561389] & [371.8629648, 371.8629664] & [207.0117708, 207.0117725] & \cdots \\ [-64.90256194, -64.90256177] & [-372.9790604, -372.9790588] & [2159.650682, 2159.650694] & \cdots \\ [188.5693153, 188.5693158] & [4078.803166, 4078.803181] & [-2417.018964, -2417.018928] & \cdots \end{bmatrix}$$

次に、厳密計算での結果を精度桁 10 桁で近似したものを参考のために記載する。

厳密計算の近似 (精度桁 10 桁)

$$\begin{bmatrix} -2.945584412 & -2.0 & -0.2142857143 & 0.0 & 0.8318903306 \\ 0.3732891891 & -2.297790931 & 1.276955261 & -0.1716919200 & 1.726713562 \\ 61.34561381 & 371.8629656 & 207.0117717 & 1155.671168 & 297.0123636 \\ -64.90256186 & -372.9790596 & 2159.650688 & -1155.950725 & 1152.278048 \\ 188.5693156 & 4078.803173 & -2417.018946 & 2283.246433 & -434.851819 \end{bmatrix}$$

以上の実験結果からわかるように、安定化手法の結果を各成分ごとに見てみると、厳密計算の結果である値をそれぞれ含んでいることがわかる。安定化手法の精度桁を 10 桁から徐々に上げていっても、同様な結果を得ることができた。従って、安定化手法では、精度桁 10 桁から安定化したと考えられる。

計算時間: 計算時間については、次の表 1 の通りである。精度桁は 70 桁で固定している。

表 1. 計算時間

単位 [sec]	**厳密計算**	近似計算	**安定化手法**
$\mathbb{Z}^{5\times5}$	0.546	0.374	3.260
$\mathbb{K}^{5\times5}$	137.499	0.359	2.917
$\mathbb{K}^{6\times6}$	**1 週間以上**	1.294	16.287
$\mathbb{K}^{7\times7}$	**1 週間以上**	3.011	45.677

表 1 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 137.499 秒、安定化手法が 2.917 秒と安定化手法の方が圧倒的に早い結果となった。従って、一般の代数体 $\mathbb{K}$ 上では、安定化手法が有効であるということが確認できた。

考察: 3.3 節で述べたように、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。また、今回の実験では、安定化手法では、ユークリッドのアルゴリズムやブッフバーガーアルゴリズムに比べれば、比較的低い精度桁で安定化しているということが確認できた。

5 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。本論は、数値計算で行われている便法を否定するものではない。当意即妙に、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。さらに、安定化手法の発展形であるISCZ法を適用することも課題として挙げられる。ISCZ法の詳細については、文献[8]を参照されたい。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を調査することも重要である。

References

[1] Kiyoshi Shirayanagi, Moss 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] 薩摩 順吉, 大石 進一, 杉原 正顯: 応用数理ハンドブック, 株式会社朝倉書店, (2013)

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

[8] 伊井誠和, 白柳 潔: 安定化手法の発展形ISCZ法におけるシンボルリストの新しい評価法, 数理解析研究所講究録1907 数式処理とその周辺分野の研究, pp.71-79, 京都大学数理解析研究所, (2014)