数理解析研究所講究録
第1955巻 2015年 13-26

安定化手法の発展形ISCZ法のスツルムアルゴリズムへの適用とwebアプリへの応用

伊井 誠和
TOMOKAZU II
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 †

1 はじめに

安定化理論 [1, 2] は、近似計算で実行すると不安定となるアルゴリズムに対し、それを変形して近似計算で実行しても誤差の影響を抑制し、安定な出力を得られるようにするための理論である。昨年 [3] は、その安定化手法の発展形である ISCZ 法 [4, 5, 6] をユークリッドの互除法に適用した実験結果を報告した。本年は ISCZ 法をスツルムアルゴリズム [7] およびグラハムアルゴリズム [8] に適用した結果を報告する。その際、数式処理システム以外でも実装が可能となる実装方法を用いたため、スツルムアルゴリズムについては JavaScript でも実験を行った。この結果についても報告する。本論では 2 章で安定化理論と ISCZ 法について復習する。3 章で実装方法について説明し、4 章でその実装方法を用いた実験結果を、計算時間に注目し、本手法の有効性について検討する。

2 復習

2.1 安定化理論

次のアルゴリズムを対象に安定化理論の復習を簡単に行う。

- データは、すべて多項式環 $R[x_1, \cdots, x_m]$ の元からなる。R は実数体の部分体である。
- データ間の演算は、$R[x_1, \cdots, x_m]$ 内の加減乗である。
- データ上の述語は、不連続点をもつとすればそれは 0 のみである。

*6513001i@nc.toho-u.ac.jp
†kiyoshi.shirayanagi@nc.toho-u.ac.jp

述語の不連続点が 0 という意味は、If "$C = 0$" then ... else ... のように、値が 0 か否かによって分岐が別れることである。従って、$C = 0$ の代わりに $C > 0$ や $C \leq 0$ などでもよい。上記クラスのアルゴリズムを、不連続点 0 の代数的アルゴリズムと呼ぶ。ほとんどの数式処理のアルゴリズムはこのクラスに入るか、このクラスのアルゴリズムに変換可能である。

さて、安定化の 3 つのポイントは、

- アルゴリズムの構造は変えない。
- データ領域において、ふつうの係数を区間係数に変える。
- 述語の評価の直前で、区間係数のゼロ書換えを行なう。

である。すなわち、安定化されたアルゴリズムは次のようになる。

区間領域 データ領域は区間係数多項式の集合。区間係数は $[A, B]$ なる形で、$A, B \in \mathbb{R}$, $[A, B]$ は集合 $\{r \in \mathbb{R} \mid A \leq r \leq B\}$ を意味する。

区間演算 二項演算 $\star \in \{+, -, \times, \div\}$ に対し、

$$[A, B] \star [C, D] = [min(A \star C, A \star D, B \star C, B \star D), max(A \star C, A \star D, B \star C, B \star D)]$$

ゼロ書換え 不連続点 0 をもつ述語を評価する直前で、各区間係数 $[A, B]$ に対し、

$A \leq 0 \leq B$ ならば $[A, B]$ を $[0, 0]$ に書き換えよ。
そうでないならばそのままとせよ。

今、入力 $f \in R[x_1, \cdots, x_m]$ を

$$f = \sum_{i_1, \cdots, i_m} r_{i1 \cdots im} x_1^{i_1} \cdots x_m^{i_m}$$

と表したとき、f に対する近似列 $Int(f)_j$ を

$$Int(f)_j = \sum_{i_1, \cdots, i_m} [(a_{i1 \cdots im})_j, (b_{i1 \cdots im})_j] x_1^{i_1} \cdots x_m^{i_m}$$

で定義する。ここに、すべての $i_1, \cdots, i_m$ について、

$$(a_{i1 \cdots im})_j \leq r_{i1 \cdots im} \leq (b_{i1 \cdots im})_j \ for \ \forall j$$

$$(b_{i1 \cdots im})_j - (a_{i1 \cdots im})_j \to 0 \ as \ j \to \infty$$

このとき、単に

$$Int(f)_j \to f$$

と書く。

さて、A を安定化したアルゴリズムを $Stab(\mathcal{A})$ と書くと、次が安定化理論の基本定理 ([2]) である。

定理 1 (安定化理論の基本定理)
A は不連続点 0 の代数的アルゴリズムで、入力 $f \in R[x_1, \cdots, x_m]$ に対し正常終了するとせよ。このとき、f に対する任意の近似列 $\{Int(f)_j\}_j$ に対し、ある n が存在して、$j \geq n$ ならば、$Stab(\mathcal{A})$ は $\{Int(f)_j\}_j$ に対し正常終了し、

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

簡明を期すため、入力は一つだけの多項式にしているが、入力はもちろん、多項式の有限集合でもよい。

2.2 ISCZ法

2.2.1 シンボル付き区間

区間と形式的なシンボルを組み合わせた係数（シンボル付き区間）を導入する。区間は、従来と同様の意味の区間である。シンボルは、アルゴリズム実行中に現れる係数のlog（記録）を取るのに使われる。例えば、入力係数が$\frac{1}{3}$と$\frac{1}{7}$だったとしよう。これらの精度3の区間はそれぞれ$[0.333, 0.334]$と$[0.142, 0.143]$である。さて、$\frac{1}{3}$に対するシンボルをs、$\frac{1}{7}$に対するシンボルをtとして、区間と組み合わせると、それぞれ、$[[0.333, 0.334], s]$と$[[0.142, 0.143], t]$となる。次に、これらの間の演算、

$$[[0.333, 0.334], s] + [[0.142, 0.143], t] = [[0.333, 0.334] + [0.142, 0.143], s\dot{+}t]$$

と定義する。$[0.333, 0.334] + [0.142, 0.143]$に対しては通常の区間演算を使う。シンボル部分$s\dot{+}t$は再び形式的なシンボルで、加算を実施したことを記録できれば何でもよい。効率的なシンボル付けについては2.3節で議論する。

アルゴリズム終了後、最終的なシンボルを正確係数に復元する。先の簡単な例で言えば、もし最終的なシンボルが$s\dot{+}t$であったとすれば、sに$\frac{1}{3}$を、tに$\frac{1}{7}$を代入し、$\dot{+}$には加算の意味を与えて、$\frac{1}{3} + \frac{1}{7} = \frac{10}{21}$と復元する。

シンボル付き区間のことをinterval-symbol、あるいは単にISと呼ぶ。

2.2.2 手続き

Aを不連続点0の代数的アルゴリズムとする。ISCZ法の手続きは次の通りである。

R-to-IS 各入力係数rを$[[A, B], Symbol_r]$に変換する。ここに、$[A, B]$はrの予め定められた精度の区間、$Symbol_r$はrを表すシンボル(以下、入力シンボルと呼ぶ)である。

IS演算 IS間の演算を次のように実行する:

$$[[A, B], s] + [[C, D], t] = [[A, B] + [C, D], s\dot{+}t]$$

$$[[A, B], s] - [[C, D], t] = [[A, B] - [C, D], s\dot{-}t]$$

$$[[A, B], s] \times [[C, D], t] = [[A, B] \times [C, D], s\dot{\times}t]$$

$$[[A, B], s] \div [[C, D], t] = [[A, B] \div [C, D], s\dot{\div}t]$$

すなわち、区間部分については区間演算を用い、シンボル部分については加算、減算、乗算、除算の形式的なシンボル$\dot{+}, \dot{-}, \dot{\times}, \dot{\div}$を使って、どういう演算が行なわれたかを記録する。

正しいゼロ書換え 任意のIS$[[A, B], s]$に対し、$A \leq 0 \leq B$ならば、sをそれに対応する実数$r(s)$に復元する。もし、$r(s) = 0$ならば、次のステップに進む。そうでなければ、精度を上げて**R-to-IS**に戻る。

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。

この手法をISCZ法(IS method with correct zero rewriting)と呼ぶ。

さて、ある精度jで$Int(f)_j$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、真の出力fをAに入力したときの実行過程と完全に一致する。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、定理1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。従って、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正確係数を与える。

これを定理にまとめる。([4])

定理 2 (ISCZ 法の停止性と正当性)
A が入力 I で正常終了するとせよ。このとき、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、すなわち、$\mathcal{A}(I)$ の出力と同じ結果を与える。

ISCZ 法の利点は、ISZ 法と違い、出力の正当性を確認する必要がないことである。任意の IS$[[A,B],s]$ に対し、$A \leq 0 \leq B$ でない限り、正確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。換言すれば、ゼロでない係数についての正確計算を省略することができる。従って、本手法は、$A \leq 0 \leq B$ でない場合が $A \leq 0 \leq B$ である場合よりも多ければ多いほど有効であるということができる。

2.3 シンボルリスト

ISCZ 法では、IS のシンボル部分が膨張してしまうことがある。それを防ぐため、IS に用いるシンボルは全て整数とし、計算の経過を行番号に対応させて保存する行列 (シンボルリスト) を用いることとする。シンボルリストについては ([5, 6]) において提案されている。

シンボルリストには n 行 3 列の行列を用い、0 列目・1 列目には数値もしくはシンボル、2 列目には演算を保存する。もし、シンボルが表すものが実数であれば、0 列目に実数、1 列目と 2 列目に -1 を保存し、演算ではないことを示す。シンボル番号は、IS 演算を行うごとに新しい番号をつけていく。例えば、

$$\begin{aligned}\frac{1}{3}+\frac{1}{7}\times\frac{2}{9} &= [[0.333,0.334],0]+[[0.142,0.143],1]\times[[0.222,0.223],2]\\ &= [[0.333,0.334],0]+[[0.031524,0.031889],3]\\ &= [[0.364524,0.365889],4]\end{aligned}$$

の計算経過をシンボルリストに保存する場合、

$$SL = \begin{array}{c} \\ 0\\1\\2\\3\\4\end{array}\begin{array}{c}\begin{array}{ccc}0 & 1 & 2\end{array}\\ \begin{pmatrix}\frac{1}{3} & -1 & -1\\ \frac{1}{7} & -1 & -1\\ \frac{2}{9} & -1 & -1\\ 1 & 2 & \times\\ 0 & 3 & +\end{pmatrix}\end{array}$$

となる。

しかし、入力実数が整数以外の場合、数式処理システム以外では入力実数をそのまま保存することは難しい。そのため、分数 (有理数) については、0 行目に分子、1 行目に分母を保存するように変更した。

例えば、前述の計算経過をシンボルリストに保存する場合、

$$SL = \begin{array}{c} \\ 0\\1\\2\\3\\4\end{array}\begin{array}{c}\begin{array}{ccc}0 & 1 & 2\end{array}\\ \begin{pmatrix}1 & 3 & -1\\ 1 & 7 & -1\\ 2 & 9 & -1\\ 1 & 2 & \times\\ 0 & 3 & +\end{pmatrix}\end{array}$$

となる。

$\sqrt{2}$ や π などの無理数については、評価の際の計算が正確に行えないため、今回は考えないこととした。

3 行列を用いたISCZ法のアルゴリズム実装

3.1 実装の考え方

ここでは、実装の際の考え方について説明を行う。なお、説明を簡単にするためにシンボルを付加せず、区間演算のみを行うものとする。

今までの計算法では、例えば $\frac{1}{3}x^2 \times \frac{1}{7}x$ を計算する場合は以下の流れで行われていた。

$$\begin{aligned} & \frac{1}{3}x^2 \times \frac{1}{7}x \\ \rightarrow\ & [0.333, 0.334]x^2 \times [0.142, 0.143]x \\ =\ & [0.333, 0.334][0.142, 0.143]x^3 \\ =\ & [0.047286, 0.047762]x^3 \end{aligned}$$

つまり、

- 次数ごとにまとめる計算
- 1つの次数に区間係数が2つ存在したら区間係数を計算

の順番に計算を行い、答えとなる区間係数を出力していた。しかし、これは数式的な書き方・計算の仕方のため、数式処理システム以外では実装が難しいことが問題としてあげられる。

そこで、数式処理システム以外の言語でも実装が容易な新しい計算法を考えた。今までの計算法と同様に、新しい計算法で $\frac{1}{3}x^2 \times \frac{1}{7}x$ を計算する場合の流れを以下に示す。

$$\begin{aligned} & \frac{1}{3}x^2 \times \frac{1}{7}x \\ A\ =\ & \frac{1}{3}x^2 \qquad\qquad B = \frac{1}{7}x \\ A\ =\ & 2\ \begin{pmatrix} 0 & 1 \\ 0.333 & 0.334 \end{pmatrix} \qquad B = 1\ \begin{pmatrix} 0 & 1 \\ 0.142 & 0.143 \end{pmatrix} \\ A \times B\ =\ & [A_{2,0} \times B_{1,0}, A_{2,1} \times B_{1,1}] \\ =\ & [0.333 \times 0.142, 0.334 \times 0.143] \\ =\ & [0.047286, 0.047762] \rightarrow C_{2+1} = C_3 \\ C\ =\ & 3\ \begin{pmatrix} 0 & 1 \\ 0.0472896 & 0.047762 \end{pmatrix} \\ \rightarrow\ & [0.047286, 0.047762]x^3 \end{aligned}$$

つまり、

- 各次数の係数を行列に保存
  → n 次の項の係数を n 行目に
- 行列の要素を取り出して計算

という流れである。この方法は、多次元配列が実装可能な言語であれば実装は容易である。詳細な区間の保存方法や演算の方法は以降の節で説明する。

3.2 区間化

区間化は入力実数 x に対し $a \leq x \leq b$ となる a,b ならばどのような a,b を用いても良い。しかし、a,b が x から離れるほど区間内に 0 が含まれやすくなり、ゼロ書き換え・シンボル判定の回数が増えてしまう。よって、今回は小数点以下精度桁 +1 桁目を切り捨て・切り上げしたものを区間の下界・上界とした。この区間を行列に保存する際には、下界を行列の 0 列目に、上界を 1 列目に、元の実数を表すシンボルを 2 列目に保存することにする。

[1 変数多項式 $f(x)$ の精度桁 n 桁での区間化の流れ]

Input:1 変数多項式 $f(x)$, 精度桁 n
Output:区間係数を保存する行列 A

$i=0$ から $f(x)$ の次数まで i を増やしながら以下を繰り返す

$$\begin{aligned} r &= f(x) \text{ の } x^i \text{の係数} \\ A_{i,0} &= \frac{\lfloor r \times 10^n \rfloor}{10^n} \\ A_{i,1} &= \frac{\lceil r \times 10^n \rceil}{10^n} \\ A_{i,2} &= sn \end{aligned}$$

$\lfloor x \rfloor$ は x 以下の最大の整数を、$\lceil x \rceil$ は x 以上の最小の整数を表す。
sn はシンボル番号。

[1 変数多項式 $\frac{1}{3}x^2+\frac{1}{7}x+\frac{2}{9}$ の精度桁 3 桁での区間化例]

Input:1 変数多項式 $\frac{1}{3}x^2+\frac{1}{7}x+\frac{2}{9}$, 精度桁 3

$$\begin{aligned} r &= f(x) \text{ の } x^0 \text{の係数} = \frac{2}{9} \\ A_{0,0} &= \frac{\lfloor \frac{2}{9} \times 10^3 \rfloor}{10^3} = \frac{\lfloor 0.22222\ldots \times 10^3 \rfloor}{10^3} \\ &= \frac{\lfloor 222.22\ldots \rfloor}{10^3} = \frac{222}{10^3} = 0.222 \\ A_{0,1} &= \frac{\lceil \frac{2}{9} \times 10^3 \rceil}{10^3} = 0.223 \\ r &= f(x) \text{ の } x^1 \text{の係数} = \frac{1}{7} \\ &\vdots \end{aligned}$$

$$Output : A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0.222 & 0.223 & 0 \\ 0.142 & 0.143 & 1 \\ 0.333 & 0.334 & 2 \end{pmatrix} \end{array}$$

3.3 区間演算

区間演算は区間が保存されている行列の要素を取り出して行う。

[加算・減算の場合]

Input:$A = \frac{1}{3}, B = \frac{1}{7}$ 精度桁 n
Output:$C = A + B$

$$
\begin{array}{rcl}
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \qquad \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
A & = & 0 \begin{pmatrix} 0.333 & 0.334 & 1 \end{pmatrix} \quad B = 0 \begin{pmatrix} 0.142 & 0.143 & 2 \end{pmatrix} \\
i & = & \text{計算する次数} = 0 \\
C_{i,0} = C_{0,0} & = & A_{0,0} + B_{0,0} = 0.333 + 0.142 = 0.475 \\
C_{i,1} = C_{0,1} & = & A_{0,1} + B_{0,1} = 0.334 + 0.143 = 0.477 \\
C_{i,2} = C_{0,2} & = & sn = 3 \\
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
C & = & 0 \begin{pmatrix} 0.475 & 0.477 & 3 \end{pmatrix}
\end{array}
$$

sn はシンボル番号。

減算も同様なので省略する。

[乗算の場合]

Input:$A = \frac{1}{3}x^2, B = \frac{1}{7}x$ 精度桁 n
Output:$C = A \times B$

$$
\begin{array}{rcl}
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \qquad \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
A & = & 2 \begin{pmatrix} 0.333 & 0.334 & 1 \end{pmatrix} \quad B = 1 \begin{pmatrix} 0.142 & 0.143 & 2 \end{pmatrix} \\
i & = & \text{計算する } A \text{ の次数} = 2 \\
j & = & \text{計算する } B \text{ の次数} = 1 \\
C_{i+j,0} = C_{3,0} & = & A_{2,0} \times B_{1,0} = 0.333 \times 0.142 = 0.047286 \\
C_{i+j,1} = C_{3,1} & = & A_{2,1} \times B_{1,1} = 0.334 \times 0.143 = 0.047762 \\
C_{i+j,2} = C_{3,2} & = & sn = 3 \\
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
C & = & 3 \begin{pmatrix} 0.047286 & 0.047762 & 3 \end{pmatrix}
\end{array}
$$

sn はシンボル番号。

[除算の場合]

Input:$A = \frac{1}{3}x^2, B = \frac{1}{7}x$ 精度桁 n
Output:$C = \frac{A}{B}$

$$
\begin{array}{rcl}
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \qquad \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
A & = & 2\begin{pmatrix} 0.333 & 0.334 & 1 \end{pmatrix} \quad B = 1\begin{pmatrix} 0.142 & 0.143 & 2 \end{pmatrix} \\
i & = & \text{計算する } A \text{ の次数} = 2 \\
j & = & \text{計算する } B \text{ の次数} = 1 \\
C_{i-j,0} = C_{1,0} & = & \dfrac{A_{2,0}}{B_{1,1}} = \dfrac{0.333}{0.143} = 2.328671 \\
C_{i-j,1} = C_{1,1} & = & \dfrac{A_{2,1}}{B_{1,0}} = \dfrac{0.334}{0.142} = 2.352112 \\
C_{i-j,2} = C_{1,2} & = & sn = 3 \\
 & & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\
C & = & 1\begin{pmatrix} 2.328671 & 2.352112 & 3 \end{pmatrix}
\end{array}
$$

sn はシンボル番号。

3.4 ゼロ書き換え

ゼロ書き換えを行う条件は「区間の下界と上界の間に 0 を含むこと」であるが、実装では「区間の下界と上界の積が 0 以下であること」とする。

$$X = [[a, b], s] \text{ のとき}$$
$$(a \leq 0)\ and\ (0 \leq b) \Leftrightarrow ab \leq 0$$

このとき、$ab \leq 0$ を満たすならば、シンボル s を用いて本当に 0 であるかの評価を行う。

3.5 シンボルリストの評価方法

シンボルリストの評価手法については昨年 [3] も記載したが、確認のため再度記載する。

3.5.1 行列法

2.3 節で保存したシンボルリストを用いて、シンボル番号 4 を評価する手順によって説明する。

$$
SL = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c}
\begin{array}{ccc} 0 & 1 & 2 \end{array} \\
\begin{pmatrix}
\frac{1}{3} & -1 & -1 \\
\frac{1}{7} & -1 & -1 \\
\frac{2}{9} & -1 & -1 \\
1 & 2 & \times \\
0 & 3 & +
\end{pmatrix}
\end{array}
$$

流れとしては、シンボルリストから評価行列を作り、その評価行列を用いて計算することになる。

各行の0列目にはその行にいくつのシンボルリストから取った3つ組が存在するかを保存する。例えば0行目には評価を行うシンボル番号4に対応するシンボルリストの行のみを入れるため、3つ組の数は1となり、0行0列目に1が保存される。

$$eva = \begin{matrix} & 0 & 1 & 2 & 3 \\ 0 & \begin{pmatrix} 1 & 0 & 3 & + \end{pmatrix} \end{matrix}$$

eva行列のn行$3m+3$列目$(m \geq 0, m \in \mathbb{Z})$を読み、演算記号ならばシンボルリストから$3m$列目・$3m+1$列目のシンボルに対応するシンボルリストの行を$n+1$行目に左詰めで入れる。演算記号ではなく$-1$ならば何も行わずスキップする。$n$行目の全ての列が終わったら$n+1$行目以降にも同様の操作を行い、$n$行目に演算シンボルがなくなるまで繰り返す。

$$eva = \begin{matrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & \begin{pmatrix} 1 & 0 & 3 & + & & & \end{pmatrix} \\ 1 & \begin{pmatrix} 2 & \frac{1}{3} & -1 & -1 & 1 & 2 & \times \end{pmatrix} \end{matrix}$$

$$eva = \begin{matrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & \begin{pmatrix} 1 & 0 & 3 & + & & & \end{pmatrix} \\ 1 & \begin{pmatrix} 2 & \frac{1}{3} & -1 & -1 & 1 & 2 & \times \end{pmatrix} \\ 2 & \begin{pmatrix} 2 & \frac{1}{7} & -1 & -1 & \frac{2}{9} & -1 & -1 \end{pmatrix} \end{matrix}$$

評価行列が完成してから演算を行う。n行の評価行列での演算はn行目,$n-1$行目,$\cdots$,0行目の順に行い、演算シンボルがあれば1行下に保存されている実数を用いて計算を行うこととする。計算結果は演算シンボルが属する3つ組に上書きする。

$$eva = \begin{matrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & \begin{pmatrix} 1 & 0 & 3 & + & & & \end{pmatrix} \\ 1 & \begin{pmatrix} 2 & \frac{1}{3} & -1 & -1 & 1 & 2 & \times \end{pmatrix} \\ 2 & \begin{pmatrix} 2 & \frac{1}{7} & -1 & -1 & \frac{2}{9} & -1 & -1 \end{pmatrix} \end{matrix}$$

$$\downarrow$$

$$eva = \begin{matrix} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & \begin{pmatrix} 1 & 0 & 3 & + & & & \end{pmatrix} \\ 1 & \begin{pmatrix} 2 & \frac{1}{3} & -1 & -1 & \frac{2}{63} & -1 & -1 \end{pmatrix} \\ 2 & \begin{pmatrix} 2 & \frac{1}{7} & -1 & -1 & \frac{2}{9} & -1 & -1 \end{pmatrix} \end{matrix}$$

また、同じシンボルを複数回評価するのは無駄であるため、一度評価されたシンボルはシンボルリストに実数として上書きする。

$$SL = \begin{matrix} & 0 & 1 & 2 \\ 0 & \frac{1}{3} & -1 & -1 \\ 1 & \frac{1}{7} & -1 & -1 \\ 2 & \frac{2}{9} & -1 & -1 \\ 3 & 1 & 2 & \times \\ 4 & 0 & 3 & + \end{matrix} \rightarrow SL = \begin{matrix} & 0 & 1 & 2 \\ 0 & \frac{1}{3} & -1 & -1 \\ 1 & \frac{1}{7} & -1 & -1 \\ 2 & \frac{2}{9} & -1 & -1 \\ 3 & \frac{2}{63} & -1 & -1 \\ 4 & 0 & 3 & + \end{matrix}$$

この操作を繰り返し、0行目が計算された時の0行1列目が評価結果となる。

3.5.2 式列法

式列法の基本形は次のようになる。

$$eva = \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ \left(\star \right. & s & \text{“}s\text{”} & t & \text{“}s\text{”} & \left. u \right) \end{array}$$

ここで、s,t,u はシンボル番号を、“s” は次の数がシンボルであることを、$\star$ は演算記号を表す。この行列を 0 列目から走査し、以下の条件に当てはまったら操作を加える。

- n 列目が “s” かつ $n+1$ 列目のシンボルが演算を表すとき
  n 列目を起点として基本形に展開する。$n+2$ 列目以降を右に 4 つシフトする。

$$eva = \begin{array}{cccccccc} \cdots & n-2 & n-1 & n & n+1 & n+2 & n+3 & \cdots \\ \left(\cdots \right. & \star & s & \text{“}s\text{”} & t & \text{“}s\text{”} & u & \left. \cdots \right) \end{array}$$

$$\downarrow$$

$$eva = \begin{array}{cccccccccc} \cdots & n-2 & n-1 & n & n+1 & n+2 & n+3 & n+4 & n+5 & n+6 & \cdots \\ \left(\cdots \right. & \star & s & \star & t & \text{“}s\text{”} & v & \text{“}s\text{”} & w & \text{“}s\text{”} & \left. \cdots \right) \end{array}$$

- n 列目が “s” かつ $n+1$ 列目のシンボルが実数を表すとき
  シンボルを表す “s” を実数を表す “r” に変え、後ろに実数を入れる。

$$eva = \begin{array}{cccccccc} \cdots & n & n+1 & n+2 & n+3 & n+4 & n+5 & \cdots \\ \left(\cdots \right. & \star & s & \text{“}s\text{”} & t & \text{“}s\text{”} & u & \left. \cdots \right) \end{array}$$

$$\downarrow$$

$$eva = \begin{array}{cccccccc} \cdots & n & n+1 & n+2 & n+3 & n+4 & n+5 & \cdots \\ \left(\cdots \right. & \star & s & \text{“}r\text{”} & p & \text{“}s\text{”} & u & \left. \cdots \right) \end{array}$$

- $n+2$ 列目と $n+4$ 列目が共に “r” のとき
  演算を行い、$n+6$ 列目以降を左に 4 つシフトする。

$$eva = \begin{array}{cccccccccc} \cdots & n & n+1 & n+2 & n+3 & n+4 & n+5 & n+6 & n+7 & \cdots \\ \left(\cdots \right. & \star & s & \text{“}r\text{”} & p & \text{“}r\text{”} & q & \text{“}s\text{”} & u & \left. \cdots \right) \end{array}$$

$$\downarrow$$

$$eva = \begin{array}{cccccc} \cdots & n & n+1 & n+2 & n+3 & \cdots \\ \left(\cdots \right. & \text{“}r\text{”} & r & \text{“}s\text{”} & u & \left. \cdots \right) \end{array}$$

また、演算後は行列法と同様にシンボルリストにも演算結果を上書きする。

この操作を繰り返し、0 列目が “r” になった時の 1 列目が評価結果となる。

4 実験

この実験で使用するPCおよび実行環境は以下の通り。

使用コンピュータ
OS:Windows 7 Professional
CPU:Intel(R) Pentium(R) CPU G840 @ 2.80GHz
メモリ:4.00GB
使用ソフト:数式処理ソフト Maple14
JavaScriptを動作させるブラウザ:Chrome38.0.2125.104m

4.1 スツルムアルゴリズムへの適用

4.1.1 実験方法

- 1変数多項式に対してスツルムアルゴリズム[7]を適用し、その実解の個数を計算する。
- 以下の3つの方法で計算開始から出力までの時間・メモリ使用量を計測する。ISCZ法については評価行列の最大要素数も計測する。
  1. ISCZ法のスツルムアルゴリズムに、行列法の評価方法を組み込んだもの(M行列)
  2. ISCZ法のスツルムアルゴリズムに、式列法の評価方法を組み込んだもの(M式列)
  3. 自作の正確演算でのスツルムアルゴリズム(M厳密)
  4. JavaScriptで実装したISCZスツルムアルゴリズムに、式列法の評価方法を組み込んだもの(JS)
- 入力する多項式は実解の個数を指定し、その実解および重複度はランダムとして生成した
- ISCZ法の開始精度は2桁とし、0判定での評価の結果「本当は0でない」と判断した場合は、精度桁を1桁ずつ増やす

スツルムアルゴリズムでは数式処理システム以外の言語としてJavaScriptを用いて実験を行った。JavaScriptを選択した理由は、

- webブラウザさえあればPCのOSによらず実行が可能である
- 動的配列の宣言が容易である

からである。

2番目の理由にある「動的配列」とは変数宣言時に配列の要素数を定義しなくてもよい配列のことである。今回のISCZ法では、必要なシンボルリストの行数が不明となっている。そのため、変数宣言時に要素数を定義しなければならない「静的配列」で実装する場合、シンボルリスト行列の宣言時に巨大な領域を持つように宣言することとなる。その場合、メモリ使用量が増えてしまうため、動的配列の宣言が容易な言語を用いるのがよいと考えた。

表 1: スツルム実験結果

入力次数	出力精度	ゼロ判定数	シンボルリスト長	実行時間 (秒)			
				M 行列	M 式列	M 厳密	JS
4	3(4)	23	45	0.031	0.015	0.016	0.016
5	4(5)	48	86	0.063	0.031	0	0.016
7	5	86	144	0.093	0.078	0.015	0.016
10	7	280	288	0.328	0.296	0.016	****

- 4・5 次式入力では Maple と JS で出力精度が異なったため、JS での精度を () 内に記載
- 10 次式においては JS で出力が得られなかった。詳細は 4.1.2 項に記載

4.1.2 実験結果

実験結果を表 1 に記載した。Maple と JavaScript で出力精度が異なっているが、今回は正しい出力をしているか・どの程度の時間がかかったかに重点を置いたため無視した。

Maple の安定化演算では厳密計算のほうが短時間で計算できたが、JavaScript で実装した安定化演算は 7 次式までにおいて Maple の厳密計算とほぼ同じ時間で計算を終えた。

JavaScript においては、小数は小数点以下 15 桁で切り捨てる仕様となっている。そのため、各次数の係数が小さく (10^{-15} 程度以下で) 区間化が不可能な場合や、0 判定での評価の結果において「0 でない」が続き、精度桁が 15 桁以上となった場合には、正確な出力を行うことが不可能となる。

4.2 グラハムアルゴリズムへの適用

4.2.1 実験方法

- ランダムに生成した点 (x,y) の集合の凸包をグラハムアルゴリズム [8] を用いて計算する
- 以下のそれぞれの方法について、計算開始から出力までの時間を計測する
  1. ISCZ 法のグラハムアルゴリズムに、行列法の評価方法を組み込んだもの (M 行列)
  2. ISCZ 法のグラハムアルゴリズムに、式列法の評価方法を組み込んだもの (M 式列)
  3. 自作の正確演算でのグラハムアルゴリズム (M 厳密)
- 入力する点集合は以下のそれぞれの中から指定した点数をランダムに生成する
  1. 有 A/x, y : $\frac{1 \sim 1000}{3 \sim 100}$ $\quad x^2 + y^2 < 100$
  2. 有 B/x, y : $\frac{1 \sim 1000}{3 \sim 100}$ $\quad x^2 + y^2 < 100 \quad y > \frac{x}{2} \quad y < 2x$
  3. 無 C/x, y : $\sqrt{\frac{1 \sim 10000}{3 \sim 100}}$ $\quad x^2 + y^2 < 100$
  4. 無 D/x, y : $\sqrt{\frac{1 \sim 10000}{3 \sim 100}}$ $\quad x^2 + y^2 < 100 \quad y > \frac{x}{2} \quad y < 2x$

表 2: グラハム実験結果 (有理数点)

実験条件	入力点数	出力精度	ゼロ判定数	シンボルリスト長	実行時間 (秒)		
					M 行列	M 式列	M 厳密
有 A	250	5	3	128382	3.400	3.542	0.281
有 A	500	6	4	506840	23.743	24.305	1.623
有 A	750	6	4	1135340	51.418	52.775	2.730
有 A	1000	7	5	2013854	98.514	99.591	6.334
有 B	250	6	4	128368	5.663	5.803	0.281
有 B	500	7	5	506882	37.752	37.518	1.747
有 B	750	6	4	1135361	31.450	31.543	2.855
有 B	1000	7	5	2013861	97.142	97.641	6.786
有 B	1125	7	5	2546854	78.313	78.530	6.225

表 3: グラハム実験結果 (無理数点)

実験条件	入力点数	出力精度	ゼロ判定数	シンボルリスト長	実行時間 (秒)		
					M 行列	M 式列	M 厳密
無 C	100	5	3	21310	1.669	1.388	232.832
無 C	200	5	3	82682	5.616	5.990	900.953
無 C	300	5	3	184068	9.141	9.189	1903.212
無 C	500	6	4	506861	22.683	22.791	7913.276
無 D	100	4	2	21310	1.498	1.466	237.091
無 D	200	5	3	82689	6.147	5.413	1138.683
無 D	300	5	3	184075	12.121	12.059	3124.903
無 D	500	6	4	506854	16.068	16.240	7876.396

入力を扇形にすることで、凸包が半径の直線に近づき、低い精度において、グラハムアルゴリズムの点を残すか・削除するかの判定で誤判定を生じやすくなる。

- ISCZ 法の開始精度は 2 桁とし、0 判定での評価の結果「本当は 0 でない」と判断した場合は、精度桁を 1 桁ずつ増やす

4.2.2 実験結果

有理数点での計算 (表 2) は、厳密計算の方が ISCZ 法より高速に計算することができた。表に載せた点数以上も実験を行おうとしたが、Object Too Large エラーが発生し、計算を終えることができなかった。これは点数が多くなることで計算回数が多くなり、シンボルリスト行列が膨張したことが原因であると考えられる。

一方、無理数点での計算 (表 3) においては、ISCZ 法が厳密計算と比べ高速に計算できることが確認できた。こちらについては厳密計算の計算時間の関係で 500 点でやめてしまったが、計算時間の増え方から、点

数を増やすほど有効性が増すと考えられる。また、今回使用した無理数は $\sqrt{n}$ のみであるので、今後は e や π などを含めた実験も行う。

5 おわりに

今回の研究では、スツルムアルゴリズム、グラハムアルゴリズムにISCZ法を組み込み、同手法の有効性を検証した。その結果、スツルムアルゴリズムについては、厳密計算の方が高速に計算できることがわかった。また、グラハムアルゴリズムについては、有理数点では厳密計算の方が高速に計算できたが、無理数点ではISCZ法の方が高速に計算できた。しかし、シンボルリストの膨張により計算出来なくなることがあったため、シンボルの保存方法についても新たな方法を考えたい。

また、数式処理システムのみで実装されてきたISCZ法を用いたアルゴリズムを、数式処理システムではないJavaScriptで実装することに成功した。しかし、「実験可能なのは有理数のみである」ことや、「小数点以下15桁しか保存できない」という制限があるため、この制限をなくす、もしくは広げる工夫をすることが求められる。さらに、C言語などの他言語でも実装し実験を行いたい。

謝辞

今回の研究を行うにあたり東京理科大学 関川浩教授より多くの助言を頂きました。この場をお借りして感謝申し上げます。

参 考 文 献

[1] 白柳 潔, 関川 浩: 安定化理論における台収束の応用について, 京都大学数理解析研究所講究録 **1568**, 2007, 20–26

[2] 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

[3] 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, 京都大学数理解析研究所講究録 **1907**, 2014, 71–79

[4] 白柳 潔, 関川 浩: 安定化理論に基づくISCZ法の凸包構成への応用, 京都大学数理解析研究所講究録 **1815**, 2012, 133–142

[5] 白柳 潔, 関川 浩: 安定化理論に基づくISCZ法の有効性について, 京都大学数理解析研究所講究録 **1814**, 2012, 29–35

[6] 白柳 潔, 関川 浩: 安定化理論に基づくlog methodについて, 京都大学数理解析研究所講究録 **1666**, 2009, 98–105

[7] 高木 貞治:「第3章 スツルムの問題, 根の計算」, 『代数学講義 改訂新版』, 共立出版, 1965, 81–119

[8] R.L.Graham: An efficient algorithm for determining the convex hull of a finite planar set. , Information Processing Letters, **1(4)**, 1972, 132–133