アルゴリズム 77－2
（2001．3．12）

ボロノイ図構成時の幾何学的条件判定式の誤差の見積もり

神田 毅
東京大学大学院 工学系研究科 計数工学専攻

概要

条件分岐のための符号判定式の計算を浮動小数点演算で行なうと、計算誤差のためにその符号判定を誤り、アルゴリズムでは考慮していない事態が起こる可能性があり、問題が解けなくなる。それを避けるために、計算誤差の生じない多倍長整数演算を採用することができるが、そうすると計算コストが高い。そこで、浮動小数点演算で符号判定式の計算を行なった結果、その値が0とどの程度離れていればその符号判定結果を信用して良いのかを見積もり、浮動小数点演算による判定で済むところは済ませるという方針が考えられる。実際はその種の方法に変種がいくつか考えられるので、問題の規模、入力の性質などに依存して、それらの変種の中でどの解き方が適するのかを探る。

Error Estimation of Floating-Point Expressions for Voronoi Diagram Construction

Takeshi Kanda
Department of Mathematical Engineering and Information Physics, Graduate School of Engineering, University of Tokyo

Abstract

A widely used strategy to decrease the execution time of robust geometric algorithms using the multiprecision integer arithmetic is the hybrid use of integer and floating-point arithmetic. In this strategy, the multiprecision integer arithmetic is avoided if the sign decision in the floating-point arithmetic used in advance is regarded as sufficiently reliable. For this strategy, it is important to estimate the upper bound of the error calculated in floating-point arithmetic. However, the method of the estimation is not unique. The more tightly the error is estimated, the more frequently the multiprecision integer arithmetic is avoided. However, a tight upper bound requires high computational cost, and hence there is a trade-off between tightness of the upper bound and the cost to compute the upper bound. This paper investigates how the suitable estimation varies as the type and the scale of inputs vary. Experimental consideration is done using the incremental method for constructing the Voronoi diagram as an example of the geometric algorithm.

1 はじめに

幾何アルゴリズムを素直にプログラミングして実行すると、破綻することが多い。なぜなら、条件分岐のための符号判定式の計算を、計算精度が有限であり、表現できる数値も実数全体ではないという限界を持つ浮動小数点演算で行なうと、入力自体の誤差や計算誤差のためにその符号判定を誤り、アルゴリズムでは考慮していない事態が起こる可能性があるからである。プログラムの書き方にもよるが、この時に異常終了したり、無限ループに陥ったりする。そのようなことが起こらなかった場合でも、構成問題では、得た出力の位相構造が現実にはあり得ないものであったり、検出問題では、用いた数値計算法の分解能を考えれば当然発見できるはずと思われるような部分でさえ見落としたりする可能性もある。このような問題点は、いろいろな方面から解決されようとしている。その方針としては、以下のものが考えられる。

- **位相優先法:** 数値計算による幾何学的条件判定は、得られる結果の位相構造が矛盾したものにならない範囲でしか用いない [1] [4] [6] [7] [8] [9] [10] [13]。

- **整数帰着法**: 全ての幾何学的条件判定を誤差の生じない多倍長整数演算で行ない [1] [4] [5] [10] [11] [12] [13] [14]、例外的状況 (条件分岐のための符号判定式の計算結果が 0 になる場合) は、**記号摂動法** [1] [2] [3] [4] [5] [11] [12] で処理する。

- **混合演算**: 浮動小数点演算による幾何学的条件判定式の誤差の上限を見積もり、得られた結果が信頼できる場合だけにその結果を用いることによって、高コストの多倍長整数演算はなるべく避け、信頼できない時のみ多倍長整数演算を用いる [11] [12] [14]。

本論文では、幾何アルゴリズムの例としてはボロノイ図を構成するための逐次添加法を考え、上記の「混合演算」という方針について細かく考察する。「幾何学的条件判定式の誤差の見積もり」とは、ここでは「条件分岐のための符号判定式の計算を浮動小数点演算で行なった時、その値がどの程度 0 から離れていれば、符号判定結果を信頼して良いかを計算すること」という意味である。ただし、そのような見積もり方法には、厳しいものから甘いものまでいくつも変種が考えられる。そして一般には、見積もりが厳しくなればなる程、見積もり自体の計算には時間を取られるが、高コストの多倍長整数演算を避けることのできる可能性は高まるという、トレードオフがある。よって、入力の規模・性質などに依存して、それらの変種の中でどの解き方が適するのかを探る。

2 ボロノイ図

2 次元の**ボロノイ図 (Voronoi diagram)** [15] とは、平面上に与えられた点集合

$$\{P_i(x_i,\ y_i) \mid i = 1,\ 2,\ \ldots,\ N\} \tag{1}$$

の中で、どの点に最も近いかによって平面を分割して作った図形で、図 1 がその例である。式で表すなら、

$$\mathcal{V}(P_i) = \bigcap_{j \neq i} \{P(x,\ y) \mid d_i(P) \leq d_j(P)\}, \tag{2}$$

$$d_i(P) = (x - x_i)^2 + (y - y_i)^2 \tag{3}$$

で定義される勢力圏の集合 $\{\mathcal{V}(P_i) \mid i = 1,\ 2,\ \ldots,\ N\}$ によって、平面を分割した図形ということになる。なお、ボロノイ図を作る時に与える点 $P_i(x_i,\ y_i)$ を母点と呼び、ボロノイ図に現れる頂点をボロノイ頂点、ボロノイ図に現れる辺をボロノイ辺、領域 $\mathcal{V}(P_i)$ を母点 P_i に対応するボロノイ領域と呼ぶ。

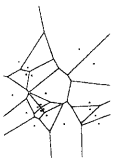

図 1: ボロノイ図の例

3 逐次添加法によるボロノイ図構成

3.1 概略

ボロノイ図構成法の 1 つに、**逐次添加法 (incremental method)** がある [15] [16]。2 個の母点のボロノイ図から始めて、母点を 1 個ずつ添加しながらボロノイ図を更新していく方法である。

この方法では、データ構造を工夫して母点の添加順序を適切に変えることにより、入力母点数 N の様々なタイプの入力に対して、平均的に $O(N)$ の計算量でボロノイ図を構成できる [17] [18] [19] [20]。ただし、最悪の場合の計算量は $O(N^2)$ となる。他の多くの方法では、平均計算量、最悪計算量ともに $O(N \log N)$ となる。逐次添加法で必要となる幾何学的条件判定には、以下の 2 種類がある。

3.2 幾何学的条件判定 1 - 近い点の判定

新母点を追加して古いボロノイ図を更新する際に、まずその新母点が古いボロノイ図の中のどのボロノイ領域に属すかを知る必要がある。どのボロノイ領域に属すかを求めることは、どの母点に最も近いかを求めることと同じことである。この手順を**点位置決定 (point location)** という。図 2 のように、点位置決定をしたい新母点を $(x_n,\ y_n)$、すでに調べた点の中で新母点に最も近い点を $(x_i,\ y_i)$、新しい最近点の候補を $(x_j,\ y_j)$ とする。$(x_i,\ y_i)$ と $(x_j,\ y_j)$ のうちでどちらがより $(x_n,\ y_n)$ に近いかを判定する必要があるが、それは

$$\begin{aligned} D \ \equiv\ & \{(x_j - x_n)^2 + (y_j - y_n)^2\} \\ & - \{(x_i - x_n)^2 + (y_i - y_n)^2\} \end{aligned} \tag{4}$$

という 2 次式の符号によって決まる。この式は

$$\begin{aligned} D' \ \equiv\ & (x_j - x_i)\{(x_j - x_n) + (x_i - x_n)\} \\ & + (y_j - y_i)\{(y_j - y_n) + (y_i - y_n)\} \end{aligned} \tag{5}$$

と同じ値を取る。以後の実験では式 (5) を使う。図 2 に示すように、$(x_n,\ y_n)$ の位置に応じて D の符号が変わる。

3.3 幾何学的条件判定 2 - 外接円の内外判定

逐次添加の主要部では、新母点のボロノイ領域によって、古いボロノイ図の中のあるボロノイ頂点が消されるかどう

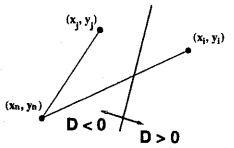

図 2: 近い点の判定

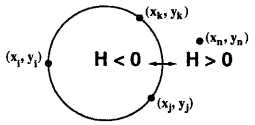

図 3: 外接円の内外判定

かを判定する必要がある。図3のように、これから添加したい新母点を $(x_n,\ y_n)$、消されるかどうかを判定したいボロノイ頂点のまわりの3領域の3母点の座標を $(x_i,\ y_i)$、$(x_j,\ y_j)$、$(x_k,\ y_k)$ とする。まず、これら3点を通る外接円の方程式は

$$\begin{vmatrix} 1 & x_i & y_i & x_i^2+y_i^2 \\ 1 & x_j & y_j & x_j^2+y_j^2 \\ 1 & x_k & y_k & x_k^2+y_k^2 \\ 1 & x & y & x^2+y^2 \end{vmatrix} = 0 \qquad (6)$$

である。なぜなら、左辺は x^2 の係数と y^2 の係数が等しく xy の項のない2次式なので、この方程式は円を表し、$(x,\ y) = (x_i,\ y_i),\ (x_j,\ y_j),\ (x_k,\ y_k)$ を代入した時には等式が成り立つので、これら3点を通ることもわかるからである。新母点 $(x_n,\ y_n)$ のボロノイ領域によって、あるボロノイ頂点が消されるかどうかは、この新母点が先ほどの外接円の中にあるか外にあるかで決まる。なぜなら、新母点がこの外接円の中にあることは、外接円を作る3母点よりも、この新母点の方がボロノイ頂点(外接円の中心)に近いことを意味するからである。つまり

$$H \equiv \begin{vmatrix} 1 & x_i & y_i & x_i^2+y_i^2 \\ 1 & x_j & y_j & x_j^2+y_j^2 \\ 1 & x_k & y_k & x_k^2+y_k^2 \\ 1 & x_n & y_n & x_n^2+y_n^2 \end{vmatrix} \qquad (7)$$

が負ならば $(x_n,\ y_n)$ が3母点の外接円内にあることになるから、ボロノイ頂点が消されることになる。ただし、外接円に沿って x 軸から y 軸へ 90° の回転をさせる回転方向(通常の座標軸の書き方をした時の反時計周り)で、$(x_i,\ y_i)$、$(x_j,\ y_j)$、$(x_k,\ y_k)$ の順に外接円上に並んでいるものとする。この式は

$$H' \equiv -\begin{vmatrix} x_n-x_i & y_n-y_i & (x_n-x_i)^2+(y_n-y_i)^2 \\ x_n-x_j & y_n-y_j & (x_n-x_j)^2+(y_n-y_j)^2 \\ x_n-x_k & y_n-y_k & (x_n-x_k)^2+(y_n-y_k)^2 \end{vmatrix} \qquad (8)$$

と同じ値を取る。以後の実験では式(8)を使う。図3に示すように、$(x_n,\ y_n)$ の位置に応じて H の符号が変わる。

4 幾何アルゴリズムでの多倍長整数演算利用法

図4の左上のように、入力される浮動小数点座標を $(x_i,\ y_i)$ $(i = 1,\ 2,\ \ldots,\ N)$ とする。また、多倍長整数演算において $[-L,\ L]\times[-L,\ L]$ の範囲の多倍長整数が扱えるものとする。ある範囲の多倍長整数が「扱える」とは、その範囲の多倍長整数の入力に対して、例えば式(4)、(7)などの式の値を、誤差なしで計算できるということである。そして、図4の右上のように、多倍長整数演算を行なうための多倍長整数座標 $(X_i,\ Y_i)$ $(i = 1,\ 2,\ \ldots,\ N)$ を

$$X_i = \lfloor ax_i + b_x \rfloor,\ Y_i = \lfloor ay_i + b_y \rfloor \qquad (9)$$

で得る。ただし、a、b_x、b_y は浮動小数点型の変数で、多倍長整数座標で表わされた点 $(X_i,\ Y_i)$ $(i = 1,\ 2,\ \ldots,\ N)$ が $[-L,\ L]\times[-L,\ L]$ の範囲になるべく広く分布するように選ばれる。以後は、$(x_i,\ y_i)$ の代わりに、これにごく近い点である $(\frac{X_i-b_x}{a},\ \frac{Y_i-b_y}{a})$ が真の入力であるとみなして、図4の右下のように、その入力に対するボロノイ図の位相構造を得る。最後に、図4の左下のように、その位相構造を元の浮動小数点座標の入力に対するボロノイ図の位相構造と同じとみなして、答えとする。もっとも、幾何学的条件判定を $(\frac{X_i-b_x}{a},\ \frac{Y_i-b_y}{a})$ に対して行なうことと $(X_i,\ Y_i)$ に対して行なうことは、全く同じ意味を持つので、判定には後者を用いる。それでも、多倍長整数座標 $(X_i,\ Y_i)$ を用いた条件判定式の計算は計算量が大きいので、可能な限り元の浮動小数点座標 $(x_i,\ y_i)$ を用いた条件判定で代用する。よって、浮動小数点座標 $(x_i,\ y_i)$ を用いた条件判定において、判定式の値が0からどの程度遠ければ、その符号が多倍長整数座標 $(X_i,\ Y_i)$ を用いた判定式の符号と一致すると言い切れるのかを調べる。以後は、$\frac{X_i-b_x}{a}$、$\frac{Y_i-b_y}{a}$ を真の値とみなした時の、元の値 x_i、y_i の相対誤差の上限を δ とする。この時

$$\begin{aligned} \frac{x_i - \frac{X_i-b_x}{a}}{M} &\le \frac{x_i - \frac{X_i-b_x}{a}}{\frac{X_i-b_x}{a}} \le \delta, \\ \frac{y_i - \frac{Y_i-b_y}{a}}{M} &\le \frac{y_i - \frac{Y_i-b_y}{a}}{\frac{Y_i-b_y}{a}} \le \delta \end{aligned} \qquad (10)$$

が成り立つ。M は、$|x_i|,\ |y_i| \le M$ $(i = 1,\ 2,\ \ldots,\ N)$ を満たす任意の正数である。

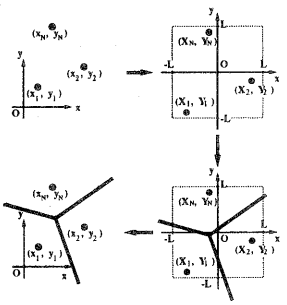

図 4: 幾何アルゴリズムでの多倍長整数型演算利用法

5 誤差の見積もり

5.1 基本方針

a をある式とした時、$|a|$ の上限を $\mathrm{Max}[a]$、a の絶対誤差の上限を $\mathrm{Error}[a]$ と表すことにする。以下では

$$\begin{aligned}
\mathrm{Max}[a \pm b] &= \mathrm{Max}[a] + \mathrm{Max}[b], && (11)\\
\mathrm{Error}[a \pm b] &= \mathrm{Error}[a] + \mathrm{Error}[b] + \mathrm{Max}[a \pm b]\ \delta\\
&= \mathrm{Error}[a] + \mathrm{Error}[b]\\
&\quad + (\mathrm{Max}[a] + \mathrm{Max}[b])\ \delta, && (12)\\
\mathrm{Max}[ab] &= \mathrm{Max}[a]\ \mathrm{Max}[b], && (13)\\
\mathrm{Error}[ab] &= \mathrm{Max}[a]\ \mathrm{Error}[b] + \mathrm{Max}[b]\ \mathrm{Error}[a]\\
&\quad + \mathrm{Max}[ab]\ \delta\\
&= \mathrm{Max}[a]\ \mathrm{Error}[b] + \mathrm{Max}[b]\ \mathrm{Error}[a]\\
&\quad + \mathrm{Max}[a]\ \mathrm{Max}[b]\ \delta && (14)
\end{aligned}$$

の性質を用いることにする。ただし、Max[] や Error[] の記号を使った時の等号 = は、「左辺の上限として右辺を使ってもよい」と解釈することにする。式(12)、(14)の右辺の第1項、第2項は、演算を行なう前に持っていた誤差が新しい計算結果に伝わるために生ずる誤差を表す。この誤差を**伝搬誤差 (propagation error)** という。第3項は得た計算結果を表す時に生じる誤差を表す。この誤差を**表現誤差 (representation error)** という。ただし、式(14)では絶対誤差の2次以上の項を無視している。それでも、$\mathrm{Error}[a]$ は $\mathrm{Max}[a]$ に比べて非常に小さいため、絶対誤差の見積もりの最終結果の値を少しだけ大きめにとれば、2次以上の項も含んだ見積もりとなる。ともかく、これらの性質を用いて条件判定式の絶対誤差の上限を見積もっていく。

5.2 幾何学的条件判定式の誤差の見積もりの結果

入力母点座標 $(x_i,\ y_i)\ (i = 1,\ 2,\ \ldots,\ n)$ に対して

$$\mathrm{Max}[x_i] = M,\ \mathrm{Max}[y_i] = M, \qquad (15)$$

$$\mathrm{Error}[x_i] = M\delta,\ \mathrm{Error}[y_i] = M\delta \qquad (16)$$

が成り立っていると仮定する。式(10)より、これは現実的な仮定である。近い点の判定式(5)に関しては

$$\mathrm{Error}[D'] \quad = \quad 112M^2\delta \qquad (17)$$

と見積もることができ、外接円の内外判定式(8)に関しては

$$\mathrm{Error}[H'] = 2944M^4\delta \qquad (18)$$

と見積もることができる[1]。

5.3 誤差の見積もりの利用法

例えば式(5)の D' 符号を知りたい時、まずは浮動小数点演算でこれを計算し、その値の絶対値が式(17)の $\mathrm{Error}[D']$ を越えたら、その符号は必ず正しいと判断して幾何アルゴリズムの処理を進める。越えなかったらその符号は信頼できないとみなして、多倍長整数演算を採用して符号を求める。

6 数値実験 1: 基本的な実験

6.1 浮動小数点演算・多倍長整数演算・混合演算使用時の実行時間比較

正方形の枠内に一様に発生させた 5000、10000、...、50000 点の母点に対するボロノイ図を構成するための計算時間を測定した。ただし、ボロノイ図の位相構造を得る所までで、ボロノイ頂点の座標を求める時間は含まれていない。ボロノイ頂点の座標の計算時間は全体の 1% 程度に過ぎない。Hybrid Arithmetic がここで紹介している混合演算で、Long[2] Arithmetic は条件判定式の計算は全て多倍長整数演算で行なったもの、Double Arithmetic は、条件判定式の計算は全て元の浮動小数点演算で行なってそこで得た符号を必ず用いるものである。これは正常に動作する保証がないが、ここで扱ったような素直なタイプの点数が少ない入力に対して、ほとんどの場合に正常に動作する。図5の横軸が入力母点数、縦軸が実行時間(秒)を表す。混合演算で必要になる条件判定式の形としては式(5)の D' と式(8)の H' を採用し、それに対応する誤差評価(17)と(18)を用いた。これらの結果は、全て10回の平均である。

プログラミング言語として C 言語を用い、「浮動小数点演算」としては C 言語における倍精度浮動小数点型である `double` 型変数を用いた。「多倍長整数演算」については、まずは `double` 型の入力を絶対値が $L = 10^{16}$ 以下の

[1] 式(8)のような行列式の計算は、その行列式を積の和の形に完全に展開してから、左から順に1項ずつ足していく方法を採るものとする。足す順序を変えると誤差の見積もりが違ってくる。

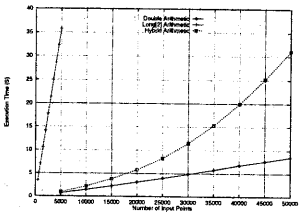

図 5: 浮動小数点演算・多倍長整数演算・混合演算使用時の実行時間比較

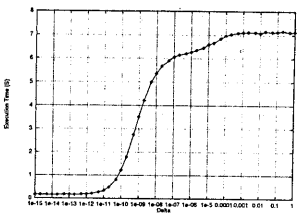

図 6: 浮動小数点型の有効桁と混合演算使用時の実行時間

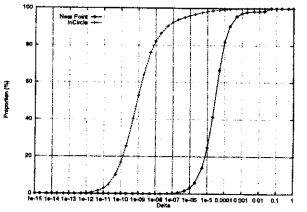

図 7: 浮動小数点型の有効桁と混合演算使用時の多倍長整数演算採用率

範囲に広く分布するようにスケール変換し、整数に丸め、C 言語における倍精度整数型である long 型変数 1 個に絶対値 10^8 までの整数を記憶させて、これを 2 個並べることにより絶対値が $L = 10^{16}$ 以下の入力を表した。10^{16} という数値は、実用上十分な入力精度及び、プログラミングの容易さを意識している。そして、このような整数を入力として、条件判定式の値の計算が厳密に行なえるようにしてある。また、δ の値としては、double 型の有効最小桁 2^{-56} 及び、ここで用いた多倍長整数表現の誤差 10^{-16} よりやや大きめの 10^{-15} を用いた。

Long[2] Arithmetic に対して Hybrid Arithmetic では劇的に計算時間が減っており、母点数 10000 程度であれば Double Arithmetic より少し計算時間が大きい程度に過ぎない。母点数 50000 となっても、近い点の判定においては 1 度も多倍長整数演算が採用されることはなく、外接円の内外判定において 5% 程度採用されるようになった。

6.2 浮動小数点型の有効桁数と混合演算使用時の実行時間との関係

正方形の枠内に一様に発生させた 1000 点の母点に対するボロノイ図を混合演算で求める時の実行時間が、δ の値が大きくなる、つまり浮動小数点型の有効桁数が少なくなるとともにどう変わるかを調べた。図 6の横軸が浮動小数点型の有効最小桁 δ、縦軸が実行時間 (秒) を表す。グラフの最も左の 10^{-15} が、通常の倍精度浮動小数点型の有効最小桁程度であり、それより右の部分は、浮動小数点型の計算精度がそれよりも低い場合を想定している。そして、図 7は 2 種類の条件判定での多倍長整数演算採用率、つまり浮動小数点演算では符号を決定しきれなかった率である。Near Point は近い点の判定、InCircle は外接円の内外判定を表す。やはり、これらの結果は全て 10 回の平均である。

当然ながら、δ の値が小さい時は、図 5の Hybrid Arithmetic のグラフから予想される計算時間となる。逆に、δ の値が大きくなると見積もられる絶対誤差が大きくなり、いづれの条件判定についても多倍長整数演算採用率が上がり、それに伴って実行時間も上がる。そして、最終的には混合演算の効果が全くなくなり、図 5の Long[2] Arithmetic のグラフから予想される計算時間にまで上昇する。

7 誤差のきつい見積もり

以上の方法を以下では Method 1 と呼ぶ[2]。これは一般的に受け入れられているものよりも誤差の見積もり方が甘い。そのため、母点の数がもっと大きい場合には、逐次添加が進むにつれて本当に D や H の値が 0 に近いものが頻繁に登場するようになり、多倍長整数演算に切替えなくて済む確率が下がっていってしまい、計算時間が急激に大きくなると予想できる。また、もともと退化の激しい (条件判定式の値が 0 になりやすいようなタイプの) 入力だった場合も、やはり多倍長整数演算に切替えなくて済む確率が小さいと予想できる。このような場合、誤差の見積もりが厳しくなっていれば、多倍長整数演算に切替えなくて済む確率の減少をある程度は食い止められるはずである。

[2]全部で 3 種類の見積もり方法を示すが、その中で見積もり方の甘いものから順に Method 1、 Method 2、 Method 3 としている。この後、既存の Method 3 を紹介し、その次に Method 3 のある手順を簡略化した Method 2 を示す。

7.1 とてもきつい見積もり

ここでは[11]、[12]の方法を紹介する。以下ではMethod 3と呼ぶ。近い点の判定式の形として式(5)を採用し、外接円の内外判定式の形として式(8)を採用する。まず座標同士の差を

$$\begin{aligned} u_{pq} &\equiv x_p - x_q, \\ v_{pq} &\equiv y_p - y_q, \ (p,\ q = 1,\ 2,\ \ldots,\ n) \end{aligned} \tag{19}$$

と1文字で表して、

$$\begin{aligned} D' &= (x_j - x_i)\{(x_j - x_n) + (x_i - x_n)\} \\ &\quad + (y_j - y_i)\{(y_j - y_n) + (y_i - y_n)\} \\ &= u_{ji}(u_{jn} + u_{in}) + v_{ji}(v_{jn} + v_{in}), \end{aligned} \tag{20}$$

$$\begin{aligned} H' &= -\begin{vmatrix} x_n - x_i & y_n - y_i & (x_n - x_i)^2 + (y_n - y_i)^2 \\ x_n - x_j & y_n - y_j & (x_n - x_j)^2 + (y_n - y_j)^2 \\ x_n - x_k & y_n - y_k & (x_n - x_k)^2 + (y_n - y_k)^2 \end{vmatrix} \\ &= -\begin{vmatrix} u_{ni} & v_{ni} & u_{ni}^2 + v_{ni}^2 \\ u_{nj} & v_{nj} & u_{nj}^2 + v_{nj}^2 \\ u_{nk} & v_{nk} & u_{nk}^2 + v_{nk}^2 \end{vmatrix} \end{aligned} \tag{21}$$

とおく。もし、式(20)のu_{ji}、u_{jn}、u_{in}、v_{ji}、v_{jn}、v_{in}、式(21)のu_{ni}、u_{nj}、u_{nk}、v_{ni}、v_{nj}、v_{nk}を計算する所までが多倍長整数演算で正確に行なわれていて、かつ、D'を計算するたびに

$$N_D \equiv \max\{|u_{ji}u_{jn}|, |u_{ji}u_{in}|, |v_{ji}v_{jn}|, |v_{ji}v_{in}|\} \tag{22}$$

を、H'を計算するたびに

$$N_H \equiv \max_{(p,q,r)\in\Pi_{ijk}} |u_{np}v_{nq}(u_{nr}^2 + v_{nr}^2)| \tag{23}$$

を求めるようにすれば、

$$\begin{aligned} \mathrm{Error}[D'] &= 16N_D\delta, & (24) \\ \mathrm{Error}[H'] &= 68N_H\delta & (25) \end{aligned}$$

と見積もることができる。ただし、Π_{ijk}は(i,j,k)の6種の置換の集合である。式(24)の値は式(17)の値より小さく、式(25)の値は式(18)の値より小さいので、多倍長整数演算に切替えなくて済む確率が高くなるはずである。その長所と、式(20)や(21)の中の座標同士の差の部分を多倍長整数演算で毎回正確に計算して、式(22)や(23)も毎回評価する手間との間に、トレードオフがある。特に、前者の手間が大きい。後者に関しては、式(22)や(23)の中の各項の値は元々計算する必要があったものなので、この評価をするために追加される計算コストは小さい。

7.2 ややきつい見積もり

ここでも先ほどと同様に、近い点の判定式の形として式(5)を採用し、内接円の内外判定式の形として式(8)を採用する。そして、Method 3の時と同様に、D'を計算するたびに

$$\begin{aligned} N_{Dx} &\equiv \max\{u_{ji}, u_{jn}, u_{in}\}, & (26) \\ N_{Dy} &\equiv \max\{v_{ji}, v_{jn}, v_{in}\} & (27) \end{aligned}$$

を、H'を計算するたびに

$$\begin{aligned} N_{Hx} &\equiv \max\{u_{ni}, u_{nj}, u_{nk}\}, & (28) \\ N_{Hy} &\equiv \max\{v_{ni}, v_{nj}, v_{nk}\} & (29) \end{aligned}$$

を求めるようにすれば、Method 3の時のように式の途中までは多倍長整数演算によって正確に求めるということはやめても、

$$\begin{aligned} \mathrm{Error}[D'] &= 8M(N_{Hx} + N_{Hy})\delta \\ &\quad + 10(N_{Hx}^2 + N_{Hy}^2)\delta, & (30) \\ \mathrm{Error}[H'] &= 12M(N_{Hx} + N_{Hy})^3\delta \\ &\quad + 68N_{Hx}N_{Hy}(N_{Hx}^2 + N_{Hy}^2)\delta & (31) \end{aligned}$$

と見積もることができる。この方法を以下ではMethod 2と呼ぶことにする。この誤差はMethod 3で見積もられた誤差と比べて大きく、その分多倍長整数演算に切替えなくて済む確率は小さくなってしまうはずだが、式(5)や(8)の計算のために、座標同士の差までは多倍長整数演算によって正確に求めるということは要求していなくて、式(5)や(8)の毎回の計算の手間はMethod 3と比べて小さい。これは、Method 3と始めに紹介したMethod 1との間の中間的な方法であると言える。まとめると以下のようになる。

	Method 1	Method 2	Method 3
引用	なし	なし	[11] [12]
座標同士の引き算までの厳密計算	しない	しない	する
式の計算経果を見積もりに反映	しない	する N_{Dx}, N_{Dy}を使用 N_{Hx}, N_{Hy}を使用	する N_Dを使用 N_Hを使用
$\mathrm{Error}[D']$	$112M^2\delta$	$8M(N_{Hx}+N_{Hy})\delta$ $+10(N_{Hx}^2+N_{Hy}^2)\delta$	$16N_D\delta$
$\mathrm{Error}[H']$	$2944M^4\delta$	$12M(N_{Hx}+N_{Hy})^3\delta$ $+68N_{Hx}N_{Hy}$ $\times(N_{Hx}^2+N_{Hy}^2)\delta$	$68N_H\delta$

8 数値実験2: 各種混合演算使用時の実行時間比較

8.1 多数個の入力母点

正方形の枠内に一様に発生させた母点に対するボロノイ図を、Method 1、Method 2、Method 3のそれぞれの方法で作った。図8、図9の横軸が入力母点数で、縦軸が実行時間(秒)を表す。図9の方が入力母点数を多くしている。これらの結果は全て10回の平均である。

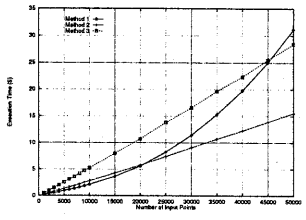

図 8: 各種混合演算の実行時間 (母点数: 小)

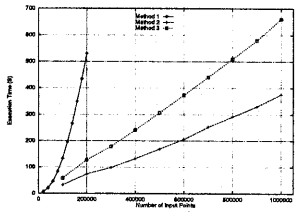

図 9: 各種混合演算の実行時間 (母点数: 大)

Method 1 のような甘い誤差の見積もりを行なうと、入力母点の数が多くなるほど多倍長整数演算採用率が高まり、それが計算時間の増大につながっている。ただし、母点数 200000 になるまでの間に、近い点の判定においては 1 度も多倍長整数演算が採用されることはなく、外接円の内外判定においては、先に示したように母点数 50000 で 5% 程度、さらに母点数 200000 で 40% 程度の割合で多倍長整数演算が採用されるようになった。Method 2、Method 3 については、図 8、9の実験の範囲内では多倍長整数演算が全く採用されなかった。そして、グラフから以下のことがわかる。

- 母点数が 20000 程度までの通常の入力では、Method 1 の見積もりを用いるのが最も早い。
- 母点数がそれ以上になっても、Method 2 の見積もりを用いるのが最も早い。

8.2 密集した入力母点

次に、Method 3 が Method 2 よりも有利である状況を見つけるために、正方形の枠の中心で最大確率をとる 2 次元正規乱数で発生させた 1000 点の母点に対するボロノイ図を、Method 1、Method 2、Method 3 のそれぞれの方法で作った。正規乱数の標準偏差が小さいことが、母点が密集していることを示し、幾何学的条件判定式の値が 0 に近くなりやすく、これらの方法によって解きにくい問題であることを意味する。図 10の横軸が正規乱数の標準偏差で、母点を発生させ得る枠の 1 辺の長さを 1 としている。縦軸は実行時間 (秒) を表す。母点の密集の度合によって採るべき方針が違ってくることがわかる。この結果も全て 10 回の平均である。

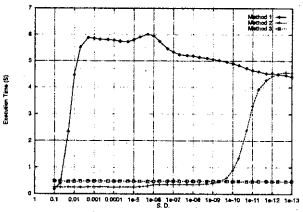

図 10: 密集した入力に対する各種混合演算の実行時間

Method 3 のような誤差のきつい見積もりを行なわない限り、標準偏差が小さくなって入力点が密集してくるほど、多倍長整数演算採用率が高まり、それが計算時間の増大につながっている。しかし、計算時間増大の度合は、Method 1 よりも Method 2 の方が緩やかである。なお、Method 1 で正規乱数の標準偏差が小さい時に徐々に計算時間が下がるのは、多倍長整数演算採用率は下がらないものの、個々の多倍長整数演算において、桁数の小さい入力を扱うようになるからである。そして、グラフから以下のことがわかる。

- 1000 点程度の入力母点数で、密集度もあまり高くない場合では、Method 1 の見積もりを用いるのが最も早い。
- かなり密集度が高い場合でも、一様分布の場合の 10 億倍程度の密集度にならない限り、Method 2 を使えば十分である。

9 結論

以上の実験から以下のことがわかる。

- 混合演算の時に、式の計算結果を誤差見積もりに反映させなくても、かなり広い範囲の入力例で効率が良い。
- 混合演算の時に、式の計算結果を誤差見積もりに反映させることは、入力の規模・性質によっては、効率化のために必要となる。
- 混合演算の時に座標の差までは必ず多倍長整数演算で行なう方針は、よほど特殊な入力例以外では効率が悪い。

参考文献

[1] 杉原 厚吉: 計算幾何工学, アドバンストエレクトロニクスシリーズ II-2, 培風館, 1994.

[2] Herbert Edelsbrunner, Ernst Peter Mücke: Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, *Proceedings of the 4th ACM Annual Symposium on Computational Geometry*, pp. 118-133, Urbana-Champaign, 1988.

[3] Chee-Keng Yap: A Geometric Consistency Theorem for a Symbolic Perturbation Scheme, *Proceedings of the 4th ACM Annual Symposium on Computational Geometry*, pp. 134-142, Urbana-Champaign, 1988.

[4] 杉原 厚吉: 幾何アルゴリズムの数値的破綻とその対策, 応用数理, Vol. 1, No. 4, pp. 280-299, 1991.

[5] K. Sugihara: A Simple Method for Avoiding Numerical Errors and Degeneracy in Voronoi Diagram Construction, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E75-A, No. 4, pp. 468-477, April 1992.

[6] Kokichi Sugihara, Masao Iri: Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic, *Proceedings of the IEEE*, Vol. 80, pp. 1471-1484, 1992.

[7] 杉原 厚吉: 数値誤差と幾何学的整合性 - 暴走の心配のないアルゴリズムをめざして, 電子情報通信学会誌, Vol. 76, No. 6, pp. 618-625, June 1993.

[8] 杉原 厚吉: Voronoi 図 - 世界チャンピオンへの道, *bit*, Vol. 25, No. 1, pp. 24-34, January 1993.

[9] Kokichi Sugihara, Masao Iri: A Robust Topology-Oriented Incremental Algorithm for Voronoi Diagrams, *International Journal of Computational Geometry & Applications*, Vol. 4, No. 2, pp. 179-228, 1994.

[10] Tsuyoshi Minakawa, Kokichi Sugihara: Topology Oriented vs. Exact Arithmetic - Experience in Implementing the Three-Dimensional Convex Hull Algorithm, Algorithms and Computations, *8th International Symposium, ISSAC '97*, pp. 273-282, Singapore, December 1997.

[11] Kokichi Sugihara: Experimental Study on Acceleration of an Exact-Arithmetic Geometric Algorithm, *Proceedings of the 1997 International Conference on Shape Modeling and Applications*, Aizu-Wakamatsu, Japan, March 3-6, 1997.

[12] Kokichi Sugihara: Exact Computation of 4-D Convex Hulls with Perturbation and Acceleration, *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications*, Seoul, Korea, October 5-7, 1999.

[13] Kokichi Sugihara: How to Make Geometric Algorithms Robust, *IEICE Transactions on Information and Systems*, Vol. E83-D, No. 3, pp. 447-454, March 2000.

[14] 杉原 厚吉: 厳密計算と記号摂動と遅延評価を用いた幾何アルゴリズムの実装, 2000 年日本応用数理学会年会, pp. 170-171, 2000 年 10 月.

[15] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara: Spatial Tessellations - Concepts and Applications of Voronoi Diagrams, John Wiley & Sons, 1992.

[16] P. J. Green, R. Sibson: Computing Dirichlet Tessellations in the Plane, *The Computer Journal*, Vol. 21, No. 2, pp. 168-173, 1978.

[17] 大屋 隆生, 室田 一雄: Voronoi Diagram を求める算法とデータ構造の比較・検討, 日本 OR 学会 1982 年度春季研究発表会アブストラクト集, 2C-3, pp. 185-186.

[18] 大屋 隆生, 伊理 正夫, 室田 一雄: Voronoi Diagram を求める算法の改良, 日本 OR 学会 1982 年度秋期研究発表会アブストラクト集, E-8, pp. 152-153.

[19] Takao Ohya, Masao Iri, Kazuo Murota: A Fast Voronoi-Diagram Algorithm with Quaternary Tree Bucketing, *Information Processing Letters*, Vol. 18, pp. 227-231, 1984.

[20] Takao Ohya, Masao Iri, Kazuo Murota: Improvements of the Incremental Method for the Voronoi Diagram with Computational Comparison of Various Algorithms, *Journal of Operations Research Society of Japan*, Vol. 27, No. 4, pp. 69-97, December 1984.