アルゴリズム 39－3
（1994．5．13）

剰余演算による多項式の符号判定と計算幾何学への応用

今井敏行
東京大学工学部

計算幾何学のアルゴリズムには入力が単長整数でも，計算機内で入力変数の多項式の値の符号を判定する必要が生じ，加減乗算の結果，桁数が2倍長や3倍長程度に達するものが少なくない．一般に多倍長整数の符号判定のために多倍長演算を用意すると各演算に時間がかかる．剰余演算を利用して演算時間を短縮する方法が知られているが，剰余から数値を復元するために，結局多倍長演算と変わらないことになる．本稿では，2，3倍長程度の整数に対し，剰余演算の結果から直接に符号判定を行なう方法を提案する．さらに，その方法と普通の多倍長演算による方法をシミュレートした凸包構成プログラムで計算時間を比較した結果を報告する．

Modular Methods to Determine the Sign of the Values of Polynomials and Their Applications to Computational Geometry

Toshiyuki Imai
University of Tokyo, Faculty of Engineering

Not a few geometric algorithms determine signs of values of polynomials whose variables are input data to decide the structure of geometric objects and require double or triple long integer arithmetics such as addition, subrtaction and multiplication even if all the input data are integers of single length.

In the standard method, multiple long integer arithmetic takes much time, especially in multiplication. It has been known that modular arithmetic reduces time of such calculations. Modular arithmetic, however, has to use multiple long integer arithmetics to get back the values themselves from their residues and it takes the same or more time compared to the standard method.

In this paper, we show some techniques to determine the sign of a double or triple long integer directly from the result of modular arithmetic method, and show the performances of the applications to a geometric problem of making a convex hull.

1 はじめに

一般に計算幾何学では, 対象とする図形を表すデータを計量的データと位相的データに分けることができる. 幾何的アルゴリズムでは, 位相的データを決定する判定式は入力データの多項式である. 判定式の値の符号判定を繰り返すことにより図形の位相的データが得られる. 判定式の値が0になる場合, 入力図形が退化状態にあるといわれる.

計算幾何学の研究により開発されてきた高速なアルゴリズムには, 退化状態では, 誤出力, データ破壊, 暴走を招くものが少なくない. また, 厳密には退化状態でなくても, 浮動小数点計算による数値誤差がある場合には誤判定により, 同様なことが起きる.

近年, これらに対処する研究が始まっている. 研究の立脚点としては, 退化状態にあることの判定が可能とする立場と, 実用上は数値誤差は不可避とする立場がある.

退化判定が可能とする立場では, 正確に退化と判定されたときの一般的な対処法の開発に主眼が置かれ, 判定式の値の符号判定は入力を全て整数にして無誤差計算をすればよい, などとされてきた.

整数の無誤差計算の結果を求めるには, 演算1回だけで考えても, 一般に結果の表現に必要なビット数が, 乗算で2倍になり, 加減算で1ビット増える. すなわち, 入力が単長整数としても, 判定式の値は多倍長になるわけである. 計算幾何学の基本的なアルゴリズムの判定式は比較的簡単なものもあり, 2倍長, 3倍長程度の演算が用意されていれば十分なことも少なくないが, 単長ですむ場合はほとんどない. もちろん, 判定式の値が3倍長を越える場合も多くある.

多倍長計算は単長計算に較べ時間がかかる. 単純に n 倍長計算をすると加減算で $O(n)$ 倍, 乗算で $O(n^2)$ 倍の時間がかかる. 乗算は高速乗算法を利用できるが, それでも $O(n \log n \log\log n)$ 倍の時間がかかる [1, 5, 7] うえ, n がある程度大きくないと効果がない. そのために退化判定を多倍長計算で行なうと, 実用上, 計算時間がかかりすぎて退化に対処することができなくなる.

多倍長の多項式の値を剰余演算を利用して高速に計算することは計算幾何学の成立の以前から提案されている [6] が, 問題となるのは, 計算結果の剰余から元の値を復元するときで, 結局, 多倍長計算を用意することになる. これはプログラム作成の負担になるうえ, 多倍長計算と変わらない時間がかかる [1, 5]. 本論文では, 2倍長, 3倍長程度の場合に, 剰余計算の結果から, 元の値を復元せずに値の符号を判定する方法を示す. さらに, 凸包構成アルゴリズムにその方法を適用したものと, 従来の多倍長計算によるものを計算機上に実装し, 計算時間を比較する実験の結果を報告する. また最後にさらに桁数を延ばす方法について検討する.

2 計算幾何学における退化回避と多倍長計算

計算幾何学の研究の中で開発されてきた多くの効率的アルゴリズムは, 正常に動かないような苦手な入力を持つものが少なくない. この苦手な入力とは図形が特殊な位置にある場合が多く, 退化した入力と呼ばれる. 3点が同一直線上に並ぶ, 平面上の4点が同一円周上に並ぶ, 多角形が正多角形になる, など, 退化はアルゴリズムに依存して決まるが, 入力データを乱数で与えると仮定すると退化が起きる確率は低い. そこで計算幾何学の研究の初期においては退化は起こらないこととして放置され, 専らアルゴリズムの効率化のみに研究の重点が置かれた. しかし実用上は, 入力データは乱数ではなく退化した入力は十分に起こることである. また, アルゴリズムを実際に計算機プログラムにして実行すると, 数値誤差が退化時と同様な挙動を引き起こすことがある. このようなことに対する反省から, 近年, アルゴリズムの退化時の統一的対処法が研究されるようになった.

幾何的アルゴリズムは図形の計量的データと位相的データを扱う. 計量的データとは座標や角度などの連続値をとり得るデータで, 位相的データとは辺の接続関係や図形の包含関係などのもともと離散的なデータである.

幾何的アルゴリズムの位相的 / 計量的データは次のように処理される:

- 入力図形の位相的データを見ながら計量的データを計算し, 出力図形の計量的データを得たり, 計算値の正負に応じて出力図形の位相的データを求める,
- 出力図形の位相的データは入力図形の計量的データに加減乗算と正負の符号判定を繰り返すことで得られる.

すなわち, 幾何的アルゴリズムには, 入力データを変数とする, ある多変数多項式の集合があり, それらの判定多項式に入力の計量データを代入したときの値の正負で, 出力の位相的データが決定される. また, 位相的データを得るとき符号判定の結果が 0 になる時が退化であり, 退化や計算誤差による誤判定のために図形の位相的データが破壊されていくわけである.

退化に統一的に対処する方法の研究には 2 つの方向がある:

1. 退化状態にあることの正確な判定が可能とし, 退化とわかってから対処する方向,
2. 実用上は数値誤差は避けられず, 退化状態の正確な判定は不可能とし, 誤差にみあう出力を求める方向.

前者の場合, 無誤差判定が必須であるが, 多倍長で整数計算する, など [2] で片付けられることが多い.

3 剰余演算による多倍長計算

アルゴリズム中の判定多項式 $f(x)$ は実際には, 入力変数 $x = (x_1, \cdots, x_k)$ の加減乗算の手続きの組み合わせとして与えられる. ここでは, 加減乗算を基本演算とよぶ.

定理 1 整数 n, n', m, m' と正整数 q に対し, $m \equiv m'$, $n \equiv n' \bmod q$ ならば

$$m \pm n \equiv m' \pm n',\ mn \equiv m'n' \bmod q \tag{1}$$

が成立する.

この定理から, 基本演算ごとに対応する剰余演算におきかえるだけで, アルゴリズムを剰余演算を利用する方法にできることがわかる. これは, 剰余演算を利用するためには, 汎用の基本剰余演算のアルゴリズムを用意すれば, 事実上, 既存のアルゴリズムの本体を変更する必要がないことを示している.

加減乗算に対応する剰余演算は次のようにおこなう [4, 6]. ここで利用する計算機は, 符号なし単長 (n ビット) 整数の基本演算で, 次のような機能が利用できるものとする:

- 加減算に関して, $n+1$ ビット目への操り上がりがあるかどうかの情報,
- 乗算に関して, 結果の下位 n ビットと上位 n ビット.

これは CISC 型の CPU ならば一般に備わっている機能である. この意味で本論文で示す方法はハードウェアに依存する. 以下では, 一般に $N \in \mathbf{Z}$, $q \in \mathbf{N}$ に対し, N を q で整除した剰余を $\mathrm{MOD}(N, q)$ とする. また, $p_0 = 2, p_1 = 2^n - 1, p_2 = 2^{n-1} - 1, p_3 = 2^n - 3$ とし, これらの積を $p = p_0p_1p_2p_3$, $\tilde{p} = p_1p_2p_3$ とする. ここで, X, Y を整数とし, $X_i = \mathrm{MOD}(X, p_i), Y_i = \mathrm{MOD}(Y, p_i)$ $(i = 0, \cdots, 3)$ とする. X_0, Y_0 は 1 ビットの数で, 和と差 $\mathrm{MOD}(X \pm Y, p_0)$ はどちらも X_0 と Y_0 の排他的論理和 (exclusive or) であり, 積 $\mathrm{MOD}(XY, p_0)$ は X_0 と Y_0 の論理積 (and) である. $i = 1, \cdots, 3$ のときは, 基本剰余演算を次のようにする. $p_i = 2^m - \alpha$ $(m = n, n-1)$ と

する, 基本演算の結果 Z が m ビットに納まらないときには, 下位 m ビットを Z_{L} とし, 上位ビットを Z_{U} とすると $Z = 2^m Z_{\mathrm{U}} + Z_{\mathrm{L}}$ となり, $2^m \equiv \alpha \bmod p_i$ より $Z \equiv Z_{\mathrm{L}} + \alpha Z_{\mathrm{U}} \bmod p_i$ であるから, 手続き

$$Z_{\mathrm{L}} \leftarrow Z\text{の下位}\, m\, \text{ビット}; Z_{\mathrm{U}} \leftarrow Z\text{の上位ビット}; Z \leftarrow Z_{\mathrm{L}} + \alpha Z_{\mathrm{U}};$$

を反復して Z が m ビットに納める. もし $Z < p_i$ ならば Z が, さもなければ $Z - p_i$ が基本剰余演算の結果 $\mathrm{MOD}(Z, p_i)$ である. 上記の手続きの反復回数は, 一般に $p > \alpha^2$ ならば 3 回以下であることが知られている [4]. また実際には α を掛ける操作は乗算を行なうのではなく α の 2 進数表記を利用してビットシフトを組み合わせる. すなわち, $i = 1, 2$ のときには $\alpha = 1$ であるから, 実際には α をかける必要はない. また, $i = 3$ のときには $\alpha = 3$ であるが, これは 2 進数で $11_{(2)}$ であるから, Z_{U} を 1 ビット左にシフトしたものを $(Z_{\mathrm{U}} << 1)$ とかくと $Z_{\mathrm{L}} + \alpha Z_{\mathrm{U}}$ のかわりに $Z_{\mathrm{L}} + (Z_{\mathrm{U}} << 1) + Z_{\mathrm{U}}$ とすることができ, この部分で乗算を回避できる.

また実際には, 積 $p_0 p_2$ $(= 2^n - 2)$ が n ビットに納まるから, まず, $\mathrm{MOD}(Z, p_0 p_2)$ を計算し, それから $\mathrm{MOD}(Z, p_0)$, $\mathrm{MOD}(Z, p_2)$ を求めることで計算時間を短縮することもできる. 具体的には,

$$\mathrm{MOD}(Z, p_0) = \mathrm{MOD}(Z, p_0 p_2)\text{の最下位のビット}, \tag{2}$$

$$\mathrm{MOD}(Z, p_2) = \begin{cases} \mathrm{MOD}(Z, p_0 p_2), & (\mathrm{MOD}(Z, p_0 p_2) < p_2), \\ \mathrm{MOD}(Z, p_0 p_2) - p_2, & (\text{その他}), \end{cases} \tag{3}$$

である. このようにして計算時間を短縮する方式を複合方式とよび, $\mathrm{MOD}(Z, p_0)$, $\mathrm{MOD}(Z, p_2)$ を直接求める方式を個別方式とよぶことにする.

4 多項式の符号判定

$x = (x_1, \cdots, x_n)$ を入力変数ベクトルとする幾何的アルゴリズムで入力データが $a \in \mathbf{Z}^n$ であるとする. $f(x)$ を, このアルゴリズム中で符号を判定する多項式とする, すなわち, 入力が a のとき $f(a) \in \mathbf{Z}$ の正負によりアルゴリズム中の処理が分岐する. ここで, $f(x)$ は整数係数多項式で $-\tilde{p} \le f(a) < \tilde{p}$ を仮定する. 幾何的アルゴリズムではこの条件を満たすものは特殊ではない. この範囲 $[-\tilde{p}, \tilde{p})$ にある整数の個数は p $(= (2^n - 1)(2^n - 2)(2^n - 3))$ であり, 2^{3n} に近い. すなわち $f(a)$ のとり得る値は 3 倍長程度である.

剰余演算を利用する方法では, $f(a)$ を多倍長整数として求めず, 各 p_i $(i = 0, \cdots 3)$ による剰余 $N_i = \mathrm{MOD}(f(a), p_i)$ を求める.

定理 2 $f(a)$ が非負であるための必要十分条件は $\mathrm{MOD}(f(a), \tilde{p})$ と N_0 のパリティが一致することである.

証明 N_0 と $f(a)$ のパリティは一致する. $0 \le f(a) < \tilde{p}$ のとき $f(a) = \mathrm{MOD}(f(a), \tilde{p})$, 従って $\mathrm{MOD}(f(a), \tilde{p})$ と $f(a)$ のパリティは一致する. また, $-\tilde{p} \le f(a) < 0$ のとき $f(a) = \mathrm{MOD}(f(a), \tilde{p}) - \tilde{p}$ で, $\tilde{p}$ は奇数であるから $f(a)$ と $\mathrm{MOD}(f(a), \tilde{p})$, のパリティは一致しない. 従って $f(a)$ が非負のときのみ $\mathrm{MOD}(f(a), \tilde{p})$ と N_0 のパリティが一致する.

また, $f(a) = 0$ であるための必要十分条件は $N_0 = N_1 = N_2 = N_3 = 0$ であるから, 上の定理と合わせて $f(a)$ の符号判定ができる.

つぎに, $\mathrm{MOD}(f(a), \tilde{p})$ のパリティを N_1, N_2, N_3 から求めることを考える.

中国剰余定理により, 整数 u_1, u_2, u_3 を, $u_1 p_2 p_3 + u_2 p_1 p_3 + u_3 p_1 p_2 = 1$ になるようにとると

$$\mathrm{MOD}(f(a), \tilde{p}) = \mathrm{MOD}(N_1 u_1 p_2 p_3 - N_2 u_2 p_1 p_3 + N_3 u_3 p_1 p_2, \tilde{p}) \tag{4}$$

となる．ここでは $u_1 = u_3 = 1, u_2 = -1$ にとる．$N_1p_2p_3 - N_2p_1p_3 + N_3p_1p_2$ を $\tilde{p}$ で整除した商を Q とすると，剰余は $\mathrm{MOD}(f(a), \tilde{p})$ であるから

$$\mathrm{MOD}(f(a), \tilde{p}) = N_1p_2p_3 - N_2p_1p_3 + N_3p_1p_2 - Q\tilde{p} \tag{5}$$

となる．$p_1, p_2, p_3, \tilde{p}$ は奇数であるから $\mathrm{MOD}(f(a), \tilde{p})$ のパリティは，Q が偶数のとき $N_1+N_2+N_3$ のパリティに一致し，Q が奇数のときには一致しないことがわかる．

また $0 \leq N_i < p_i\ (i = 1, \cdots, 3)$ より，

$$-\tilde{p} < N_1p_2p_3 - N_2p_1p_3 + N_3p_1p_2 < 2\tilde{p} \tag{6}$$

である．これは $-1 \leq Q \leq 1$ を意味する．

したがって $Q = 0$ のときにのみ Q は偶数となる．これより，次の定理を得る．

定理 3 $\mathrm{MOD}(f(a), \tilde{p})$ のパリティは，$Q = 0$ のとき，$N_1 + N_2 + N_3$ のパリティに一致し，それ以外のとき一致しない．

$Q = 0$ の必要十分条件 $0 \leq N_1p_2p_3 - N_2p_1p_3 + N_3p_1p_2 < \tilde{p}$ であるかどうか判定するのは，そのままでは多倍長整数の大小比較になるので，$2p_2 = p_1 - 1 = p_3 + 1$ を用いて同値変形して

$$0 \leq N_1 - 2N_2 + N_3 - \frac{N_1p_3 - N_3p_1}{p_1p_3} < 2p_2 \tag{7}$$

を得る．ここで $0 \leq N_1 < p_1, 0 \leq N_3 < p_3$ より，$-1 < \dfrac{N_1p_3 - N_3p_1}{p_1p_3} < 1$ であり，整数性を考慮すると，

$$\epsilon_1 = \begin{cases} 1, & (N_1p_3 - N_3p_1 > 0), \\ 0, & (\text{その他}), \end{cases} \tag{8}$$

と定義して，$Q = 0$ であるための必要十分条件

$$0 \leq N_1 - 2N_2 + N_3 - \epsilon_1 < 2p_2 \tag{9}$$

を得る．この不等式は比較方法を工夫することによって成立するかどうかの判定が単長で可能である．さらに，$p_1 = p_3 + 2$ であることを利用して，

$$\epsilon_2 = \begin{cases} 1, & (N_3 \geq 2^{n-1} - 1), \\ 0, & (\text{その他}), \end{cases} \tag{10}$$

とすることにより，$N_1p_3 - N_3p_1 > 0$ と同値な式 $N_1 > N_3 + \epsilon_2$ を得る．これですべての計算を単長内に納めることができた．まとめると次の定理が得られる．

定理 4 $Q = 0$ であるための必要十分条件は

$$\epsilon_2 = \begin{cases} 1, & (N_3 \geq 2^{n-1} - 1), \\ 0, & (\text{その他}), \end{cases} \quad \epsilon_1 = \begin{cases} 1, & (N_1 > N_3 + \epsilon_2), \\ 0, & (\text{その他}), \end{cases} \tag{11}$$

として，

$$0 \leq N_1 - 2N_2 + N_3 - \epsilon_1 < 2p_2 \tag{12}$$

である．

この条件が成り立つかどうかを判定するために細かい場合分けをして単長計算するかわりに，再び剰余計算による方法をとることができる．この場合，$-(2^n-4) \leq N_1-2N_2+N_3-\epsilon_1 \leq 2^{n+1}-6$ であるから，3倍長をとる必要はない．p_0,p_1,p_2 による剰余計算で2倍長程度の整数の符号判定ができれば十分である．以下に，その方法について説明する．既に示した p_0,p_1,p_2,p_3 による剰余計算を用いる方法を3倍長版，これから示す p_0,p_1,p_2 による方法を2倍長版とよぶことにする．2倍長版の場合には関係式 $p_1-2p_2=1$ に中国剰余定理を用いて同様の議論ができる．$M=N_1-2N_2+N_3-\epsilon_1$ として，M を p_1p_2 で整除した商を Q' とすると，

$$\mathrm{MOD}(M,p_1p_2)=N_1-2N_2+N_3-Q'p_1p_2 \tag{13}$$

であり，Q' が奇数になるのは

$$0<\mathrm{MOD}(M,p_1)-\mathrm{MOD}(M,p_2)\leq p_2 \tag{14}$$

のときに限ることが容易にわかるので，$\mathrm{MOD}(M,p_1p_2)$ のパリティは Q' のパリティと N_1,N_2,N_3 から容易にわかる．$M_0=\mathrm{MOD}(M,p_0),\mathrm{MOD}(M,p_1p_2)$ のパリティが一致するときのみ M は非負である．これで M の符号が判定できる．とくに3倍長演算を必要としない場合には，こちらを用いたほうが高速な処理ができる．

いずれにしても，剰余演算を用いた方法では，多倍長整数を直接扱う方法に較べて判定多項式の値 $f(a)$ の正負の符号判定に手間がかかる．しかし，その符号判定をする前に，$f(a)$ の計算に，一般には複数回の乗算が必要であり，$f(a)$ を求めるために行なう乗算の回数がある程度あれば正負の符号判定での計算時間の増加以上に乗算での計算時間の短縮が見込める．

5 凸包構成への適用例

剰余計算を利用した方法によって計算時間がどう変わるかは適用するアルゴリズムに依存する．一般には判定多項式の値 $f(a)$ を求めるのに必要な乗算回数が多いほど剰余演算を利用した方法が計算時間で有利である．

具体的な問題で計算時間を比較するため，凸包構成アルゴリズムに多倍長演算と剰余演算を用いた方法を適用した．凸包構成アルゴリズムは文献[3]による．これは逐次添加型のアルゴリズムで，第1座標順にソートされた点集合を入力とし，K 点の点集合に対する凸包を求めるのに $O(K)$ の計算時間ですむ．

この数値計算部分を2倍長演算，3倍長演算，剰余演算のそれぞれで実装したプログラムを用意した．使用した言語はC，使用した計算機は富士通S-4/ECである．この計算機のCPUはRISC型のSPARCチップであるため，本来の単長整数のビット長32ビットを2倍長とみなし，多倍長演算と剰余演算はCの関数として自作した．入力データを，多倍長演算用に16ビットに区切ることと，剰余演算用に各 p_i での剰余を求めることは，別のプログラムとして用意し，計算時間からは除外した．また，退化時の特別な対処は行なっていない．

このアルゴリズムで入力の点集合を点列と見て $v=(v_1,\cdots,v_K)$ とし，点 v_i の座標を (x_{i1},x_{i2}) とすると，判定多項式は

$$f_{ijk}(x)=\begin{vmatrix} 1 & 1 & 1 \\ x_{i1} & x_{j1} & x_{k1} \\ x_{i2} & x_{j2} & x_{k2} \end{vmatrix} \tag{15}$$

である．各座標値が $[-L,L]$ の範囲の整数値を取るとき $-4L^2 \leq f_{ijk}(x) \leq 4L^2$ である．実験1のため，$4L^2<p_0p_1p_2$ となるように $L=23169$ と選び，入力点の総数 K は46339 $(=2L+1)$ とし，第1座標は v_1 から順に -23169 $(=-L)$ から始まる連続する整数で，v_K の第1座標は23169 $(=L)$ にとった．第2座標は $[-L,L]$ の範囲の整数値を一様乱数によりとった．実験1

ではこのようにして得た5種類の点集合から, はじめの10000点, 20000点, 30000点, 40000点および全点をとり, 各プログラムにより凸包を求め, 計算時間を測定した. 実際には計算時間がほとんど一致したため, 点の個数ごとに計算時間の平均をとり表2に示した. また, 実験2として, 4点 $v_1(-29491, 16384), v_2(-22937, -6553), v_3(16384, 29491), v_4(29491, -16384)$ の凸包を求めた。これは計算結果が2倍長を超える例である. 剰余計算は p_0, p_2 による剰余を求めるときに個別に求める個別方式と p_0p_2 による剰余から求める複合方式を採用した. 実験対象としたプログラムは次の7種類である.

表1 プログラムの名前と内容

c2s	2倍長演算
c3s	3倍長演算
c2m	剰余演算, 2倍長版, 個別方式
c2n	剰余演算, 2倍長版, 複合方式
c3m	剰余演算, 3倍長版, 個別方式, 場合分けして単長で符号判定
c3n	剰余演算, 3倍長版, 複合方式, 場合分けして単長で符号判定
c3np	符号判定に2倍長版を利用したc3nの符号判定簡略化

表2 実験1の結果(単位は秒)

点数	c2s	c2m	c2n	c3s	c3m	c3n	c3np
10000	4.4	8.5	6.6	9.7	10.3	8.9	9.6
20000	8.9	17.0	13.2	19.5	20.6	17.9	19.4
30000	13.3	25.7	19.9	29.1	31.0	27.0	29.1
40000	17.6	34.2	26.5	38.7	41.4	35.9	38.7
46339	20.2	39.7	30.8	44.8	47.9	41.7	45.0

表3 実験2の結果(凸包上を時計周りにみた点の添字の順序)

プログラム	c2s	c2m	c2n	c3s	c3m	c3n	c3np
凸包	1432	1432	1432	1342	1342	1342	1342

実験1の結果この凸包構成アルゴリズムに関しては2倍長では多倍長演算を用意したほうが剰余演算を利用するより速いが, 3倍長で剰余演算を利用する方法が追い付き, 剰余演算の仕方によっては逆転することがわかる. また, さらに桁数が延びた場合には, 剰余演算が計算時間で有利になることもわかる. また, 実験2では, 実際に入力が単長でも2倍長では判定を誤る例が存在し, その例に対し3倍長では正確な判定ができたことがわかる.

6 まとめ

整数の加減乗算で得られる多項式の値 $f(a)$ が2倍長, 3倍長程度の整数になるとき, 剰余演算で計算時間を短縮し, 剰余から元の整数 $f(a)$ を復元することなく符号判定する方法を提案した.

これは，幾何的アルゴリズム中の判定多項式の値の符号判定に利用できる．凸包構成アルゴリズムに適用し，従来の多倍長計算と較べ3倍長版では高速になることを確認した．さらに桁数が延びれば，従来の多倍長計算と較べ，より計算時間で有利になることも確認できた．

問題点としては，現状では，2倍長，3倍長程度しかサポートできないことが挙げられる．また，剰余演算を利用するために実際には2倍長，3倍長にわずかに届かないことも挙げられる．

今後の課題としては，CISC型のCPUの持つ乗算命令を利用して剰余演算部分をアセンブラ化し，より実用的な状況で速度比較を行なうこと，さらに桁数を延ばすため，剰余をとり，符号判定するのによい数の組を発見することなどが挙げられる．

本論文で提案した方法は2倍長，3倍長だけでなく，さらに桁数を延ばす場合でも利用できる部分がある．奇数 $p_1, \cdots, p_n$ に対して，改めて $\tilde{p} = p_1 p_2 \cdots p_n$ とし $q_i = \tilde{p}/p_i$ とする．整数 $u_1, \cdots, u_n$ を $u_1 q_1 + u_2 q_2 + \cdots + u_n q_n = 1$ を満たすようにとり（このように u_i がとれるように p_i もとらねばならない）．$0 \leq N_i < p_i$ をみたす整数 N_i に対して，$N_1 u_1 q_1 + N_2 u_2 q_2 + \cdots + N_n u_n q_n$ を $\tilde{p}$ で割ったときの商 Q が偶数になる必要十分条件を求める（このとき u_i の絶対値が小さいほうが Q の取り得る値の種類が少なくてすむ．そうなるような p_i を選ぶのが重要であると思われる）．商 Q が偶数になる条件を整数性を利用してなるべく簡単にする．（このとき，条件の不等式が単長での判定が難しいときには，この判定にはより短い桁数用の剰余演算を利用した方法を使うことができる）．$N_i = \mathrm{MOD}(N, p_i)$ を満たす整数 N $(-\tilde{p} \leq N < \tilde{p})$ の符号は，この条件判定と各 N_i のパリティ，および $N_0 = \mathrm{MOD}(N, 2)$ から判定できる．

桁数が延びた場合，より広範囲の幾何的アルゴリズムで，入力退化かどうかの高速で正確な判定の一方法になり，記号摂動法などの退化の一般的回避法の組み込みも現実的になり得る．また，本論文で説明した方法は，一般的な枠組みで多項式の正負の符号判定ができるので計算幾何学以外にも広い適用範囲を持つ．

謝辞 文献[6]を紹介してくださり，本研究を始めるきっかけを与えてくださった，日本大学生産工学部の高橋磐郎教授に感謝します．なお，本研究の一部は科学研究費補助金（奨励研究（A））の援助を受けている．

参考文献

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

[2] H. Edelsblunner and E. P. Mücke: Simulation on Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, Proc. 4th ACM Annual Symposium on Computational Geometry, pp. 134–142, 1988.

[3] D. Avis, 今井浩, 松永信介: 計算幾何学, 離散幾何学, 朝倉書店, 1994.

[4] 伊理正夫, 野崎昭弘, 野下浩平 (編著): 計算の効率化とその限界, 数学セミナー増刊, 日本評論社, 1980.

[5] D. E. Knuth: The Art of Computer Programming, vol. 2, Seminumerical Algorithms, 2nd ed., Addison-Wesley, 1981.

[6] 高橋秀俊, 石橋善弘: 電子計算機による exact な計算の新方法 (modulo p 演算の応用), 情報処理, vol. 1, pp. 28–42, 1960.

[7] 和田秀男: 高速乗算法と素数判定法, 上智大学数学講究録 No. 15, 上智大学数学教室, 1983.