ELSEVIER

Mathematics and Computers in Simulation 42 (1996) 509–528

MATHEMATICS AND COMPUTERS IN SIMULATION

Floating point Gröbner bases

Kiyoshi Shirayanagi [1]

NTT Communication Science Laboratories, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

Abstract

Bracket coefficients for polynomials are introduced. These are like specific precision floating point numbers together with error terms. Working in terms of bracket coefficients, an algorithm that computes a Gröbner basis with floating point coefficients is presented, and a new criterion for determining whether a bracket coefficient is zero is proposed. Given a finite set F of polynomials with real coefficients, let G_μ be the result of the algorithm for F and a precision value μ, and G be a true Gröbner basis of F. Then, as μ approaches infinity, G_μ converges to G coefficientwise. Moreover, there is a precision M such that if $\mu \geqslant M$, then the sets of monomials with non-zero coefficients of G_μ and G are exactly the same. The practical usefulness of the algorithm is suggested by experimental results.

1. Introduction

This paper is a refined version of [12]. Gröbner basis (GB) techniques are a valuable tool for solving many problems in polynomial ideal theory. As is well known, the process of computing a GB may involve large numbers of intermediate coefficients – say from a field k – even when the final GB does not involve many coefficients. In fact, the cost of performing exact arithmetic in k with the intermediate coefficients is a major factor determining the computational cost of computing the GB. This paper proposes a new approach using floating point computation that can be applied when k is a subfield of the real numbers. [2]

Basically we mimic Buchberger's algorithm in [3]. However, the big question then would be "how small must floating point coefficients be, to be considered zero?". The subject of this paper is to propose a criterion for answering this question. Our key idea is to calculate and keep track of an error term for every coefficient that occurs at each step of the S-polynomial calculation or polynomial reduction, and to judge coefficients as zero by estimation of their accumulated errors. To keep track of the errors, bracket coefficients for polynomials are introduced. These are like floating point numbers together with error terms. In place of bracket coefficients, intervals in *interval arithmetic* (see [1,9,10]) can also work well.

[1] E-mail: shirayan@progn.kecl.ntt.jp.

[2] While our approach applies to the case where k is the field of rational numbers, several other approaches apply there as well. We refer the reader to p-adic or modular approaches as found in [11,13,14].

SSDI 0378-4754(96)00027-4

The proposed algorithm gives a sort of *approximate* GB (or AGB), not a *true* GB. There are a number of problems that are typically solved by computing a GB for which it is only necessary to compute an AGB. Moreover, for numerous examples, it is much faster to compute an AGB than a GB.

In Section 2, a new notion for the convergence of a sequence of floating point polynomials and an approximate GB are defined to describe what our algorithm does. We present an example illustrating why the most naive approach does not work. In addition, possible applications are mentioned. In Section 3, basic notions for our algorithm are provided as well as a key theorem. Bracket coefficient polynomials are introduced. They play a central role in this paper. The key theorem presents a criterion for determining whether a true coefficient is zero, in terms of the bracket coefficients. In Section 4, the algorithm is described and the termination and correctness of it are proved. In Section 5, examples of running the algorithm are presented. Observations are also included. Section 6 summarizes this paper. Finally, important open problems are given.

Throughout this paper, we assume that the set of all floating point numbers is contained in the real field $\mathbb{R}$. When we simply say a polynomial, it denotes a floating point polynomial or a real polynomial in variables $x_1, \ldots, x_n$.

2. Floating point Gröbner sequence

In this section, we introduce a sequence of sets of floating point polynomials that converges to a true Gröbner basis in a strong way. First of all, we define the support of a polynomial or a finite set of polynomials.

Definition 1 (*Support*). The support of a polynomial

$$f = \sum_{i_1,\ldots,i_n} a_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$$

is the set of power products

$$\{x_1^{i_1} \cdots x_n^{i_n} \mid a_{i_1,\ldots,i_n} \neq 0\},$$

denoted $Supp(f)$. The support of a finite set $F = \{f_1, \ldots, f_n\}$ of polynomials is the set of subsets of the set of power products: $\{Supp(f_1), \ldots, Supp(f_n)\}$, denoted $Supp(F)$.

Note that $Supp(0) = \emptyset$.

Examples

(1) For $f = 2.99x^2 + 0.999xy^2 - 1.002z + 0.249$, $Supp(f) = \{x^2, xy^2, z, 1\}$.

(2) For $F = \{3x^2 + xy^2 - z + \frac{1}{4}, \frac{1}{3}x + y^2z + \frac{1}{2}, x^2z - \frac{1}{2}x - y^2\}$, $Supp(F) = \{\{x^2, xy^2, z, 1\}, \{x, y^2z, 1\}, \{x^2z, x, y^2\}\}$.

Next we discuss the general notion of (coefficientwise) convergence and the specialized notion of support-wise convergence which we require.

The natural definition of the coefficientwise convergence of a sequence of polynomials is given by the following.

Definition 2 (*Coefficientwise convergence*). Let $\{f_\nu\}_\nu$ be a sequence of polynomials and f be a polynomial. Then, f_ν coefficientwise converges to f ($f_\nu \to f$ coefficientwise) iff $f_\nu = \sum_{i_1,\ldots,i_n} a^\nu_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$ and $f = \sum_{i_1,\ldots,i_n} a_{i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}$ such that $\lim_{\nu\to\infty} a^\nu_{i_1,\ldots,i_n} = a_{i_1,\ldots,i_n}$ for all $i_1, \ldots, i_n$.

The stronger property of supportwise convergence is one of the central themes of this paper.

Definition 3 (*Supportwise convergence*). Let $\{f_\nu\}_\nu$ be a sequence of polynomials and f be a polynomial. Then, f_ν supportwise converges to f ($f_\nu \to f$ supportwise) iff
(1) $\{f_\nu\}_\nu$ is coefficientwise convergent to f, and
(2) there is an N such that $Supp(f_\nu) = Supp(f)$ for all $\nu \geqslant N$.

Moreover let $\{F_\nu\}_\nu$ be a sequence of finite sets of polynomials and F be $\{f_1, \ldots, f_n\}$, a finite set of polynomials. Then, F_ν supportwise converges to F ($F_\nu \to F$ supportwise) iff there is an N such that for all $\nu \geqslant N$, $F_\nu = \{f_1^\nu, \ldots, f_n^\nu\}$ where $f_i^\nu \to f_i$ supportwise for all $i \in [1, n]$.

In other words, supportwise convergence emphasizes that coefficients that are going to converge to zero, reach zero in a *finite* number of steps.

Examples
(1) $((\nu + 1)/\nu)x^3y \to x^3y$ supportwise.
(2) $(1/\nu)x^3y \to 0$ coefficientwise, but $(1/\nu)x^3y \not\to 0$ supportwise, since $Supp((1/\nu)x^3y) \neq Supp(0)$ for any ν.

The aim of this paper is to provide an algorithm that computes a set G_μ for each μ where $\{G_\mu\}_\mu$ is supportwise convergent to a true Gröbner basis G. Let us call such a sequence $\{G_\mu\}_\mu$ a *floating point Gröbner sequence*. If $\{G_\mu\}_\mu$ is any floating point Gröbner sequence, by definition there is an M where $Supp(G_\mu) = Supp(G)$ for all $\mu \geqslant M$. For $\mu \geqslant M$ we shall call G_μ a *floating point Gröbner basis* with precision μ, or simply an *approximate Gröbner basis*.

Of course there is an obvious way to construct a floating point Gröbner sequence. That is,
(1) compute a true Gröbner basis G by the conventional Buchberger's algorithm [3], and then
(2) for each μ, truncate the coefficients of G to precision μ.
However, our algorithm avoids exact arithmetic and the associated memory requirements. On the other hand, we must be able to compute with arbitrarily high precision coefficients.

For this purpose one naive idea may be first to take F_μ, that is, the floating point approximation of F in coefficients to precision μ, and then to apply Buchberger's algorithm to F_μ. However, such an approach fails before the GB computation begins. This can be seen by a trivial example $\{3x - 1, x - \frac{1}{3}\}$. With any truncation of $\frac{1}{3}$, the first polynomial is no longer a scalar multiple of the second one, and the ideal generated by them has changed to $\mathbb{R}[x]$. See Section 5 for the result of our algorithm for a similar and less trivial example.

Let V be the quotient algebra

$$\mathbb{R}[x_1, \ldots, x_n]/Ideal(F),$$

where $Ideal(F)$ is the ideal generated by F. Once an approximate Gröbner basis G_μ is obtained, the following problems can be solved *not approximately but strictly* by the conventional methods, since $Supp(G_\mu) = Supp(G)$ for a true Gröbner basis G; in particular, the sets of all the leading power products of both coincide.

(1) Decide whether V is finite-dimensional or not as the $\mathbb{R}$ vector space,
(2) Compute an $\mathbb{R}$-basis of V,
(3) Decide whether the system F of polynomials is solvable or not, etc.

Moreover, the system F may be *approximately* solved, since every coefficient of G_μ converges to the corresponding one of G. See [3, Methods 6.6, 6.8–6.10, etc.], for the details on the conventional methods.

3. Theoretical foundation

3.1. Basic notions

It would be natural to mimic Buchberger's algorithm but with floating point coefficients. However, the biggest problem then is "*how small must coefficients be for us to consider them to be zero?*". It should be noted that *if our algorithm can judge a coefficient as zero iff the corresponding coefficient in Buchberger's algorithm is truly zero, then the results are exactly the same in supports*. Thus we must provide a useful criterion for this zero judgment. Our idea is to calculate and keep track of an error of every coefficient that occurs at each step of the S-polynomial calculation or polynomial reduction.

To keep track of the errors, a *bracket coefficient polynomial*, or simply a *BC polynomial* with a given precision μ is introduced. This is 0 or a polynomial of the form

$$\sum_{i_1,\ldots,i_n} [A_{i_1,\ldots,i_n}, \alpha_{i_1,\ldots,i_n}] x_1^{i_1} \cdots x_n^{i_n},$$

where $A_{i_1,\ldots,i_n}$ and $\alpha_{i_1,\ldots,i_n}$ are floating point numbers with precision μ. $A_{i_1,\ldots,i_n}$ and $\alpha_{i_1,\ldots,i_n}$ are intended to be an approximation of a true coefficient and its error, respectively. A special symbol **0** denotes the BC $[0, 0]$.

Next we have to define the S-BC polynomial and BC reduction for BC polynomials. In this paper, using 10 as the base, a floating point number A with precision μ is expressed by

$$A = \pm.a_1 a_2 \ldots a_\mu \times 10^{e(A)},$$

where $1 \leqslant a_1 \leqslant 9$, $0 \leqslant a_i \leqslant 9$ $(2 \leqslant i \leqslant \mu)$, and $e(A)$ is the exponent of A. For floating point arithmetic, we apply *roundoff* manipulation, which means counting 0.5 and higher fractions as a unit and cutting away the rest. In this paper we assume that transgressions of the range of numbers (underflow and overflow) do not happen. Following [8], we use notations $\oplus_\mu$, $\ominus_\mu$ and $\otimes_\mu$ (or simply $\oplus$, $\ominus$ and $\otimes$ when there is no confusion) for floating point addition, subtraction and multiplication with precision μ, respectively. Throughout this paper, floating point division is not considered because of its complicated error analysis.

Furthermore we apply *round up* with precision μ, denoted $\uparrow_\mu$ or simply $\uparrow$, see [1], for addition or multiplication of errors. That is, $\uparrow (\alpha + \beta)$ (respectively $\uparrow (\alpha\beta)$) is the same as $\alpha \oplus \beta$ (respectively $\alpha \otimes \beta$) except that it rounds the $(\mu + 1)$th fraction upward-directly to 10. We assume that the result of $\uparrow$ is always not smaller than the exact result in absolute values. Note that when loss of information happens,

$\alpha + \beta > \uparrow_\mu (\alpha + \beta)$ is possible. Thus, strictly speaking, we should devise a modified version of round up so as to cope with such a case.

Note that floating point (and round up) addition, subtraction and multiplication with fixed precision have difficulties in that the associative law or distributive law does not necessarily hold in general. Fortunately these deficiencies will not cause a problem for our algorithm.

We define floating point arithmetic for bracket coefficients in BC polynomials. $\uparrow (\alpha + \beta + \gamma + \cdots + \xi)$ denotes $\uparrow (\uparrow (\cdots \uparrow (\uparrow (\alpha + \beta) + \gamma) + \cdots) + \xi)$.

Definition 4 (*BC arithmetic*). With precision μ,
addition: $[A, \alpha] \oplus [B, \beta] = [A \oplus B, \uparrow (\alpha + \beta + 5 \times 10^{e(A \oplus B)-(\mu+1)})]$;
subtraction: $[A, \alpha] \ominus [B, \beta] = [A \ominus B, \uparrow (\alpha + \beta + 5 \times 10^{e(A \ominus B)-(\mu+1)})]$;
multiplication: $[A, \alpha] \otimes [B, \beta] = [A \otimes B, \uparrow (\uparrow (\alpha\beta) + \uparrow (\alpha|B|) + \uparrow (\beta|A|) + 5 \times 10^{e(A \otimes B)-(\mu+1)})]$.

Remark 1. In the above addition, if $A \oplus B = 0$, then the term $5 \times 10^{e(A \oplus B)-(\mu+1)}$ in the error is dropped. In the case of general base b of floating point numbers, $5 \times 10^{e(A \oplus B)-(\mu+1)}$ is replaced by $\lceil \frac{1}{2} b \rceil \times b^{e(A \oplus B)-(\mu+1)}$, where $\lceil \frac{1}{2} b \rceil$ is the minimal integer not less than $\frac{1}{2} b$. It is similar for the subtraction and multiplication.

The adequacy of this definition will be clarified in Lemma 1 later. BC arithmetic may be a kind of *interval arithmetic* [9], more precisely *machine* (or *rounded*) *interval arithmetic* [1] or *circular arithmetic* [1,10], whose definitions slightly differ from ours. However, the emphasis of this paper is not on which arithmetic to choose, but on the idea of keeping track of errors and more importantly on the zero criterion of coefficients by estimation of the accumulated errors.

Consider BC polynomials with a given precision μ. We can add, subtract and multiply such polynomials as well as usual polynomials as follows:

$$[A, \alpha]t + [B, \beta]t = ([A, \alpha] \oplus_\mu [B, \beta])t \qquad (\text{or } [A, \alpha]t - [B, \beta]t = ([A, \alpha] \ominus_\mu [B, \beta])t),$$

$$\sum_i [A_i, \alpha_i]t_i \cdot \sum_j [B_j, \beta_j]u_j = \sum_{i,j} ([A_i, \alpha_i] \otimes_\mu [B_j, \beta_j])t_i u_j,$$

where t, t_i, u_j are power products. Here, for 0, we have

$$f + 0 = 0 + f = f,$$

for any BC polynomial f.

Moreover, as a convention, we apply the following laws:

$$\mathbf{0} \cdot t = 0, \qquad -([A, \alpha]t) = [-A, \alpha]t.$$

Note that in general the set of BC polynomials is not a ring because, as mentioned above, the arithmetic operators are not associative and distributive in the bracket coefficients.

Now we are prepared to define the S-BC polynomial and BC reduction. Given an admissible term ordering, $LP(f)$ denotes the leading power product of a polynomial or BC polynomial f. If we write $f = f' + rest(f)$, then f' is a term of f and $rest(f)$ denotes the other terms of f.

Definition 5 (*S-BC polynomial*). Let f and g be BC polynomials with precision μ $[A, \alpha]LP(f)+rest(f)$ and $[B, \beta]LP(g)+rest(g)$, respectively. Also let LCM be $lcm(LP(f), LP(g))$. Then the S-BC polynomial of f and g with precision μ is the BC polynomial

$$[B, \beta] \cdot \frac{LCM}{LP(f)} \cdot f - [A, \alpha] \cdot \frac{LCM}{LP(g)} \cdot g$$

(with precision μ), denoted S-BCpoly$_\mu(f, g)$.

Definition 6 (*BC reduction*). Let f be a BC polynomial and F a finite set of BC polynomials with precision μ. Then

$$f \overset{BC}{\to}_F h(\text{“}f \text{ BC reduces to } h \text{ modulo } F\text{”}) \quad \text{with precision} \mu$$

$$\text{iff} \quad f \overset{BC}{\to}_{g,u} \text{ and } h = [B, \beta] \cdot f - [A, \alpha]u \cdot g \quad (\text{with precision } \mu),$$

where $f \overset{BC}{\to}_{g,u}$ (“f is BC reducible using g and u”) iff there are $g \in F$ and a power product u such that $f = [A, \alpha] \cdot u \cdot LP(g) + rest(f)([A, \alpha] \neq \mathbf{0})$, $g = [B, \beta]LP(g) + rest(g)$.

The *BC normal form* of f modulo F with precision μ, denoted BC-NForm$_\mu(f, F)$, is defined using BC reduction, in a manner similar to the conventional normal form.

Remark 2. The S-BC polynomial and BC reduction slightly differ from their conventional definitions when using polynomials having coefficients in a field. However, they have their natural definitions when working with polynomials having coefficients in a ring that is not a field, because the aim is to avoid a division of bracket coefficients.

In both the S-BC polynomial and BC reduction, the resulting coefficients have only two types: *product type*

$$[A, \alpha] \otimes [B, \beta]$$

and *product-difference type*

$$[A, \alpha] \otimes [B, \beta] \ominus [C, \gamma] \otimes [D, \delta].$$

Because in Buchberger's algorithm, any transformation of polynomials is either an S-polynomial calculation or a polynomial reduction, it suffices to consider these two types only as bracket coefficient calculations.

3.2. The key theorem

We introduce an algorithm called **R-GB** which we want to mimic. This is a slightly modified version of Buchberger's algorithm. Only the definitions of the S-polynomial and reduction are changed so that they correspond to Definitions 5 and 6 for BC polynomials. That is, in **R-GB**, for real polynomials $f = A \cdot LP(f) + rest(f)$ and $g = B \cdot LP(g) + rest(g)$, we define the *S-R polynomial* of f and g by

$$B \cdot \frac{LCM}{LP(f)} \cdot f - A \cdot \frac{LCM}{LP(g)} \cdot g.$$

Also for a polynomial f and a finite set F of polynomials, $f \xrightarrow{R}_F h$ (*R-reduction*) iff there are $g \in F$ and a power product u such that $f = A \cdot u \cdot LP(g) + rest(f)(A \neq 0)$, $g = B \cdot LP(g) + rest(g)$, and $h = B \cdot f - A \cdot u \cdot g$.

It is clear that **R-GB** gives the same result as Buchberger's algorithm up to coefficient normalization (i.e. making all the resulting polynomials monic).

Now let E be a (possibly zero) real coefficient of a polynomial[3] at an arbitrary step of **R-GB**. Let us consider how E arises. Let $\mathcal{C}$ be the set of all coefficients in the input polynomials. Let $\mathcal{C} = \{A_i\}_{i\in[1,l]}$. For each $i \in [1, l]$, introduce a new indeterminate $\tilde{A}_i$ which corresponds one to one with $A_i \in \mathcal{C}$. Let $\tilde{\mathcal{C}} = \{\tilde{A}_i\}_{i\in[1,l]}$ and $\mathcal{X}$ be the set (formally) generated by $\tilde{\mathcal{C}}$ with two binary relations "$\cdot$" and "$-$". (As the "minus" in usual arithmetic, this "$-$" may also work as a unary operator. That is, if $x \in \mathcal{X}$, then $-x \in \mathcal{X}$.) E at an nth step ($n \geqslant 2$) is the result of either $\pm AB$ (*product type*) where A and B were coefficients at an earlier step or $A - B$ (*product-difference type*) where $A = ab$ and $B = cd$ where a, b, c and d were coefficients at an earlier step, because E arises by S-R polynomials and R-reductions. Then from E we can define $\tilde{E} \in \mathcal{X}$ inductively as follows:

$$\tilde{E} = \begin{cases} \tilde{A} \cdot \tilde{B} \text{ (resp. } -\tilde{A} \cdot \tilde{B}) & \text{if } E \text{ is the result of } AB \text{ (resp. } -AB), \\ \tilde{A} - \tilde{B} & \text{if } E \text{ is the result of } A - B. \end{cases}$$

Moreover, let $\mathcal{BC}$ be the set of bracket coefficients and their negatives. For a precision μ we define a map $\sigma_\mu : \mathcal{X} \to \mathcal{BC}$ by the following:

$$\sigma_\mu(\tilde{A}_i) = [\langle A_i \rangle_\mu, \alpha_i],$$

where $\langle A_i \rangle_\mu$ is the floating point approximation of A_i to precision μ, α_i is its roundoff error $5 \times 10^{-(\mu+1)} \times 10^{e(\langle A_i \rangle_\mu)}$, and

For $x, y \in \mathcal{X}$,

$$\sigma_\mu(-x) = -\sigma_\mu(x), \quad \sigma_\mu(x \cdot y) = \sigma_\mu(x) \otimes_\mu \sigma_\mu(y), \quad \sigma_\mu(x - y) = \sigma_\mu(x) \ominus_\mu \sigma_\mu(y).$$

Recall Definition 4 for BC arithmetic. For E we hereby write

$$[\langle E \rangle_\mu, \epsilon_\mu]$$

as the result of $\sigma_\mu(\tilde{E})$.

For example, if $\tilde{E} = (\tilde{\frac{2}{5}}) \cdot ((\tilde{\frac{1}{3}}) \cdot (\tilde{\frac{3}{4}}) - (\tilde{\frac{5}{6}}) \cdot (\tilde{\frac{1}{7}})) - (\tilde{\frac{1}{2}}) \cdot (\tilde{\frac{2}{3}})$, then this means that E arises by $\frac{2}{5} \times (\frac{1}{3} \times \frac{3}{4} - \frac{5}{6} \times \frac{1}{7}) - \frac{1}{2} \times \frac{2}{3}$, and $\langle E \rangle_5$ is the result of $0.40000 \otimes (0.33333 \otimes 0.75000 \ominus 0.83333 \otimes 0.14286) \ominus 0.50000 \otimes 0.66667$. As to the error, in general, we have the following lemma.

Lemma 1 (Error propagation). $|\langle E \rangle_\mu - E| \leqslant \epsilon_\mu$ *for all* μ.

Proof. By induction on the structure of $\tilde{E}$.

[3] The result or an intermediate polynomial (e.g. $B \cdot (LCM/(LP(f)) \cdot f$ in the above notation) of an S-R polynomial calculation or an R-reduction.

Base case: $\tilde{E} = \tilde{A}_i$. Obvious since $|\langle A_i\rangle_\mu - A_i| \leqslant \alpha_i = \epsilon_\mu$.

Multiplication case: $\tilde{E} = \tilde{A} \cdot \tilde{B}$ or $-\tilde{A} \cdot \tilde{B}$. It suffices to see the case $\tilde{E} = \tilde{A} \cdot \tilde{B}$ only. Let $|\langle A\rangle_\mu - A| \leqslant \alpha$ and $|\langle B\rangle_\mu - B| \leqslant \beta$ by induction hypothesis. Let $\sharp$ refer to the equation: $|\langle A\rangle_\mu \otimes \langle B\rangle_\mu - AB|$.

When $\langle A\rangle_\mu \otimes \langle B\rangle_\mu \neq 0$:

$$\begin{aligned}\sharp &= |(\langle A\rangle_\mu - A)(B - \langle B\rangle_\mu) + \langle B\rangle_\mu(\langle A\rangle_\mu - A) + \langle A\rangle_\mu(\langle B\rangle_\mu - B) + (\langle A\rangle_\mu \otimes \langle B\rangle_\mu - \langle A\rangle_\mu\langle B\rangle_\mu)| \\ &\leqslant \alpha\beta + \alpha|\langle B\rangle_\mu| + \beta|\langle A\rangle_\mu| + 5 \times 10^{-(\mu+1)} \times 10^{e(\langle E\rangle_\mu)} \leqslant \epsilon_\mu.\end{aligned}$$

When $\langle A\rangle_\mu \otimes \langle B\rangle_\mu = 0$: $\langle A\rangle_\mu\langle B\rangle_\mu = 0$ and so

$$\begin{aligned}\sharp &= |(\langle A\rangle_\mu - A)(B - \langle B\rangle_\mu) + \langle B\rangle_\mu(\langle A\rangle_\mu - A) + \langle A\rangle_\mu(\langle B\rangle_\mu - B)| \leqslant \alpha\beta + \alpha|\langle B\rangle_\mu| + \beta|\langle A\rangle_\mu| \\ &\leqslant \epsilon_\mu \quad \text{(By Remark 1).}\end{aligned}$$

Subtraction case: $\tilde{E} = \tilde{A} - \tilde{B}$. Let $|\langle A\rangle_\mu - A| \leqslant \alpha$ and $|\langle B\rangle_\mu - B| \leqslant \beta$ by induction hypothesis. Let $\sharp\sharp$ refer to the equation: $|(\langle A\rangle_\mu \ominus \langle B\rangle_\mu) - (A - B)|$.

When $\langle A\rangle_\mu \ominus \langle B\rangle_\mu \neq 0$: If $\langle B\rangle_\mu \neq 0$, then there are two cases:
(1) $\langle A\rangle_\mu \ominus \langle B\rangle_\mu \neq \langle A\rangle_\mu$ (normal case), and
(2) $\langle A\rangle_\mu \ominus \langle B\rangle_\mu = \langle A\rangle_\mu$ (*loss of information*).

In case (1),

$$\begin{aligned}\sharp\sharp &= |(\langle A\rangle_\mu - A) + (B - \langle B\rangle_\mu) + \{(\langle A\rangle_\mu \ominus \langle B\rangle_\mu) - (\langle A\rangle_\mu - \langle B\rangle_\mu)\}| \\ &\leqslant \alpha + \beta + 5 \times 10^{-(\mu+1)} \times 10^{e(\langle E\rangle_\mu)} \\ &\leqslant \epsilon_\mu.\end{aligned}$$

In case (2), it is easy to see from floating point arithmetic that $e(\langle B\rangle_\mu) \leqslant e(\langle A\rangle_\mu) - \mu$ and when the equality holds,

$$|\text{mantissa of } \langle B\rangle_\mu| \leqslant 0.\overbrace{500\ldots00}^{\mu}$$

$$\left(\text{i.e.} \leqslant .0.\overbrace{500\ldots00}^{\mu} \text{ if } \langle B\rangle_\mu \text{ is positive, and } < .0.\overbrace{500\ldots00}^{\mu} \text{ otherwise}\right).$$

Hence when the equality holds, $\langle B\rangle_\mu \leqslant 0.5 \times 10^{e(\langle B\rangle_\mu)} = 0.5 \times 10^{e(\langle A\rangle_\mu)-\mu}$. Then $\sharp\sharp = |(\langle A\rangle_\mu - A) + (B - \langle B\rangle_\mu) + \langle B\rangle_\mu| \leqslant \alpha + \beta + |\langle B\rangle_\mu| \leqslant \alpha + \beta + 5 \times 10^{e(\langle E\rangle_\mu)-(\mu+1)} \leqslant \epsilon_\mu$. If the equality does not hold, $\sharp\sharp \leqslant \alpha + \beta + |\langle B\rangle_\mu| < \alpha + \beta + 1 \times 10^{e(\langle E\rangle_\mu)-(\mu+1)} < \epsilon_\mu$.
On the other hand, if $\langle B\rangle_\mu = 0$, then $\sharp\sharp = |(\langle A\rangle_\mu - A) + (B - \langle B\rangle_\mu)| \leqslant \alpha + \beta < \epsilon_\mu$.
When $\langle A\rangle_\mu \ominus \langle B\rangle_\mu = 0$: we have $\langle A\rangle_\mu - \langle B\rangle_\mu = 0$ from floating point arithmetic, and hence $\sharp\sharp = |(\langle A\rangle_\mu - A) + (B - \langle B\rangle_\mu)| \leqslant \alpha + \beta \leqslant \epsilon_\mu$ □

Lemma 2 (Error convergence). *For the notation as above,*

$$\lim_{\mu\to\infty} \epsilon_\mu = 0.$$

Proof. By induction on the structure of $\tilde{E}$.

Base case: $\tilde{E} = \tilde{A}_i$. Since $\alpha_i = 5 \times 10^{-(\mu+1)} \times 10^{e(\langle A_i \rangle_\mu)}$ and $e(\langle A_i \rangle_\mu)$ is bounded on μ, $\epsilon_\mu = \alpha_i \to 0$.

Multiplication case: $\tilde{E} = \tilde{A} \cdot \tilde{B}$ or $-\tilde{A} \cdot \tilde{B}$. It suffices to see the case $\tilde{E} = \tilde{A} \cdot \tilde{B}$ only. Let α and β be the errors of $\langle A \rangle_\mu$ and $\langle B \rangle_\mu$, respectively. When $\langle E \rangle_\mu \neq 0$, $\epsilon_\mu = \uparrow (\uparrow (\alpha\beta) + \uparrow (\alpha|\langle B \rangle_\mu|) + \uparrow (\beta|\langle A \rangle_\mu|) + 5 \times 10^{e(\langle E \rangle_\mu)-(\mu+1)})$. By induction hypothesis, we have $\alpha \to 0$ and $\beta \to 0$. Since $|\langle A \rangle_\mu|$, $|\langle B \rangle_\mu|$ and $e(\langle E \rangle_\mu)$ are bounded on μ, $\epsilon_\mu \to 0$. Similar for the case $\langle E \rangle_\mu = 0$.

Subtraction case: $\tilde{E} = \tilde{A} - \tilde{B}$. Let α and β be as above. When $\langle E \rangle_\mu \neq 0$, $\epsilon_\mu = \uparrow (\alpha + \beta + 5 \times 10^{e(\langle E \rangle_\mu)-(\mu+1)})$. In the same way, $\epsilon_\mu \to 0$ is implied by that $\alpha \to 0$ and $\beta \to 0$ by induction hypothesis and that $e(\langle E \rangle_\mu)$ is bounded on μ. Similar for the case $\langle E \rangle_\mu = 0$. □

Remark 3. In general, $\{\epsilon_\mu\}_{\mu \geqslant 1}$ *may not* be monotonically decreasing, as can be seen in the multiplication case, since when $\mu' > \mu$, $|\langle A \rangle_{\mu'}| > |\langle A \rangle_\mu|$ may often occur.

The following is a key theorem for our algorithm.

Theorem 1 (Zero judgment). *Let E be a real coefficient of a polynomial at any step of* **R-GB**. *Then*

$$E = 0 \quad \textit{iff} \quad |\langle E \rangle_\mu| \leqslant \epsilon_\mu \textit{ for all } \mu.$$

Remark 4. The following also holds: "Let E be a real coefficient *of the product-difference type* of a polynomial at any step of **R-GB**. Then $E = 0$ iff $|\langle E \rangle_\mu| \leqslant \epsilon_\mu$ for all μ". In fact, if E is of the product type, E cannot be zero by the structure of **R-GB**.

Proof of Theorem 1. ($\Rightarrow$) is immediate from Lemma 1.

($\Leftarrow$): For any μ, $|E| \leqslant |E - \langle E \rangle_\mu| + |\langle E \rangle_\mu|$. $|E - \langle E \rangle_\mu| \leqslant \epsilon_\mu$ by Lemma 1 and $|\langle E \rangle_\mu| \leqslant \epsilon_\mu$ by assumption. Thus, $|E| \leqslant 2\epsilon_\mu$ for all μ. But by Lemma 2, $\epsilon_\mu \to 0$ as μ approaches infinity. Therefore, E must be 0. □

4. The algorithm

4.1. Description

For simplicity, we describe the algorithm **FP-GB** that computes an approximate Gröbner basis; it is based on the crude version of Buchberger's algorithm [3, Algorithm 6.2]. Thus here **R-GB** is the algorithm obtained by replacing S-polynomials and reductions in Algorithm 6.2 by S-R polynomials and R-reductions, respectively.

In light of Theorem 1, we propose a zero criterion for BC in **FP-GB** as follows:

For any bracket coefficient $[A, \alpha]$ such that A is of the product-difference type,

The zero criterion: $[A, \alpha] = 0$ iff $|A| \leqslant \alpha$.

This criterion will be shown to be adequate for our purpose in the correctness (Section 4.2) of **FP-GB**. Now let us describe **FP-GB**.

An admissible term ordering $<_T$ is given.

Algorithm FP-GB
Input: a finite set F of real polynomials and a natural number μ.
Output: a set G_μ of floating point polynomials with precision μ, such that $\{G_\mu\}_\mu$ makes a floating point Gröbner sequence of F.

$G :=$ **R-to-BC** (F, μ) % Data conversion to initialize
$B := \{\{f_1, f_2\} \mid f_1, f_2 \in G, f_1 \neq f_2\}$
While $B \neq \emptyset$ do
    $(f_1, f_2) :=$ a pair in B
    $B := B - \{\{f_1, f_2\}\}$
    $h :=$ **S-BCpoly** (f_1, f_2, μ)
    $h' :=$ **BC-NForm** (h, G, μ)
    if $h' \neq 0$ then
        if $h' =$ a bracket without the variables then Return $(\{1\})$ else
        $B := B \cup \{\{g, h'\} \mid g \in G\}$
        $G := G \cup \{h'\}$
BC-to-FP(G) % Data conversion to finish
% If necessary, the final errors can be viewed from G.

Subalgorithm R-to-BC (F, μ) % F is a finite set of real polynomials

$BCF := \emptyset$
For f in F do
    $n :=$ the number of terms in f
    $BCf := 0$
    For $i = 1$ to n do
        $A_i :=$ the ith coefficient in f
        $T_i :=$ the ith power product in f
        $\langle A_i \rangle_\mu :=$ the floating point approximation of A_i to precision μ
        $\alpha_i := 5 \times 10^{-(\mu+1)} \times 10^{e(\langle A_i \rangle_\mu)}$ $(*)$
        $BCf := BCf + [\langle A_i \rangle_\mu, \alpha_i] T_i$
    $BCF := BCF \cup \{BCf\}$
Remark $(*)$: If an input coefficient A_i is initially given by a floating point number with precision μ, then we can set $\alpha_i := 0$ for it.

Subalgorithm S-BCpoly (f_1, f_2, μ) % f_1 and f_2 are BC polynomials with precision μ

$f :=$ S-BCpoly$_\mu(f_1, f_2)$
$n :=$ the number of terms in f
$fZ := 0$
For $i = 1$ to n do
    $[E^i_\mu, \epsilon^i_\mu] :=$ the ith bracket coefficient in f
    $T_i :=$ the ith power product in f

if E^i_μ is of the product-difference type and $|E^i_\mu| \leqslant \epsilon^i_\mu$
then $fZ := fZ$ % **the zero criterion**
else $fZ := fZ + [E^i_\mu, \epsilon^i_\mu]T_i$

Subalgorithm BC-NForm (f, G, μ) % f is a BC polynomial and G is a finite set of BC polynomials with precision μ

$h := f$
While $\exists g \in G, u$ such that $h \xrightarrow{BC}_{g,u}$ do
choose $g \in G, u$ such that $h \xrightarrow{BC}_{g,u}$ and $u \cdot LP(g)$ is maximal (w.r.t. $<_T$)
$[A, \alpha] :=$ the bracket coefficient of $LP(g)$ in h
$[B, \beta] :=$ the leading bracket coefficient in g
$h := [B, \beta] \cdot h - [A, \alpha]u \cdot g$ (with precision μ)
$n :=$ the number of terms in h
$hZ := 0$
For $i = 1$ to n do
$[E^i_\mu, \epsilon^i_\mu] :=$ the ith bracket coefficient in h
$T_i :=$ the ith power product in h
if E^i_μ is of the product-difference type and $|E^i_\mu| \leqslant \epsilon^i_\mu$
then $hZ := hZ$ % **the zero criterion**
else $hZ := hZ + [E^i_\mu, \epsilon^i_\mu]T_i$
$h := hZ$.

Subalgorithm BC-to-FP (G) % G is a finite set of BC polynomials

$FPG := \emptyset$
For g in G do
$n :=$ the number of terms in g
$FPg := 0$
For $i = 1$ to n do
$[E_i, \epsilon_i] :=$ the ith coefficient in g
$T_i :=$ the ith power product in g
$FPg := FPg + E_i \cdot T_i$
$FPG := FPG \cup \{FPg\}$.

4.2. *Termination and correctness*

The termination of **FP-GB** can be shown in exactly the same manner as Buchberger's proof [4] for the conventional algorithm using Dickson's lemma [7], because the termination depends only on a property of a sequence of *power products*, not on *coefficients*.

Theorem 2 (Termination). *For any finite set F of real polynomials and any natural number μ,* **FP-GB** (F, μ) *terminates.*

Proof. Let t_i be the leading power product of the ith BC polynomial h_i adjoined to G in the course of **FP-GB** ($i = 1, 2, \ldots$). Assume $\exists j$ such that t_j is a multiple of t_k for some $k < j$. Then h_j is reducible using h_k. However, by the structure of **FP-GB**, h_j is a normal form modulo the $(j-1)$th set G including h_k. This is a contradiction. Thus the sequence $t_1, t_2, \ldots$ has a property that, for all j, t_j is not a multiple of any of its predecessors. Hence, by Dickson's lemma, this sequence must be finite. □

The correctness of **FP-GB** is the following.

Theorem 3 (Correctness). *Given a finite set F of real polynomials and a natural number μ, let $G_\mu =$* **FP-GB** *(F, μ). Then $\{G_\mu\}_\mu$ is a floating point Gröbner sequence of F.*

Proof. By comparison with **R-GB**. Let G be the result of **R-GB** (F). First of all, we prove that there is an M such that $Supp(G_\mu) = Supp(G)$ for all $\mu \geqslant M$. As mentioned in Section 3, it is obvious that if at any stage of **R-GB**, a coefficient E is zero iff **FP-GB** judges that the corresponding $[\langle E\rangle_\mu, \epsilon_\mu]$ is zero, then $Supp(G_\mu) = Supp(G)$. Here we could consider four cases:

	R-GB	**FP-GB**
(1)	$E = 0$	$[\langle E\rangle_\mu, \epsilon_\mu] = 0$
(2)	$E = 0$	$[\langle E\rangle_\mu, \epsilon_\mu] \neq 0$
(3)	$E \neq 0$	$[\langle E\rangle_\mu, \epsilon_\mu] = 0$
(4)	$E \neq 0$	$[\langle E\rangle_\mu, \epsilon_\mu] \neq 0$

Cases (1) and (4) are desired for our purpose. Case (2) is impossible by Theorem 1. However, the undesirable case (3) is possible. Therefore, it suffices to prove that there is an M such that for all $\mu \geqslant M$, case (3) does not occur throughout **FP-GB** (F, μ).

Let N be the sum of the numbers of S-R polynomial calculations and R-reductions in **R-GB** (F). Note that N is finite. Here when $N = 0$, it means $\sharp F = 1$, and hence we are done with this by **FP-GB** $(F, 1)$, i.e. taking 1 as M. Thus we assume $N \geqslant 1$.

Let $\{s_1, s_2, \ldots, s_N\}$ be the sequence of S-R polynomial calculations and R-reductions such that they are serially performed in **R-GB** (F) in this order. Put $\mathcal{S}_k = \{s_1, \ldots, s_k\}$ for $1 \leqslant k \leqslant N$.

We prepare the following lemma.[4]

Lemma 3. *For any $k \in [1, N]$, if for some M_0,* **FP-GB** *(F, M_0) causes no case* (3) *for $\mathcal{S}_k$, then there is an $M'(\geqslant M_0)$ such that for all $\mu \geqslant M'$,* **FP-GB** *(F, μ) causes no case* (3) *for $\mathcal{S}_k$.*

Proof. Assume the if part. If moreover no case (4) occurs in $\mathcal{S}_k$, then it means $E = 0$ for every coefficient E and hence no case (3) can occur for any μ. Otherwise, let $\{E_i, [\langle E_i\rangle_\mu, \epsilon_\mu^i]\}_{1 \leqslant i \leqslant l}$ be all the pairs in case (4) in $\mathcal{S}_k$. Put $C = \min_{i \in [1,l]}(|E_i|)$. Since $C \neq 0$ and l is finite, there is an M_1 such that $\mu \geqslant M_1 \Rightarrow \epsilon_\mu^i < \frac{1}{2}C$ for all i, by Lemma 2. For $i \in [1, l]$ and $\mu \geqslant M_1$, we have $\epsilon_\mu^i \geqslant |E_i - \langle E_i\rangle_\mu| \geqslant |E_i| - |\langle E_i\rangle_\mu|$ by Lemma 1 and the triangle inequality. Thus, $|\langle E_i\rangle_\mu| \geqslant |E_i| - \epsilon_\mu^i \geqslant C - \epsilon_\mu^i > \epsilon_\mu^i$. Therefore, $M' = \max(M_0, M_1)$ satisfies the subject of Lemma 3 by **the zero criterion**. □

[4] We need this lemma, since for an E, $\{\epsilon_\mu\}_\mu$ may not be monotonically decreasing as mentioned in Remark 3.

Proof of Theorem 3 (*continued*). Now if **FP-GB**$(F, 1)$ causes no case (3) for $\mathcal{S}_N$, then we are done by Lemma 3. Otherwise, let s_{k_1} be the first process such that case (3) occurs. Let $E_1, \ldots, E_m$ be all the coefficients in case (3) immediately after the process s_{k_1}. By an argument similar to the proof of Lemma 3, there is a μ_{k_1} such that $\mu \geqslant \mu_{k_1} \Rightarrow |\langle E_i \rangle_\mu| > \epsilon_\mu^i$ for all $i \in [1, m]$. Moreover, since **FP-GB**$(F, 1)$ causes no case (3) for $\mathcal{S}_{k_1-1}$, by Lemma 3 there is an M_{k_1} such that for all $\mu \geqslant M_{k_1}$, **FP-GB**(F, μ) causes no case (3) for $\mathcal{S}_{k_1-1}$. Therefore, when putting $\tilde{M}_{k_1} = \max(\mu_{k_1}, M_{k_1})$, **FP-GB**$(F, \tilde{M}_{k_1})$ causes no case (3) for $\mathcal{S}_{k_1}$.

Next, if **FP-GB**$(F, \tilde{M}_{k_1})$ causes no case (3) for $\mathcal{S}_N$, then we are done. Otherwise, let s_{k_2} be the first process such that case (3) occurs. Repeating this argument makes a strictly monotone increasing sequence $k_1 < k_2 < k_3 < \cdots$. But since N is finite, this sequence is finite and hence it proves the existence of M.

Moreover, as μ approaches infinity after M, by Lemmas 1 and 2, every coefficient of $\{G_\mu\}_{\mu \geqslant M}$ converges to the corresponding one of G. □

5. Examples

We can also consider an improved version of **FP-GB**, based on the improved version of Buchberger's algorithm [3, Algorithm 6.3], in a manner similar to **FP-GB**. In fact, the optimum choice of a pair for the S-polynomial, and Criterions 1 and 2 for detecting unnecessary reductions are all expressed only by the terminology of power products, not of coefficients. Obviously the termination and correctness of the improved **FP-GB** can be proved in the same manner as those of **FP-GB** in Section 4.2.

The original versions of **FP-GB** and **R-GB** are so costly in computing time that they are not appropriate for our experiments. Thus, in this section, we compare the experimental results of the improved **FP-GB** and the improved **R-GB**, where the latter algorithm gives a *reduced Gröbner basis* except that the resulting polynomials are not necessarily monic. Let us here call them simply **FP-GB** and **R-GB** as well.

We implemented them in Maple V/Sun SPARC [5]. For brevity and convenience on Maple, we coded the parts of error terms in Definition 4 by floating point arithmetic instead of round up arithmetic, assuming that the errors between them are negligible. We did not use the built-in function *gbasis*[5] in Maple for computing Gröbner bases even in the case of rationals, for the sake of fairness of the comparison between **FP-GB** and **R-GB**. Tables 1–7 show $G :=$ **R-GB**(F) and $G_\mu :=$ **FP-GB**(F, μ) $(1 \leqslant \mu \leqslant 10$ or $20)$ with cpu times for F in Examples 1–7 below, respectively. (Cpu times are not described in Tables 3 and 4. They were all within 1 s. for Example (3) and within about 3 s. for Example (4).) To preserve space, in Tables 6 and 7, each coefficient of G is a floating point expression with precision 10. tdeg (respectively plex) denotes total degree lexicographic (respectively purely lexicographic) term ordering. In the tables except for Table 7, the first precision $\tilde{M}$ satisfying $\mathrm{Supp}(G_{\tilde{M}}) = \mathrm{Supp}(G)$ is marked in bold. Note that $\tilde{M}$ may not necessarily satisfy the condition that $\mathrm{Supp}(G_\mu) = \mathrm{Supp}(G)$ for all $\mu \geqslant \tilde{M}$. In Table 7, the precision such that from it on, the series of supports *seems* to be stable is framed by a box. Here are the examples:

(1) $F = \{x(3x-1), x - \frac{1}{3}\}$, tdeg.

(2) $F = \{x(3x-1), x - 0.3333333333\}$, tdeg.

(3) $F = \{(x^2 - \frac{1}{9})^5, (x^2 + \frac{2}{3}x + \frac{1}{9})^5)\}$, tdeg.

(4) $F = \{x_2 - x_3 + 1, \frac{2}{3}x_1 + \frac{1}{3}x_2 - x_3 + \frac{5}{3}, -x_1 + x_2 + x_3 - 2\}$, plex with $x_1 > x_2 > x_3$.

[5] This function can be applied only over the rationals, but it was often much faster than the author's implementation of **R-GB**.

Table 1
Example 1

$G = \{x - \frac{1}{3}\}$ cpu time (s) = 0.130		
μ	G_μ	cpu time (s)
1	$\{1.x - 0.3\}$	0.233
2	$\{1.x - 0.33\}$	0.167
3	$\{1.x - 0.333\}$	0.200
4	$\{1.x - 0.3333\}$	0.184
5	$\{1.x - 0.33333\}$	0.184
6	$\{1.x - 0.333333\}$	0.183
7	$\{1.x - 0.3333333\}$	0.167
8	$\{1.x - 0.33333333\}$	0.150
9	$\{1.x - 0.333333333\}$	0.167
10	$\{1.x - 0.3333333333\}$	0.183
11	$\{1.x - 0.33333333333\}$	0.200
12	$\{1.x - 0.333333333333\}$	0.183
13	$\{1.x - 0.3333333333333\}$	0.200
14	$\{1.x - 0.33333333333333\}$	0.200
15	$\{1.x - 0.333333333333333\}$	0.183
16	$\{1.x - 0.3333333333333333\}$	0.200
17	$\{1.x - 0.33333333333333333\}$	0.216
18	$\{1.x - 0.333333333333333333\}$	0.184
19	$\{1.x - 0.3333333333333333333\}$	0.200
20	$\{1.x - 0.33333333333333333333\}$	0.200

Note: G_μ gives an *approximate GCD* of the input polynomials.

(5) $F = \{f_1, f_2, f_3\}$, plex with $x > y > z$, where
$f_1 = \frac{1}{7}x^2 - \frac{326548390854652}{272974017239}x + \frac{1263781236281}{712638126}y^2 + \frac{26872672361827}{7263188218281}z^2$,
$f_2 = \frac{3}{8}xy + \frac{12367812638123}{763812368213132}yz - \frac{63812638126}{77263812831}y$,
$f_3 = \frac{4}{9}x + \frac{327091270979304}{24122375460421}y + \frac{18467031595309203}{318405459032}z - \frac{356318063693141319}{6436561806418109}$.

(6) $F = \{\sqrt{2}ex^3y + \sqrt{3}xy + \sqrt{7}/e, (\sqrt{3}/e)x^2y^2 - \sqrt{7}xy + \frac{1}{11}e\sqrt{11}\}$, tdeg with $x > y$, where e is Napier's number (2.71828...).

(7) $F = \{f_1, f_2\}$, tdeg with $x > y$, where $f_1 = \sqrt{2}e/\pi x^3y + (\sqrt{3} + \pi)xy + \sqrt{7}/(e - \pi)$, $f_2 = (1 - e\sqrt{3})/e \cdot \pi x^2y^2 - (\sqrt{7} - e)xy + e/\sqrt{11}$.

Remarks and observations

(1) Example (3) is given in [6] as an example where, with *any* precision, using naive floating point computation always leads to an incorrect answer $\{1\}$. **FP-GB** gave a correct support from precision 7.

(2) Example (4) is a system of ill-conditioned linear equations as well as Example (1). That is, with any precision, naive floating point computation (Gaussion elimination) always makes x_2 non-zero, whereas the correct solution for x_2 is $x_2 = 0$. For example, in Maple, *fsolve* gave $x_2 = -0.10 \times 10^{-8}$ with precision 10, and $x_2 = -0.10 \times 10^{-998}$ with precision 10^3. **FP-GB** gave a correct support with the correct solution $x_2 = 0$ from precision 3.

Table 2
Example 2

$G = \{1\}$ cpu time (s) = 0.200

μ	G_μ	cpu time (s)
1	$\{1.x - 0.3\}$	0.150
2	$\{1.x - 0.33\}$	0.183
3	$\{1.x - 0.333\}$	0.134
4	$\{1.x - 0.3333\}$	0.134
5	$\{1.x - 0.33333\}$	0.116
6	$\{1.x - 0.333333\}$	0.150
7	$\{1.x - 0.3333333\}$	0.134
8	$\{1.x - 0.33333333\}$	0.150
9	$\{1.x - 0.333333333\}$	0.150
10	$\{1.x - 0.3333333333\}$	0.150
11	$\{1.x - 0.3333333333\}$	0.150
12	$\{1\}$	0.217
13	$\{1\}$	0.200
14	$\{1\}$	0.217
15	$\{1\}$	0.216
16	$\{1\}$	0.233
17	$\{1\}$	0.217
18	$\{1\}$	0.217
19	$\{1\}$	0.200
20	$\{1\}$	0.200

Table 3
Example 3

$G = \{\frac{6553600000}{31381059609}x^5 + \frac{32768000000}{94143178827}x^4 + \frac{65536000000}{282429536481}x^3 + \frac{65536000000}{847288609443}x^2 + \frac{32768000000}{2541865828329}x + \frac{6553600000}{7625597484987}\}$

Floating point expression of G with precision 10

$\{0.2088393471x^5 + 0.3480655785x^4 + 0.2320437190x^3 + 0.07734790633x^2 + 0.01289131772x + 0.0008594211815\}$

μ	G_μ
1	$\{1\}$
2	$\{1\}$
3	$\{1\}$
4	$\{1\}$
5	$\{1\}$
6	$\{1\}$
7	$\{0.21019x^5 + 0.348555x^4 + 0.2321459x^3 + 0.07736165x^2 + 0.01289275x + 0.0008595170\}$
8	$\{0.209028x^5 + 0.3481360x^4 + 0.23205825x^3 + 0.077349832x^2 + 0.012891517x + 0.00085943445\}$
9	$\{0.2088578x^5 + 0.34807241x^4 + 0.232045147x^3 + 0.0773480939x^2 + 0.0128913372x + 0.000859422479\}$
10	$\{0.20884360x^5 + 0.348067163x^4 + 0.2320440460x^3 + 0.07734794993x^2 + 0.01289132226x + 0.0008594214843\}$

Note: For reference, in the case of $\mu = 10$, the resulting set of BC polynomials before applying **BC-to-FP** was $\{[0.20884360, 0.0001874976625]x^5 + [0.348067163, 0.00006354676445]x^4 + [0.2320440460, 0.00001245452526]x^3 + [0.07734794993, 0.1524493493 \times 10^{-5}]x^2 + [0.01289132226, 0.1445756578 \times 10^{-6}]x + [0.0008594214843, 0.9638601869 \times 10^{-8}]\}$.

Table 4
Example 4

$G = \{\frac{2}{3}\boldsymbol{x_1} + \frac{2}{3}, -\frac{2}{3}\boldsymbol{x_2}, -\frac{2}{3}\boldsymbol{x_3} + \frac{2}{3}\}$

μ	G_μ
1	$\{-1.x_1, 1.x_2 - 1.x_3 + 1.\}$
2	$\{0.67x_1, -0.67x_2, -0.67x_3 + 0.6\}$
3	$\{0.445\boldsymbol{x_1} + 0.449, 0.667\boldsymbol{x_2}, 0.667\boldsymbol{x_3} - 0.667\}$
4	$\{0.6663x_1 + 0.666, -0.6663x_2, -0.6663x_3 + 0.6663\}$
5	$\{0.44445x_1 + 0.44449, 0.66667x_2, 0.66667x_3 - 0.66667\}$
6	$\{0.444445x_1 + 0.444449, 0.666667x_2, 0.666667x_3 - 0.666667\}$
7	$\{0.4444445x_1 + 0.4444454, 0.6666667x_2, 0.6666667x_3 - 0.666666\}$
8	$\{0.66666667x_1 + 0.6666668, -0.66666667x_2, -0.66666667x_3 + 0.6666666\}$
9	$\{0.444444445x_1 + 0.444444449, 0.666666667x_2, 0.666666667x_3 - 0.666666667\}$
10	$\{0.4444444445x_1 + 0.4444444449, 0.6666666667x_2, 0.6666666667x_3 - 0.6666666667\}$

In general Ref. [1] gives a sufficient condition for the feasibility of the Gaussion elimination with intervals. We expect that using intervals and (the interval version of) **the zero criterion** will *always* make the Gaussion elimination successful, provided we apply our versions of the S-polynomial and reduction without division.

(3) In Example (5), the reason why only the leading coefficients have small numerators and denominators is that $\tilde{M}(=6)$ was small. According to our experiments, $\tilde{M}$ was 14 when they were all replaced by a rational number with numerator and denominator of about 15 digits. (It took 261.150 s. for $\tilde{M} = 14$, whereas **R-GB** required 1278.63 s.) In the case of rationals, in general, we have a conjecture that $\tilde{M}$ depends largely on the sizes of the numerators and denominators of the leading coefficients of input polynomials.

(4) Our approach could be useless if input polynomials contain a polynomial of too high degree or with too many terms, because even using floating point computation does not solve the problem of the number of reductions or the number of polynomials that we must retain to make S-polynomials. In fact, it was not possible to compute a lexicographic floating point Gröbner basis of the"Rose system" (see [2]) which has a polynomial with 21 terms (3 variables, total degree 8), using **FP-GB** within a reasonable time. Our approach will be useful when *the growth of coefficients* is the main reason why the computation of a Gröbner basis by the usual Buchberger algorithm is slow (see Examples (5)–(7)).

6. Conclusion

Our algorithm **FP-GB** is often more efficient than the conventional **R-GB** algorithm, in particular when the coefficients of input data are complicated, such as real numbers including irrational or transcendental numbers, and the growth of intermediate coefficients is the major factor determining the computational cost. Moreover, **FP-GB** can give a correct support even for an ill-conditioned system from a finite precision. In other words, using bracket coefficients and **the zero criterion** stabilizes Buchberger's algorithm which is numerically unstable. This method could be extented to other approximations as well.

Table 5
Example 5

Floating point expression of G with precision 10

$$= \{-0.7486148880 \times 10^{31}\boldsymbol{x} + 0.2938720185 \times 10^{40}\boldsymbol{z^3} + 0.1954874591 \times 10^{39}\boldsymbol{z^2} - 0.3745685851 \times 10^{36}\boldsymbol{z} + 0.1768674726 \times 10^{33},$$
$$- 0.1684383498 \times 10^{32}\boldsymbol{y} - 0.2167251711 \times 10^{39}\boldsymbol{z^3} - 0.1441683806 \times 10^{38}\boldsymbol{z^2} - 0.4442206312 \times 10^{35}\boldsymbol{z} + 0.5572270445 \times 10^{32},$$
$$- 0.5081835293 \times 10^{76}\boldsymbol{z^4} - 0.3068656908 \times 10^{75}\boldsymbol{z^3} + 0.8958744250 \times 10^{72}\boldsymbol{z^2} - 0.8274384887 \times 10^{69}\boldsymbol{z} + 0.2446596580 \times 10^{66}\}$$

cpu time (s) = 369.16

μ	G_μ	cpu time (s)
1	$\{-0.4 \times 10^{11}z^2, -8.x + 1000., -20.y - 100000.z\}$	1.833
2	$\{-0.46 \times 10^{168}x - 0.48 \times 10^{173}z + 0.61 \times 10^{171}, -0.10 \times 10^{169}y, 0.24 \times 10^{152}z^2 + 0.14 \times 10^{151}z - 0.18 \times 10\}$	12.000
3	$\{-0.349 \times 10^{126}z^2, 0.136 \times 10^{143}y + 0.478 \times 10^{146}z - 0.464 \times 10^{143}, 0.607 \times 10^{142}x\}$	8.817
4	$\{.3025 \times 10^{1576}y + 0.7964 \times 10^{1579}z - 0.1002 \times 10^{1577},$ $0.1344 \times 10^{1576}x + 0.6750 \times 10^{1580}z - 0.3169 \times 10^{1577}, -0.9180 \times 10^{771}z^2\}$	102.600
5	$\{-0.11300 \times 10^{2591}y + 0.37380 \times 10^{2591}, 0.12153 \times 10^{1106}z, -0.50220 \times 10^{2590}x + 0.11864 \times 10^{2592}\}$	128.216
6	$\{0.537759 \times 10^{25}\boldsymbol{x} - 0.211101 \times 10^{34}\boldsymbol{z^3} - 0.140427 \times 10^{33}\boldsymbol{z^2} + 0.269066 \times 10^{30}\boldsymbol{z} - 0.127046 \times 10^{27},$ $0.120996 \times 10^{26}\boldsymbol{y} + 0.155683 \times 10^{33}\boldsymbol{z^3} + 0.103562 \times 10^{32}\boldsymbol{z^2} + 0.319102 \times 10^{29}\boldsymbol{z} - 0.400280 \times 10^{26},$ $-0.262230 \times 10^{64}\boldsymbol{z^4} - 0.158347 \times 10^{63}\boldsymbol{z^3} + 0.462286 \times 10^{60}\boldsymbol{z^2} - 0.42695 \times 10^{57}\boldsymbol{z} + 0.12621 \times 10^{54}\}$	24.650
7	$\{-0.5377591 \times 10^{25}x + 0.2110991 \times 10^{34}z^3 + 0.1404259 \times 10^{33}z^2 - 0.2690671 \times 10^{30}z + 0.1270510 \times 10^{27},$ $-0.1209958 \times 10^{26}y - 0.1556817 \times 10^{33}z^3 - 0.1035615 \times 10^{32}z^2 - 0.3191010 \times 10^{29}z + 0.4002777 \times 10^{26},$ $-0.2622272 \times 10^{64}z^4 - 0.1583455 \times 10^{63}z^3 + 0.4622788 \times 10^{60}z^2 - 0.426967 \times 10^{57}z + 0.126248 \times 10^{54}\}$	24.567
8	$\{-0.74861510 \times 10^{31}x + 0.29387212 \times 10^{40}z^3 + 0.19548753 \times 10^{39}z^2 - 0.37456868 \times 10^{36}z + 0.17686754 \times 10^{33},$ $-0.16843840 \times 10^{32}y - 0.21672524 \times 10^{39}z^3 - 0.14416843 \times 10^{38}z^2 - 0.44422077 \times 10^{35}z + 0.55722718 \times 10^{32},$ $-0.50818387 \times 10^{76}z^4 - 0.30686591 \times 10^{75}z^3 + 0.8958751 \times 10^{72}z^2 - 0.8274391 \times 10^{69}z + 0.2446602 \times 10^{66}\}$	30.100
9	$\{0.115073820 \times 10^{28}x - 0.451727265 \times 10^{36}z^3 - 0.300494807 \times 10^{35}z^2 + 0.575770523 \times 10^{32}z - 0.27187297 \times 10^{29},$ $0.258916096 \times 10^{28}y + 0.333140493 \times 10^{35}z^3 + 0.221609356 \times 10^{34}z^2 + 0.68283661 \times 10^{31}z - 0.85654515 \times 10^{28},$ $0.120076142 \times 10^{69}z^4 + 0.725077583 \times 10^{67}z^3 - 0.211681686 \times 10^{65}z^2 + 0.19551130 \times 10^{62}z - 0.5780940 \times 10^{58}\}$	27.483
10	$\{0.5377596431 \times 10^{25}x - 0.2110998780 \times 10^{34}z^3 - 0.1404263631 \times 10^{33}z^2 + 0.2690674097 \times 10^{30}z - 0.1270508920 \times 10^{27},$ $0.1209959197 \times 10^{26}y + 0.1556822505 \times 10^{33}z^3 + 0.1035618421 \times 10^{32}z^2 + 0.3191012254 \times 10^{29}z - 0.4002781954 \times 10^{26},$ $-0.2622286133 \times 10^{64}z^4 - 0.1583462665 \times 10^{63}z^3 + 0.4622816264 \times 10^{60}z^2 - 0.426967888 \times 10^{57}z + 0.126247231 \times 10^{54}\}$	24.617

Note. Even for $\mu = 100$, it took only 35.250 s.

Unfortunately, up to now, we have not had a reasonable upper bound on M such that $Supp(G_\mu) = Supp(G)$ for all $\mu \geqslant M$. That is, we could not strictly determine M in advance where G_M is an approximate Gröbner basis. Another important issue would be finding a good test for deciding if $Supp(G_\mu) = Supp(G)$ for a resulting G_μ. In practice, currently, one should guess M by detecting a subsequence of $\{G_\mu\}_\mu$ that seems to be stable on supports or seems to converge coefficientwise.

We want to stress, however, that our approach is not only theoretically well-founded, but also could be applied to solve many problems correctly or efficiently which have never been solved satisfactorily before using naive floating point computation, in approximate algebraic manipulation from gcd or Gaussian elimination to Gröbner basis calculation.

Table 6
Example 6

Floating point expression of G with precision 10
$= \{-0.1325528855 \times 10^9 \boldsymbol{y^3} - 0.1085028162 \times 10^{10} \boldsymbol{x} - 0.3879281551 \times 10^{10} \boldsymbol{y},$
$0.3443764079 \times 10^9 \boldsymbol{x^2} + 0.8401300998 \times 10^7 \boldsymbol{y^2} + 0.4145701265 \times 10^9,$
$0.3975253926 \times 10^7 \boldsymbol{xy} + 492680.0344 \boldsymbol{y^2} - 0.1293658022 \times 10^7\}$
cpu time (s) = 43.21

μ	G_μ	cpu time (s)
1	$\{-80.x + 30.y, 7000.y^4 + 20000.y^2 + 500000.\}$	2.267
2	$\{83.x^2 + 100., -2400.xy - 690.y^2 + 730., -63000.y^3 - 620000.x\}$	2.567
3	$\{1\}$	2.900
4	$\{1\}$	2.700
5	$\{-229890.\boldsymbol{y^3} - 0.18815 \times 10^7 \boldsymbol{x} - 0.67274 \times 10^7 \boldsymbol{y},$ $-597150.\boldsymbol{x^2} - 14570.\boldsymbol{y^2} - 718900.,$ $-6892.6\boldsymbol{xy} - 854.35\boldsymbol{y^2} + 2243.1\}$	3.550
6	$\{-0.132550 \times 10^9 y^3 - 0.108499 \times 10^{10} x - 0.387913 \times 10^{10} y,$ $0.344366 \times 10^9 x^2 + 0.840125 \times 10^7 y^2 + 0.414559 \times 10^9,$ $0.397519 \times 10^7 xy + 492664.y^2 - 0.129365 \times 10^7\}$	4.583 4.983
7	$\{-0.1325529 \times 10^9 y^3 - 0.1085029 \times 10^{10} x - 0.3879282 \times 10^{10} y,$ $0.3443766 \times 10^9 x^2 + 0.8401299 \times 10^7 y^2 + 0.4145703 \times 10^9,$ $0.3975255 \times 10^7 xy + 492680.1 y^2 - 0.1293659 \times 10^7\}$	3.566
8	$\{-229800.22y^3 - 0.18810585 \times 10^7 x - 0.67253136 \times 10^7 y,$ $-597027.96x^2 - 14564.909y^2 - 718719.25,$ $-6891.6968xy - 854.13454y^2 + 2242.7495\}$	
9	$\{-0.132552891 \times 10^9 y^3 - 0.108502819 \times 10^{10} x - 0.387928170 \times 10^{10} y,$ $0.344376416 \times 10^9 x^2 + 0.840130139 \times 10^7 y^2 + 0.414570140 \times 10^9,$ $0.397525401 \times 10^7 xy + 492680.049 y^2 - 0.129365806 \times 10^7\}$	4.583
10	$\{-0.1325528856 \times 10^9 y^3 - 0.1085028160 \times 10^{10} x - 0.3879281549 \times 10^{10} y,$ $0.3443764073 \times 10^9 x^2 + 0.8401301011 \times 10^7 y^2 + 0.4145701258 \times 10^9,$ $0.3975253922 \times 10^7 xy + 492680.0346 y^2 - 0.1293658021 \times 10^7\}$	4.967

Note. Even for $\mu = 100$, it took only 8.517 s.

Acknowledgements

This research is supported in part by the United States Army Research Office through the Army Center of Excellence for Symbolic Methods in Algorithmic Mathematics (ACSyAM), Mathematical Sciences Institute of Cornell University, Contract DAAL03-91-C-0027.

The author is grateful to Moss Sweedler for his helpful discussion and valuable comments in preparing this paper. Also the author wishes to thank James Davenport, Jerry Marsden, and Dana Scott for their suggestion of some relationship to interval arithmetic, and Tateaki Sasaki, Matu-Tarow Noda, and Vilmar Trevisan for their encouragement.

Table 7
Example 7

$G = ?$ cpu time (s) $> 3600\,(s)$.

μ	G_μ	cpu time (s)
2	$\{-12.x^2 - 55., 48000.xy + 160000.y^2 + 7700., -0.38 \times 10^8 y^3 + 580000.x\}$	3.200
3	$\{1\}$	1.550
4	$\{-0.2929 \times 10^{12}x, 0.9853 \times 10^{19}y^2 - 0.1421 \times 10^{18}\}$	3.883
5	$\{1\}$	3.133
6	$\{-0.304467 \times 10^8 y^3 - 57296.6x + 0.177025 \times 10^7 y,$ $57131.5x^2 - .349660 \times 10^8 y^2 + 503082.,$ $-5860.01xy + 134278.y^2 - 958.949\}$	4.367
7	$\{523698.7y^3 + 985.5738x - 30450.19y,$ $982.7229x^2 - 601411.7y^2 + 8653.376,$ $-100.7948xy + 2309.56y^2 - 16.49430\}$	3.883
8	$\{-0.30449275 \times 10^8 y^3 - 57303.400x + 0.17704401 \times 10^7 y,$ $57137.690x^2 - 0.34967823 \times 10^8 y^2 + 503126.47,$ $-5860.4880xy + 134285.50y^2 - 959.02183\}$	4.750
9	$\{-0.304493527 \times 10^8 y^3 - 57303.5247x + 0.177044409 \times 10^7 y,$ $57137.8102x^2 - 0.349679197 \times 10^8 y^2 + 503127.527,$ $-5860.49750xy + 134285.767y^2 - 959.023145\}$	4.767
10	$\{-0.3044934742 \times 10^8 y^3 - 57303.51814x + 0.1770443855 \times 10^7 y,$ $57137.80337x^2 - 0.3496791346 \times 10^8 y^2 + 503127.4531,$ $-5860.497001xy + 134285.7486y^2 - 959.0230326\}$	4.316

Notes. (1) **R-GB** (F) was not obtained after 3600 s.
(2) For $\mu = 1$, the constant term of f_1 is ill-defined since $e \sim \pi$.
(3) Even for $\mu = 100$, it took only 7.950 s.

References

[1] G. Alefeld and J. Herzberger, Introduction to Interval Computations, Computer Science and Applied Mathematics (Academic Press, New York, 1983).
[2] W. Boege, R. Gebauer and H. Kredel, Some examples for solving systems of algebraic equations by calculating Groebner bases, J. Symbolic. Comput. 1 (1986) 83–98.
[3] B. Buchberger, Gröbner bases: An algorithmic method in polynomial ideal theory, in: N.K. Bose, ed., Multidimensional Systems Theory (Reidel, Dordrecht, 1985) Chapter 6, 184–232.
[4] B. Buchberger, An algorithmical criterion for the solvability of algebraic systems of equations (German), Aequationes Mathematicae 4 (3) (1970) 374–383.
[5] B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan and S. M. Watt, First Leaves: A Tutorial Introduction to Maple V (Springer, Berlin, 1992).
[6] J.H. Davenport, Y. Siret and E. Tournier, Computer Algebra, Systems and Algorithms for Algebraic Computation (Academic Press, New York, 1993).
[7] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors, Amer. J. Math. 35 (1913) 413–426.
[8] D.E. Knuth, The Art of Computer Programming, Vol. 2 (Addison-Wesley, Reading, MA, 1969).
[9] R.E. Moore, Interval arithmetic and automatic error analysis in digital computing, Ph.D. Thesis, Mathematics Dept., Stanford Univ., October 1962.

[10] M. Petković, Iterative Methods for Simultaneous Inclusion of Polynomial Zeros, Lecture Notes in Mathematics, 1387 (Springer, Berlin, 1989).
[11] T. Sasaki and T. Takeshima, A modular method for Gröbner-basis construction over Q and solving system of algebraic equations, J. Inform. Process. 12 (4) (1989) 371–379.
[12] K. Shirayanagi, An algorithm to compute floating point Gröbner bases, in: T. Lee, ed., Mathematical Computation with Maple V: Ideas and Applications (Birkhäuser, Basel, 1993) 95–106.
[13] W. Trinks, On improving approximate results of Buchberger's algorithm by Newton's method, SIGSAM Bull. 18 (3) (1984) 7–11.
[14] F. Winkler, A p-adic approach to the computation of Gröbner bases, J. Symbolic. Comput. 6 (2–3) (1988) 287–304.