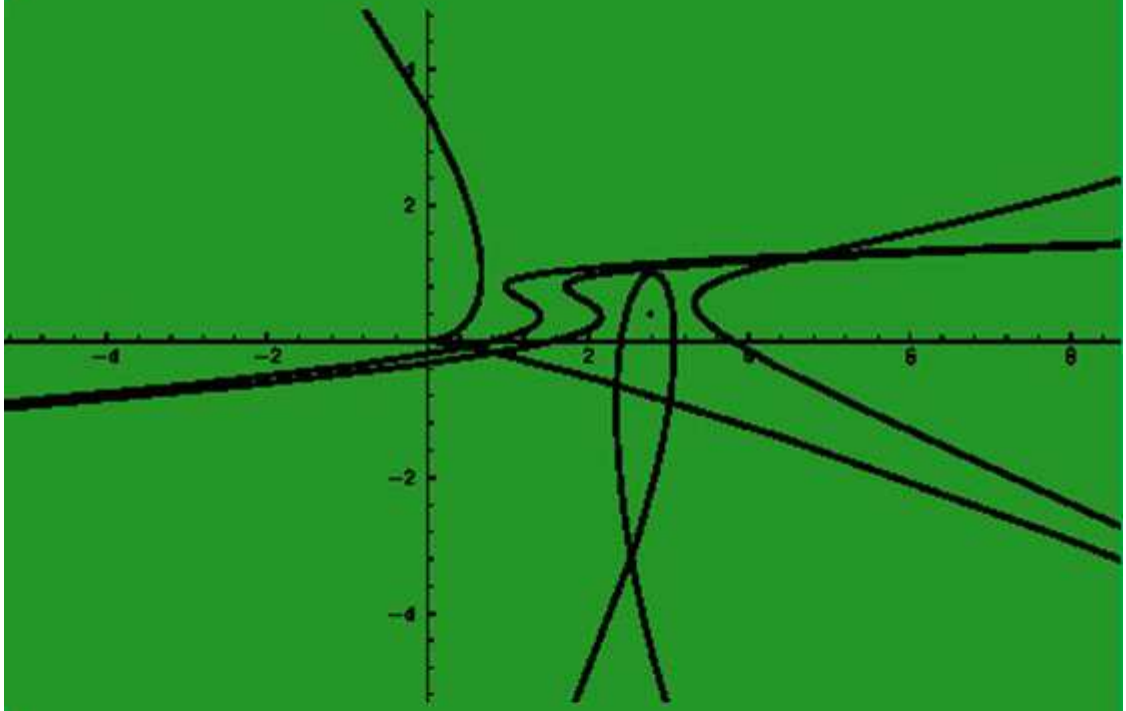
Communications of Japan Society for Symbolic and Algebraic Computation

2016
VOL.2



JSSAC

ISBN978-4-903027-28-9

Aims and Scope:
Communications of JSSAC (Japan Society of Symbolic and Algebraic Computations) is dedicated to researchers who have a special interest in symbolic and algebraic computation. Communications of JSSAC publishes original articles dealing with every aspect of symbolic and algebraic computation.

Research Areas Include but are not limited to:
・Theoretical and algorithmic issues of symbolic and algebraic computation
・Design and implementation of symbolic and algebraic computation systems
・Applications of symbolic and algebraic computation in education, science, engineering and industry, pure mathematics, etc.

Legal Requirements:
In order to submit a manuscript, at least one of the author(s) should be a member of JSSAC in principle

Manuscript Submission:
A manuscript must be written in English.
It also should be written in Latex.
A submission must include:
(1) a latex source file
(2) a dvi, ps or pdf file of (1)
(3) a title of the paper as well as the name(s) and affiliation(s) and mailing address(es) of the author(s)
(4) an abstract (no more than 150 words) and key words (5 or less)

For full and complete guide for authors, please refer to the following sites.

http://www.jssac.org/Editor/Style/index.html (in Japanese)
http://www.jssac.org/Editor/Style/index-e.html (in English)

Every submitted manuscript will undergo a standard review process and the acceptance for publication by the editorial board will be based on its originality, significance of contribution and its relevance to the scope of Communications of JSSAC.

Miscellaneous:
・The copyright of a published paper is transferred to JSSAC.
・Communications of JSSAC has no page charges.

Contents

Communications of JSSAC (2016)
Vol. 2, pp. 1 – 14

On the Inoue invariants of the puzzles of Sudoku type

Tetsuo Nakano*

Graduate School of Science and Engineering, Tokyo Denki University

Kenji Arai

Graduate School of Science and Engineering, Tokyo Denki University

Hiromasa Watanabe

Graduate School of Science and Engineering, Tokyo Denki University

(Received 29/Mar/2013 Accepted 13/Sept/2014)

Abstract

A Sudoku puzzle is a worldwide popular game, and is also an interesting object in combinatorics and computer algebra. Recently, Inoue applied his excellent algorithm on finding the singleton set solutions of a system of Boolean polynomial equations to the solution of the puzzles of Sudoku type. Further, by means of his algorithm, we have defined the Inoue invariant of puzzles of Sudoku type, which measures the mathematical difficulty of them.

The purpose of this note is study the Inoue invariants of the easier puzzles of Sudoku type, namely, 4-doku, diagonal 5-doku and diagonal 6-doku puzzles. Our main results show that all the 4-doku and diagonal 5-doku puzzles (with a unique solution) have the trivial Inoue invariant $(2, 1, 1)$ except 2 puzzles, whereas there exist many diagonal 6-doku puzzles with a non-trivial, big Inoue invariant.

1 Introduction

A Sudoku puzzle is a very popular game played by everybody in the world. Recently, numerous researches have been done on the mathematical (combinatorial) structure of Sudoku (see, for instance, the book [7] and references in it). Among them, Sato, Inoue and others [9, 10] studied it by means of Boolean Groebner bases.

Quite recently, Inoue [3] obtained an excellent method for finding the singleton set solutions of a system of Boolean polynomial equations. He also applied his algorithm to Sudoku and observed that relatively easy Sudoku puzzles can be solved without branches (namely without "else" procedure in Algorithm 34 of [3]).

*tnakano@mail.dendai.ac.jp

This work was supported by JSPS KAKENHI (23540057).

Stimulated by his observation, we went one step further and defined *the Inoue invariant* of puzzles of Sudoku type as follows ([5]). The performance of Inoue's algorithm for a Boolean polynomial ideal is well described by a tree diagram and we have defined the Inoue invariant of such an ideal as the triple of the basic numbers of this tree. We discovered that, in the case of those ideals arising from the puzzles of Sudoku type, this invariant is an excellent indicator of the difficulty of the puzzles by experiments. Thus we have defined the mathematical difficulty of the puzzles of Sudoku type as the Inoue invariant of their ideals. For example, in the case of Sudoku, the easier puzzles up to the middle level have the trivial Inoue invariant $(2, 1, 1)$, whereas the difficult ones have a non-trivial Inoue invariant. As far as we know, the biggest Inoue invariant so far is $(964, 558, 13)$, which is achieved by a 20-clues puzzle.

In this note, we study the Inoue invariants of the simpler puzzles of Sudoku type, namely 4-doku and the diagonal 5-doku. We computed many examples of them and got a conjecture that all the 4-doku and diagonal 5-doku puzzles with a unique solution have the the trivial Inoue invariant $(2, 1, 1)$. The purpose of this note is to give an answer to this conjecture. Our main results are summarized as follows.

Theorem 1 (Inoue invariants of 4-doku)
All the 4-doku puzzles with a unique solution have the trivial Inoue invariant $(2, 1, 1)$.

Theorem 2 (Inoue invariants of diagonal 5-doku)
(i) There exist exactly 30964554720 diagonal 5-doku puzzles with a unique solution.
(ii) They all have the trivial Inoue invariant $(2, 1, 1)$ *except the 2 puzzles* W_i $(i = 1, 2)$*, both of which have the Inoue invariant* $(4, 2, 2)$ *(see Table 5 in Section 5 for* W_i*,* $i = 1, 2$*).*

Thus our conjecture is false in the case of the diagonal 5-doku puzzles, but we discovered 2 special puzzles W_i $(i = 1, 2)$ with a non-trivial Inoue invariant. We also report a partial result on the Inoue invariants of the diagonal 6-doku puzzles, which shows that the there exist many diagonal 6-doku puzzles with a non-trivial, big Inoue invariant.

The contents of this note are as follows. In Section 2, we review Boolean Groebner bases, especially the stratified Boolean Groebner bases. In section 3, we summarize Inoue's algorithm and the Inoue invariants after [3, 5]. In section 4, we formulate the rules of puzzles of Sudoku type by a system of Boolean polynomial equations following [9, 10], and we report our main results in Section 5. In Appendix [6], which is separated from the main body of this note and put in our website, we summarize the detailed data and the programs used in the proof of our main results.

For the implementation of Inoue's algorithm, we have used the computer algebra system Magma [1].

Acknowledgment: we thank the referee for pointing out and correcting a critical mistake in the first version of this note.

2 Boolean Groebner Bases

In this section, we will briefly review the Groebner bases of ideals in the polynomial ring over a Boolean ring and the Boolean Groebner bases of ideals in a Boolean polynomial ring. For more details on Boolean Groebner bases, see [8, 9, 10, 11].

Let $\mathbf{B}$ be a Boolean ring. Namely, $\mathbf{B}$ is a commutative ring with an identity such that any element $a \in \mathbf{B}$ satisfies $a^2 = a$. For example, for a natural number m, $(\mathbb{F}_2)^m$ is a finite Boolean ring, where $\mathbb{F}_2 := \mathbb{Z}/2\mathbb{Z}$ is the field with 2 elements, and the addition and multiplication in $(\mathbb{F}_2)^m$ are defined componentwise. Conversely, any finite Boolean ring is isomorphic to $(\mathbb{F}_2)^m$ for some m by the Stone representation theorem.

Let $\mathbf{B}[x] = \mathbf{B}[x_1, \ldots, x_n]$ be the polynomial ring over $\mathbf{B}$ with n indeterminates with a given monomial order. For the notation on polynomials, we follow [2] as below.

Notation 3
(i) $\mathrm{LM}(f)$ *(resp.* $\mathrm{LT}(f)$*,* $\mathrm{LC}(f)$*,* $\mathrm{mdeg}(f)$*) is the leading monomial (resp. the leading term, the leading coefficient, the multidegree) of a polynomial* f *so that* $\mathrm{LT}(f) = \mathrm{LC}(f) \cdot \mathrm{LM}(f)$ *and* $\mathrm{LM}(f) = x^{\mathrm{mdeg}(f)}$ *hold.*
(ii) For monomials x^α *and* x^β*,* $x^\alpha \mid x^\beta$ *means that* x^α *divides* x^β*.*

We first show the division algorithm in $\mathbf{B}[x]$.

Theorem 4 (Division algorithm)
Given a polynomial f *and an ordered set of* s *polynomials* $F := (f_1, \ldots, f_s)$ *in* $\mathbf{B}[x]$*, we get an expression of the form* $f = a_1 f_1 + \cdots + a_s f_s + r$*, where* $(a_1, \ldots, a_s)$ *is the quotient and* r *the remainder, by the following algorithm:*
Algorithm variables: p *(intermediate dividend),* $B = (b_1, \ldots, b_s)$ *(intermediate quotient),* r *(intermediate remainder).*
Initial values: $p := f$*,* $B := (0, \ldots, 0)$*,* $r := 0$*.*
(i) If there exists i *such that* $\mathrm{LM}(f_i) \mid \mathrm{LM}(p)$ *and* $\mathrm{LC}(p) \cdot \mathrm{LC}(f_i) \neq 0$*, then take the least such* i *and redefine* $p := p - \mathrm{LC}(p) \cdot \frac{\mathrm{LM}(p)}{\mathrm{LM}(f_i)} \cdot f_i$ *and* $b_i := b_i + \mathrm{LC}(p) \cdot \frac{\mathrm{LM}(p)}{\mathrm{LM}(f_i)}$ *(***division step***).*
(ii) If there exists no i *such that* $\mathrm{LM}(f_i) \mid \mathrm{LM}(p)$ *and* $\mathrm{LC}(p) \cdot \mathrm{LC}(f_i) \neq 0$*, then redefine* $p := p - \mathrm{LT}(p)$ *and* $r := r + \mathrm{LT}(p)$ *(***remainder step***).*
This algorithm terminates (namely $p = 0$*) in a finite number of steps and yields an expression of division*

$$f = a_1 f_1 + \cdots + a_s f_s + r,$$

where r *satisfies the condition of the remainder:* $r = 0$ *or in case* $r \neq 0$*, any term* t *of* r *satisfies either* $\mathrm{LM}(f_i) \nmid \mathrm{LM}(t)$ *or* $\mathrm{LC}(t) \cdot \mathrm{LC}(f_i) = 0$ *in case* $\mathrm{LM}(f_i) \mid \mathrm{LM}(t)$ *for any* i*. Furthermore, if* $a_i f_i \neq 0$ *then* $\mathrm{mdeg}(a_i f_i) \leq \mathrm{mdeg}(f)$ *holds.*

This division algorithm in $\mathbf{B}[x]$ is quite similar to that in the polynomial ring over a field, except that one additional condition (the product of coefficients is not equal to 0) is necessary for the division step to occur.

For an ideal $I \subset \mathbf{B}[x]$, we denote by $\mathrm{LT}(I)$ the set of the leading terms of the elements (except 0) in I. We now define a Groebner basis of an ideal in $\mathbf{B}[x]$.

Definition 5 (Groebner bases)
Let $I \subset \mathbf{B}[x]$ *be an ideal and* $G := \{g_1, \ldots, g_s\} \subset I$ *a finite subset of* I*. We say* G *is a Groebner basis of* I *if* $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle$*.*

Based on the division algorithm, most of the results in [2, Chapter 2] hold with suitable modifications. Especially, the Buchberger criterion and algorithm hold (with slight modifications) so that we can obtain a Groebner basis of a finitely generated ideal by the Buchberger algorithm.

We next define reduced and stratified Groebner bases respectively. We denote by $\overline{f}^F$ the remainder of the division of f by F.

Definition 6 (Reduced Groebner bases)
Let G *be a Groebner basis of an ideal* I*.* G *is called reduced if* $\overline{g}^{G \setminus \{g\}} = g$ *holds for any* $g \in G$*.*

Reduced Groebner bases are not unique as shown in the following example.

Example 7
In the polynomial ring $(\mathbb{F}_2)^2[x]$ *of one variable,* $\{(1,0)x, (0,1)x\}$ *and* $\{(1,1)x\}$ *are both reduced Groebner bases of the same ideal* $I = \langle x \rangle$.

Definition 8 (Stratified Groebner bases)
Let $G \subset I$ *be a reduced Groebner basis.* G *is called a stratified Groebner basis if* $\mathrm{LM}(f) \neq \mathrm{LM}(g)$ *for any* $f, g \in G$, $f \neq g$.

Proposition 9 (Stratification algorithm)
Let $G \subset I$ *be a reduced Groebner basis. Divide* G *into several groups* $G_1, \ldots, G_t$ *according to leading monomials, where each member of a group has the same leading monomial and different groups have different leading monomials:* $G = G_1 \cup \cdots \cup G_t$ *(disjoint union). For each group* G_i, *set* $h_i := \sum_{g \in G_i} g$. *Then* $G' := \{h_1, \ldots, h_t\}$ *is a stratified Groebner basis of* I.

The following is the main theorem of the Groebner bases.

Theorem 10 (Existence and uniqueness of the stratified Groebner bases)
Fix a monomial order on $\mathbf{B}[x]$. *For a given finitely generated ideal* $I \subset \mathbf{B}[x]$, *a stratified Groebner basis exists and it is determined by* I *uniquely.*

For the actual computation of the stratified Groebner bases in the case of $\mathbf{B} = (\mathbb{F}_2)^m$, we use the "componentwise" method explained below. We first prepare some notations.

Consider the natural isomorphism $(\mathbb{F}_2)^m[x] \cong (\mathbb{F}_2[x])^m$ and let $\pi_i : (\mathbb{F}_2[x])^m \to \mathbb{F}_2[x]$ be the projection to the i-th component. For any $f \in (\mathbb{F}_2)^m[x]$, we set $f_i := \pi_i(f) \in \mathbb{F}_2[x]$ and call it the i-th component of f. Then the isomorphism $(\mathbb{F}_2)^m[x] \cong (\mathbb{F}_2[x])^m$ is given as $(\mathbb{F}_2)^m[x] \ni f \longleftrightarrow (f_1, \ldots, f_m) \in (\mathbb{F}_2[x])^m$. For an ideal $I \subset (\mathbb{F}_2)^m[x]$, we set $I_i := \{f_i \mid f \in I\} \subset \mathbb{F}_2[x]$ and call this the i-th component ideal of I.

The algorithm is based on the following theorem:

Theorem 11
Fix a monomial order on $(\mathbb{F}_2)^m[x]$ *and let* $I \subset (\mathbb{F}_2)^m[x]$ *be an ideal. For any* i $(1 \leq i \leq m)$, *let* $I_i \subset \mathbb{F}_2[x]$ *be the* i*-th component ideal of* I *and* G_i *the reduced Groebner basis of* I_i. *Then* $G := (G_1, 0 \ldots, 0) \cup (0, G_2, 0, \ldots, 0) \cup \cdots \cup (0, \ldots, 0, G_m)$ *is a reduced Groebner basis of* I, *where* $(G_1, 0, \ldots, 0) = \{(g, 0, \ldots, 0) \mid g \in G_1\}$ *etc.*.

Thus we can compute the stratified Groebner basis of I by Theorem 11 followed by the stratification process (Proposition 9).

We now turn to the Boolean Groebner bases. Since $\mathbf{B}[x]$ itself is not a Boolean ring, we set

$$\mathbf{B}(x) = \mathbf{B}(x_1, \ldots, x_n) := \mathbf{B}[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle.$$

$\mathbf{B}(x)$ is a Boolean ring and we call it *the Boolean polynomial ring* over $\mathbf{B}$ with n indeterminates. A monomial $x_1^{\alpha_1} \ldots x_n^{\alpha_n}$ is called *a Boolean monomial* if $\alpha_i \in \{0, 1\}$ for any i. We note any $f \in \mathbf{B}(x)$ can be written uniquely as $\sum_k c_k x^{\beta_k}$ where $c_k \in \mathbf{B}$ and x^{β_k} is a distinct Boolean monomial, which we call *the canonical representation of* f. Given a monomial order on $\mathbf{B}[x]$ and $f \in \mathbf{B}(x)$, we can define $\mathrm{LT}(f), \mathrm{LM}(f)$ and $\mathrm{LC}(f)$ using the canonical representation of f.

Definition 12 (Boolean Groebner bases)
Let $I \subset \mathbf{B}(x)$ *be an ideal and* $G := \{g_1, \ldots, g_s\} \subset I$ *a finite subset of* I. *We say* G *is a Boolean Groebner basis (a BG basis for short) of* I *if* $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_s) \rangle$.

The division algorithm (Theorem 4) works also in $\mathbf{B}(x)$ and we can define reduced and stratified BG bases as in Definition 6 and 8. Then the existence and uniqueness of the stratified BG bases hold too. We abbreviate the stratified BG bases as *the SBG bases* in the following sections.

We can compute a BG basis of the ideal $I = \langle F \rangle \subset \mathbf{B}(x)$ as follows. Compute a Groebner basis G of $\langle F \cup \{x_1^2 - x_1, \ldots, x_n^2 - x_n\}\rangle$ in $\mathbf{B}[x]$. Then $G' := G \setminus \{x_1^2 - x_1, \ldots, x_n^2 - x_n\}$ is a BG basis of I. Furthermore, if G is stratified, then G' is also stratified. We note that the componentwise method (Theorem 11) also works for the BG bases.

We finally refer to the Boolean Hilbert Nullstellensatz. For an ideal $I \subset \mathbf{B}(x)$, let $\mathbb{V}(I) := \{a \in \mathbf{B}^n \mid f(a) = 0 \text{ for any } f \in I\}$ be the affine variety defined by I.

Theorem 13 (Boolean Hilbert Nullstellensatz)
Let $I \subset \mathbf{B}(x)$ be a finitely generated ideal. Then the following assertions hold.
(i) $\mathbb{V}(I) = \phi$ if and only if I contains a non-zero constant.
(ii) Assume $\mathbb{V}(I) \neq \phi$. Then $f(x) \in I$ if and only if $f(a) = 0$ for any $a \in \mathbb{V}(I)$.

3 The Inoue algorithm and the Inoue invariants

In this section, we will briefly review the Inoue algorithm and the Inoue invariants ([3, 5]). The Inoue algorithm is an excellent and almost canonical method for computing the singleton set solutions of a system of Boolean polynomial equations.

We work in the Boolean polynomial ring $(\mathbb{F}_2)^m(x) = (\mathbb{F}_2)^m(x_1, \ldots, x_n)$. For an ideal $I \subset (\mathbb{F}_2)^m(x)$, we set

$$\mathbb{VS}(I) := \{(a_1, \ldots, a_n) \mid a_i \in \{e_1, \ldots, e_m\},\ f(a_1, \ldots, a_n) = 0 \text{ for any } f \in I\},$$

where $e_i := (\delta_{ij})_{j=1,\ldots,m}$. $\mathbb{VS}(I)$ is the set of singleton set solutions and we would like to compute this set.

The Inoue algorithm is based on the concept *almost solution polynomials* contained in the ideal. In the following, we set $E := \sum_{i=1}^{m} e_i = 1_{(\mathbb{F}_2)^m}$.

Definition 14 (Solution polynomial)
We call $f \in (\mathbb{F}_2)^m(x)$ of the form $f := E \cdot x_j + e_k = x_j + e_k$ for some j, k a solution polynomial.

We note that for a solution polynomial $f := x_j + e_k$, $f = 0$ is equivalent to $x_j = e_k$. We next define an almost solution polynomial.

Definition 15 (Almost solution polynomial)
(i) A polynomial $f(x) \in (\mathbb{F}_2)^m(x)$ is called an almost solution polynomial of type 1 (ASP of type 1 for short) if there exist j, k such that $e_k \cdot f(x) = e_k \cdot x_j + e_k$ (namely, $f_k(x) = x_j + 1$ where f_k is the k-th component of f). We call $\mathrm{Sol}(f) := x_j + e_k$ the solution polynomial associated to the ASP f. We require a solution polynomial to be excluded from the ASP's of type 1.
(ii) A polynomial $g(x) \in (\mathbb{F}_2)^m(x)$ is called an ASP of type 2 if there exist j, k such that $e_t \cdot g = e_t \cdot x_j$ for any t except k (namely, $g_t(x) = x_j$ for any t except k). We call $\mathrm{Sol}(g) := x_j + e_k$ the solution polynomial associated to g. We require a solution polynomial to be excluded from the ASP's of type 2.

Suppose f is an ASP of type 1 with its solution polynomial $\mathrm{Sol}(f) = x_j + e_k$. Then $f = 0$ implies that the k-th component of the variable x_j is 1. Thus x_j must be equal to e_k since we are computing $\mathbb{VS}(I)$. But note that $f = 0$ is not equivalent to $x_j = e_k$. A similar reasoning holds for ASP of type 2.

We prepare some notations for the Inoue algorithm.

Notation 16
Let $I \subset (\mathbb{F}_2)^m(x)$ be an ideal.
(i) $\mathrm{CONST}(I)$:= *the set of non-zero constants contained in* I.
(ii) $\mathrm{SP}(I)$:= *the set of solution polynomials contained in* I. *For a variable* x_j, *if a solution polynomial* $f = x_j + e_k$ *is contained in* $\mathrm{SP}(I)$, *then we say the variable* x_j *is determined (with the value* e_k*).*
(iii) $\mathrm{ASP}(I)$:= *the set of ASP's (of type 1 or 2) in* I.
(iv) $\mathrm{Sol}(\mathrm{ASP}(I)) := \{\mathrm{Sol}(f) \mid f \in \mathrm{AS}(I)\}$ = *the set of solution polynomials associated to ASP's contained in* I.

The following algorithm ASPTransform is the main part of the Inoue algorithm.

Algorithm 17 (ASPTransform)
Let I *be an ideal (Input).*
(i) If $\mathrm{CONST}(I) \neq \phi$, *then the output ASPTransform(I) is* I.
(ii) If $\mathrm{CONST}(I) = \phi$, *then redefine* $I := I + \langle \mathrm{Sol}(\mathrm{ASP}(I)) \rangle$. *Namely, add to* I *all the solution polynomials associated to the ASP's in* I. *Then go to (i) again.*
(iii) Repeat this process until $\mathrm{CONST}(I) \neq \phi$ *or* $\mathrm{ASP}(I) = \phi$, *and the output ASPTransform(I) is* I.

Now we can state the Inoue algorithm.

Algorithm 18 (The Inoue algorithm)
Fix a linear order on the set of variables $\{x_1, \ldots, x_n\}$ *(not a monomial order). Let* $I \subset (\mathbb{F}_2)^m(x)$ *be an ideal (input). Set* $L := \{\}$ *(empty set). We will put a singleton set solution in* L *in order.*
(i) If $\mathrm{CONST}(I) \neq \phi$, *then set* $L := L \cup \{\}$.
(ii) If $\mathrm{CONST}(I) = \phi$, *then redefine* $I :=$ *ASPTransform(I).*
(iii) If $\mathrm{CONST}(I) \neq \phi$, *then* $L := L \cup \{\}$. *If* $\mathrm{CONST}(I) = \phi$ *, we have 2 cases. (a) If* $\mathrm{SP}(I)$ *consists of* n *solution polynomials (namely* $I = \langle x_j + e_{j_k} \mid j = 1, \ldots, n \rangle$ *) so that all the variables are determined, then* $L := L \cup \{\mathrm{SP}(I)\}$ *(this is a solution). (b) Else let* x_j *be the least variable among the undetermined ones and* $\{e_{k_1}, \ldots, e_{k_p}\}$ *the possible values of* x_j. *Here* e_{k_l} *is a possible value of* x_j *if and only if* $\mathrm{CONST}(I + \langle x_j + e_{k_l} \rangle) = \phi$. *For each* l $(1 \leq l \leq p)$, *redefine* $I := I + \langle x_j + e_{k_l} \rangle$ *and go to (ii).*
(iv) The final output $\mathrm{Inoue}(I) = L$.

The following theorem makes it possible to rephrase the Inoue algorithm in terms of SBG bases instead of ideals.

Theorem 19
Let I *be an ideal in* $(\mathbb{F}_2)^m(x)$ *and* G *its SBG basis for a given monomial order. In the assertions (ii), (iii) below, we assume* I *does not contain non-zero constants.*
(i) For a non-zero constant $c \in (\mathbb{F}_2)^m$, $c \in I$ *if and only if* $c \in G$.
(ii) For a solution polynomial f, $f \in I$ *if and only if* $f \in G$.
(iii) If an ASP g *is in* I, *then there exists an ASP* $g' \in G$ *such that* $\mathrm{Sol}(g) = \mathrm{Sol}(g')$.

The assertion (iii) of Theorem 19 above is the main result (Theorem 31) of [3]. By Theorem 19, we can rephrase the Inoue Algorithm in terms of SBG bases instead of ideals. Namely, just replace the ideal I by its SBG basis G in Algorithms 17 and 18.

For the actual implementation of this algorithm, we also need the explicit classification of ASP's contained in an SBG basis (see [5, Corollary 3.9]). Inoue has implemented his algorithm on the computer algebra system Risa/Asir, whereas we have implemented it on the computer algebra system Magma [1].

We next define the Inoue invariant of an ideal $I \subset (\mathbb{F}_2)^m(x)$. The performance of the Inoue algorithm is well described by a tree diagram defined as below.

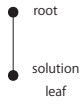
Definition 20 (Inoue Invariant)
Let I be an ideal and perform the Inoue algorithm starting from I. We will construct a tree Tree(I) of I as follows:
(i) I is the first node (root).
(ii) When the algorithm ASPTransform stops, we have a second node.
(iii) There are three cases. (a) If at this node $\mathrm{CONST}(I) \neq \phi$*, we have reached a terminal node (non-solution leaf). (b) If* $\mathrm{CONST}(I) = \phi$ *and all the n variables are determined, then we have reached a terminal node (a solution leaf). (c) If* $\mathrm{CONST}(I) = \phi$ *and there are undetermined variables, then select the least undetermined variable x_j. If there are p possible values $\{e_{k_1}, \ldots, e_{k_p}\}$ for x_j, then this tree branches in p directions at this node.*
(iv) Repeat this process until all the branches reach a (solution or non-solution) leaf.

By the process above, we get a tree Tree(I). We set $I_1 := \sharp\{\text{nodes}\}$*,* $I_2 := \sharp\{\text{leaves}\}$ *and* $I_3 :=$ *the depth of Tree(I) and call the triple* $\mathrm{Ino}(I) := (I_1, I_2, I_3)$ *the Inoue invariant of the ideal I.*

For the comparison of two Inoue invariants, we use lex order temporarily. The Inoue invariant measures the complexity of computation of the singleton set solutions $\mathbb{VS}(I)$ by the Inoue algorithm and is a very subtle invariant of I.

Example 21
Suppose $\sharp(\mathbb{VS}(I)) = 1$*. In case the Inoue algorithm calls ASPTransform only once and we reach the unique solution at once, then Tree(I) is the simplest tree with 2 nodes, 1 leaf and depth 1 (see Figure 1 below). In this case, we say I has a trivial Inoue invariant* $(2, 1, 1)$*.*

Fig. 1: The simplest tree with $\mathrm{Ino}(I) = (2, 1, 1)$



4 Formulation of puzzles of Sudoku type by a system of Boolean polynomial equations

In this section, we formulate the rules of the puzzles of Sudoku type in terms of Boolean polynomial equations after [9, 10].

A Sudoku puzzle is a partially-filled 9×9 square board with the integers $1, 2, \ldots, 9$, which should be completed in such a way that every row, column and the designated 3×3 block (see Table 1 below) is filled with no repeated entries.

We study the simpler versions of Sudoku, namely 4-doku, diagonal 5-doku and diagonal 6-doku puzzles. *A 4-doku puzzle* is a partially-filled 4×4 square board with integers $1, 2, 3, 4$. Every row, column and 2×2 block of the board should be filled with no repeated entries.

A diagonal 5-doku puzzle is a partially filled 5×5 table, where each row, column and diagonal (there are two diagonals) should be filed with numbers $1, \ldots, 5$ (no repeated entries). Since there are no blocks in 5-doku, it is natural to impose the diagonal conditions.

A diagonal 6-doku puzzle is a partially filled 6×6 table, where each row, column, 2×3 block and diagonal should be filled with numbers $1, \ldots, 6$ (no repeated entries). Note that there are six 2×3 rectangular (not square) blocks (see Table 2 below).

Table 1: An example of Sudoku puzzles

A Sudoku puzzle (clues)

					3		9	
2				5				
	6							
				2				
			7					
	9	3					8	
		8	9		1			
6						5		2
							4	

$\Longrightarrow$

The solution board

1	8	5	2	7	3	6	9	4
2	3	4	6	5	9	1	7	8
9	6	7	1	8	4	3	2	5
4	1	6	3	2	8	9	5	7
8	5	2	7	9	6	4	3	1
7	9	3	4	1	5	2	8	6
5	2	8	9	4	1	7	6	3
6	4	9	8	3	7	5	1	2
3	7	1	5	6	2	8	4	9

Table 2: An example of a diagonal 6-doku puzzle

A puzzle (clues)

					1
5					2
		6		3	

$\Longrightarrow$

The solution board

1	2	3	4	5	6
4	6	5	2	1	3
3	1	2	5	6	4
6	5	4	3	2	1
5	3	1	6	4	2
2	4	6	1	3	5

Since the formulation of the rules of these puzzles by a system of Boolean polynomial equations are similar, we take 4-doku puzzles for simplicity and formulate their rules.

Table 3: Assignment of 16 variables

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

We first assign 16 variables $a_{11}, a_{12}, \ldots, a_{44}$ as in Table 3. We then consider the Boolean polynomial ring $(\mathbb{F}_2)^4(a_{11}, a_{12} \ldots, a_{44})$ with lex order $a_{11} < a_{12} < \cdots < a_{44}$. We abbreviate as $0 = (0,0,0,0), 1 = (1,1,1,1)$ and set $e_1 := (1,0,0,0), e_2 := (0,1,0,0)$ etc.. Let us take the first row. Then the 7 equations below express the rules of 4-doku for the first row:

$$a_{11} + a_{12} + a_{13} + a_{14} + 1 = 0 \tag{1}$$

$$a_{11} \cdot a_{12} = 0, a_{11} \cdot a_{13} = 0, a_{11} \cdot a_{14} = 0, a_{12} \cdot a_{13} = 0, a_{12} \cdot a_{14} = 0, a_{13} \cdot a_{14} = 0 \tag{2}$$

For example, $(a_{11}, a_{12}, a_{13}, a_{14}) = (e_1, e_2, e_3, e_4)$ satisfies these equations. We note that there are lots of solutions in $(\mathbb{F}_2)^4$ other than this. For example, $(a_{11}, a_{12}, a_{13}, a_{14}) = (0, e_1 + e_2, e_3, e_4)$ is also a solution, which of course is not admissible as a solution for 4-doku puzzles.

There are 4 rows, 4 columns and 4 blocks so that there are $7 \times 12 = 84$ equations (or generators of an ideal) in all. Adding the clues (initial values) to the above generators, we can represent the rules of 4-doku puzzles by Boolean polynomials. We call the ideal generated by the above 84 polynomials together with the clues *the ideal of the given 4-doku puzzle*.

Let S be a puzzle of Sudoku type and I its ideal. In [5], we have defined the mathematical difficulty of S as $\mathrm{Ino}(I)$, which is supported by experimental data.

5 Main results

In this section, we state our main results and give the outlines of their proofs. For more details, see Appendix [6]. We first need a definition.

Definition 22 (Redundant and irredundant puzzles)
Let S be a puzzle of Sudoku type with a unique solution. If the deletion of any one number from S yields a puzzle which has more than one solution, we say S is an irredundant puzzle. A puzzle is called redundant if S is not irredundant.

The following proposition lessens the amount of computation very much.

Proposition 23
Let S be a redundant puzzle of Sudoku type with a unique solution, and S' a puzzle with several numbers deleted from S. We assume S' still has a unique solution. If the Inoue invariant of S' is trivial, then that of S is trivial too.

Proof Let I (resp. I') be the ideal of the puzzle S (resp. S'). Suppose we are applying the Inoue algorithm to I'. Since the Inoue invariant of I' is trivial, we reach the unique solution by applying ASPTransform once. Since we have $I \supset I'$, it holds that $\mathrm{ASP}(I) \supset \mathrm{ASP}(I'), \mathrm{SP}(I) \supset \mathrm{SP}(I'), \mathrm{Sol}(\mathrm{ASP}(I)) \supset \mathrm{Sol}(\mathrm{ASP}(I'))$.

Thus we have the following diagram:

$$\begin{array}{ccc} I' = I'_0 & \subset & I = I_0 \\ \cap & & \cap \\ I'_1 & \subset & I_1 \\ \cap & & \cap \\ & \cdots & \\ \cap & & \cap \\ J' = I'_k & \subset & I_k \end{array}$$

Here I'_j (resp. I_j) is the ideal obtained by adding to I'_{j-1} (resp. I_{j-1}) all the solution polynomials associated to the ASP's contained in I'_{j-1} (resp. I_{j-1}). Further $J' = \mathrm{ASPTransform}(I')$ is the ideal which is a solution leaf (namely, J' contains the solution polynomials of all the variables and does not contain non-zero constants).

We show that I_k also is the solution leaf. Indeed, since $I'_k \subset I_k$, I_k also contains the solution polynomials of all the variables. Further, I_k does not contain non-zero constants. Indeed, if I_k contains a non-zero constant, then $\mathbb{V}(I_k) = \phi$ and I (namely the puzzle S) does not have a solution, a contradiction to the assumption. Thus I_k satisfies the two conditions of the solution leaf. Therefore, starting from I, we reach a solution leaf I_k by one ASPTransform without branching, which means that the Inoue invariant of S is trivial. ∎

The following is the first main result of this note.

Theorem 24 (Inoue invariants of 4-doku)
Any 4-doku puzzle with a unique solution has the trivial Inoue invariant $(2, 1, 1)$.

Proof In the case of 4-doku, the irredundant puzzles (with a unique solution) exist only if the number of clues is 4,5,6 ([4]). Hence, by Proposition 23, it is enough to see that the Inoue invariant of the puzzles (with a unique solution) with 4,5,6 clues is trivial.

Further, there exist only 2 essentially different (namely different modulo the action of the 4-doku symmetry group) solution boards as shown in Table 4 ([4]):

Table 4: The essentially different 2 solution boards

No.1

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

No.2

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

Now, let S be a 4-doku puzzle with k ($k = 4, 5, 6$) clues with a unique solution T. By a suitable 4-doku symmetry transformation, we may assume T is No.1 or No.2 above. It is immediate to check that all the puzzles with a unique solution obtained by deleting l ($l = 10, 11, 12$) cells from these 2 solution boards have the trivial Inoue invariant by a computation with Magma (see Appendix [6, Section 1]). Thus our theorem is proved. ∎

We now turn to the diagonal 5-doku puzzles. We first enumerate the essentially different solution boards of diagonal 5-doku.

Let S_5 be the symmetric group of degree 5 and D_4 the dihedral group of order 8. D_4 is the symmetry group of the square with center at the origin. We note that S_5 acts on the set $\mathcal{X}$ of the solution boards of the diagonal 5-doku as the permutation of numbers. D_4 also acts naturally on $\mathcal{X}$ and actually, $S_5 \times D_4$ is the symmetry group of the diagonal 5-doku. The following theorem classifies the set $\mathcal{X}$ of solution boards modulo the action of $S_5 \times D_4$.

Theorem 25 (Essentially different solution boards of diagonal 5-doku)
There are only three different solution boards modulo $S_5 \times D_4$-action as shown below.

NSB No.1

1	2	3	4	5
2	4	5	3	1
5	3	2	1	4
3	1	4	5	2
4	5	1	2	3

NSB No.3

1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

NSB No.6

1	2	3	4	5
4	5	2	3	1
5	3	4	1	2
3	1	5	2	4
2	4	1	5	3

For the proof of Theorem 25, we first consider only S_5-action, forgetting D_4-action.

Definition 26 (Normalized solution boards of diagonal 5-doku)
A normalized solution board (NSB for short) is the one such that the first row is given by $a_{11} = 1, a_{12} = 2, a_{13} = 3, a_{14} = 4, a_{15} = 5$.

We note that each equivalence class of solution boards under the S_5-action contains a unique normalized solution board.

Proposition 27 (8 NSB's of diagonal 5-doku)
There are exactly 8 normalized solution boards as shown below.

NSB No.1

1	2	3	4	5
2	4	5	3	1
5	3	2	1	4
3	1	4	5	2
4	5	1	2	3

NSB No.2

1	2	3	4	5
2	5	4	1	3
4	3	2	5	1
5	4	1	3	2
3	1	5	2	4

NSB No.3

1	2	3	4	5
3	4	5	1	2
5	1	2	3	4
2	3	4	5	1
4	5	1	2	3

NSB No.4

1	2	3	4	5
3	5	2	1	4
5	1	4	3	2
4	3	5	2	1
2	4	1	5	3

NSB No.5

1	2	3	4	5
4	5	1	2	3
2	3	4	5	1
5	1	2	3	4
3	4	5	1	2

NSB No.6

1	2	3	4	5
4	5	2	3	1
5	3	4	1	2
3	1	5	2	4
2	4	1	5	3

NSB No.7

1	2	3	4	5
5	3	1	2	4
2	5	4	3	1
4	1	2	5	3
3	4	5	1	2

NSB No.8

1	2	3	4	5
5	3	4	1	2
4	5	2	3	1
2	4	1	5	3
3	1	5	2	4

Proof Let I be the ideal of the puzzle with the initial condition $a_{11} = 1, a_{12} = 2, a_{13} = 3, a_{14} = 4, a_{15} = 5$. Then we get the desired 8 solution boards by applying the Inoue solver (algorithm) to I (see Appendix [6, Subsection 2.1]). ∎

Proof of Theorem 25. If we transform the 8 NSB's with the D_4-action and then normalize, it is easy to check that there are 3 orbits: $\{\text{No.}1, 2, 4, 7\}, \{\text{No.}3, 5\}, \{\text{No.}6, 8\}$. Thus NSB No.1,3,6 are the complete representatives of the solution boards under $S_5 \times D_4$ action. ∎

We next enumerate the diagonal 5-doku puzzles with a unique solution.

Theorem 28 (Number of the diagonal 5-doku puzzles with a unique solution)
(i) There exist exactly 30964554720 diagonal 5-doku puzzles with a unique solution.
(ii) The irredundant puzzles exist only if the number of clues is 4,5,6, and the number of them is 7639680 in all.

Proof (i) Let S be a diagonal 5-doku puzzle with a unique solution T. By a suitable transformation, we may assume T is one of the three NSB's in Theorem 25. Now, by a time-consuming computation by Magma (see Appendix [6, Subsection 2.2]), we find that the number of puzzles which has the NSB No.1 (resp. No.3, No.6) as the unique solution is 32258030 (resp. 32246636, 32256282). Thus there are

$$(32258030 \times 4 + 32246636 \times 2 + 32256282 \times 2) \times 5! = 30964554720$$

puzzles with a unique solution in all. Note that we do not take the $S_5 \times D_4$-action into account for this enumeration.
(ii) By a similar computation as in (i), we can check that there are no irredundant puzzles which have NSB No.1, No.3, No.6 as a unique solution with k clues ($k \geq 7$). Also there are exactly 7996 (resp. 7920, 7920) irredundant puzzles which has No.1 (resp. No.3, No.6) as a unique solution (see Appendix [6, Subsection 2.2]). Thus there are

$$(7996 \times 4 + 7920 \times 2 + 7920 \times 2) \times 5! = 7639680$$

irredundant puzzles in all. ∎

We have a following impressive corollary.

Corollary 29 (The number of minimal and maximal clues of diagonal 5-doku)
(i) The minimal number of clues of the diagonal 5-doku puzzles with a unique solution is 4.
(ii) The maximal number of the clues of the irredundant ones is 6.

Remark 30
(i) Theorem 28 is significant in itself since this kind of precise enumeration seems to be known only for 4-doku so far ([4]).
(ii) To accomplish the computation for Theorem 28, we spent about a month using several Windows PC's simultaneously.

The following theorem is our second main result.

Theorem 31 (Inoue invariants of diagonal 5-doku)
All the diagonal 5-doku puzzles with a unique solution have the trivial Inoue invariant except the following two puzzles W_1, W_2 *(modulo* $S_5 \times D_4$*-action). These two have the Inoue invariant* $(4, 2, 2)$.

Table 5: The diagonal 5-doku puzzles with the non-trivial Inoue invariant $(4, 2, 2)$

W1

1			4	5
3	1		5	

W2

1				
	4			
3	1			
4	5			

Proof By Theorem 28 (ii), the irredundant diagonal 5-doku puzzles exist only if the number of clues is $4, 5, 6$.

Now, let S be a puzzle with k ($k = 4, 5, 6$) clues with a unique solution T. By a suitable 5-doku symmetry transformation, we may assume T is one of the 3 NSB's in Theorem 25. We can check that all the puzzles with a unique solution obtained by deleting l ($l = 19, 20, 21$) cells from these 3 solution boards have the trivial Inoue invariant except W_1 and W_2 by a computation with Magma (see Appendix [6, Subsection 2.3]).

Furthermore, if we add any number contained in the solution board to W_1 and W_2, it is easy to check that the resulting puzzles all have the trivial Inoue invariant. Thus by Proposition 23, we are done. ∎

Remark 32
We note that W_i *(*$i = 1, 2$*) has 6 clues, whereas the minimal number of clues of the puzzles with a unique solution is 4. Since it is natural to expect that the fewer the clues, the more difficult the puzzles are, this is an interesting phenomenon.*

We finally report a partial result on the Inoue invariants of the diagonal 6-doku puzzles, whose proof we omit since it is similar to that of Theorem 31.

Theorem 33 (Inoue invariants of diagonal 6-doku restricted to the 5-clues case)
(i) The minimal number of clues of the diagonal 6-doku puzzles with a unique solution is 5.
(ii) There exist exactly 44542080 puzzles with 5 clues which have a unique solution. Among them, there exist 10540800 puzzles with a non-trivial Inoue invariant. The biggest Inoue invariant of them is $(20, 12, 5)$*, and the puzzle with this Inoue invariant* $(20, 12, 5)$ *is the following one in Table 6.*

Table 6: The diagonal 6-doku puzzle with 5 clues with the biggest Inoue invariant $(20, 12, 5)$

				5	
					4
3					
		1	6		

Remark 34
(i) As Theorem 33 shows, there are many diagonal 6-doku puzzles with a non-trivial Inoue invariant unlike 4-doku and diagonal 5-doku puzzles, even restricted to the case of minimal 5 clues.

We also note that, as seen from the case of diagonal 5-doku puzzles, the number of clues of the puzzle with the biggest Inoue invariant may be larger than the minimal number of clues (Remark 32). Hence it may well happen that the diagonal 6-doku puzzle with the biggest Inoue invariant (unknown so far) has p clues where $p > 5$.
(ii) The reason that Theorem 33 refers only to the case of 5 clues is that it takes too much time for this computation. We estimate that it will take several years (maybe more) to achieve the same computation for all the number of clues. Thus, in the case of diagonal 6-doku, we have not obtained a complete result as in the case of diagonal 5-doku.

References

[1] Bosma, W., Cannon, J., Fieker, C. and Steel, A. (eds.), *Handbook of Magma functions*, Edition 2.18, http://magma.maths.usyd.edu.au/magma/ (2011).

[2] Cox, D., Little, J. and O'Shea, D., *Ideals, Varieties, and Algorithms*, third ed., Springer (2007).

[3] Inoue, S., Efficient Singleton Set Constraint Solving by Boolean Gröbner Bases, *Communications of JSSAC* **1**(2012), 27-37.

[4] Minami, S., Harikae, S. and Nakano, T., Enumeration of the 4-doku puzzles with a unique solution (in Japanese), *Bulletin of JSSAC* **18**(2012), No.2, 21-24.

[5] Nakano, T., Minami, S., Harikae, S., Arai, K. and Watanabe, H., On the Inoue invariant of a system of Boolean polynomial equations and its applications to puzzles of Sudoku type, *preprint* (2012).

[6] Nakano, T., Arai, K. and Watanabe, H., Appendix to the note "On the Inoue invariants of puzzles of Sudoku type", http://math.ru.dendai.ac.jp/~nakano/research.html (2013).

[7] Rosenhouse, J. and Taalman, L., *Taking Sudoku Seriously*, Oxford University Press (2011).

[8] Sato, Y., A New Type of Canonical Gröbner Bases in Polynomial Rings over Von Neumann Regular Rings, in: *Proceedings of ISSAC*(1998), ACM press, 317-321.

[9] Sato, Y., Inoue, S., Suzuki, A. and Nabeshima, K., Boolean Gröbner Bases and Sudoku, *preprint* (2008).

[10] Sato, Y., Inoue, S., Suzuki, A., Nabeshima, K. and Sakai, K., Boolean Gröbner bases. *J. of Symbolic Computation* **46**(2011), 622-632.

[11] Weispfenning, V., Gröbner Bases in Polynomial Ideals over Commutative Regular Rings, in: Davenport, E.(ed.), *EUROCAL'87*, Springer LNCS **378**(1989), 336-347.

Communications of JSSAC (2016)
Vol. 2, pp. 15 – 32

Approximate Polynomial GCD over Integers with Digits-wise Lattice

Kosaku Nagasaka*

Kobe University

(Received 29/Aug/2014 Accepted 3/Mar/2015)

Abstract

For the given coprime polynomials over integers, we change their coefficients slightly over integers so that they have a greatest common divisor (GCD) over integers. That is an approximate polynomial GCD over integers. There are only two algorithms known for this problem. One is based on an algorithm for approximate integer GCDs. The other is based on the well-known subresultant mapping and the lattice basis reduction. In this paper, we give an improved algorithm of the latter with a new lattice construction process by which we can restrict the range of perturbations. This helps us for computing approximate polynomial GCD over integers of the input erroneous polynomials having a priori errors on some digits of their coefficients.

Key words: Approximate Polynomial GCD, Lattice Basis Reduction

1 Introduction

Symbolic numeric algorithms for polynomials are very important, especially for practical computations since we have to operate with empirical polynomials having numerical errors on their coefficients. Recently, for those erroneous polynomials, many algorithms have been introduced, approximate univariate GCD and approximate multivariate factorization for example. However, for polynomials over integers having erroneous coefficients (e.g. rounded from empirical data), changing their coefficients over reals does not remain them in the polynomial ring over integers, hence we need algorithms designed over integers. In this paper, we discuss about computing a polynomial GCD of univariate or multivariate polynomials over integers approximately. Here, "approximately" means that we compute a polynomial GCD over integers by changing their coefficients slightly over integers so that the input polynomials still remain over integers. We improve one of known algorithms for computing an approximate polynomial GCD over integers defined below.

Definition 1 (Approximate Polynomial GCD Over Integers)
Let $f(\vec{x})$ and $g(\vec{x})$ be polynomials in variables $\vec{x} = x_1, \ldots, x_\ell$ over $\mathbb{Z}$, and let ε be a small positive integer. If they satisfy $f(\vec{x}) = t(\vec{x})h(\vec{x}) + \Delta_f(\vec{x})$, $g(\vec{x}) = s(\vec{x})h(\vec{x}) + \Delta_g(\vec{x})$ and $\varepsilon = \max\{\|\Delta_f\|, \|\Delta_g\|\}$

*nagasaka@main.h.kobe-u.ac.jp

for some polynomials $\Delta_f, \Delta_g \in \mathbb{Z}[\vec{x}]$, then we say that the above polynomial $h(\vec{x})$ is an ***approximate GCD over integers****. We also say that $t(\vec{x})$ and $s(\vec{x})$ are* ***approximate cofactors over integers****, and we say that their* ***tolerance*** *is ε. ($||p||$ denotes a suitable norm of $p(\vec{x})$.)* ◁

Example 2
Let $f(x_1, x_2)$ and $g(x_1, x_2)$ be the following polynomials over integers, which are relatively prime and supposed to have numerical errors on their coefficients.

$$\begin{aligned} f(x_1,x_2) &= 1530x_1^2x_2^2 - 3601x_1^2x_2 + 2109x_1^2 - 171x_1x_2^2 \\ &\qquad + 3506x_1x_2 - 3703x_1 - 699x_2^2 + 94x_2 + 1561, \\ g(x_1,x_2) &= 2755x_1^2x_2^2 - 5851x_1^2x_2 + 3110x_1^2 - 5118x_1x_2^2 \\ &\qquad + 5296x_1x_2 + 351x_1 + 2275x_2^2 - 1098x_2 - 3822. \end{aligned}$$

We would find the following approximate GCD over integers, where the underlined figures are slightly changed to make them having a non-trivial polynomial GCD.

$$\begin{aligned} f(x_1,x_2) &\approx (34x_1x_2 - 37x_1 - 25x_2 + 39) \times (45x_1x_2 - 57x_1 + 28x_2 + 40) \\ &= 1530x_1^2x_2^2 - 360\underline{3}x_1^2x_2 + 2109x_1^2 - 17\underline{3}x_1x_2^2 \\ &\qquad + 350\underline{4}x_1x_2 - 3703x_1 - \underline{700}x_2^2 + 9\underline{2}x_2 + 156\underline{0}, \\ g(x_1,x_2) &\approx (34x_1x_2 - 37x_1 - 25x_2 + 39) \times (81x_1x_2 - 84x_1 - 91x_2 - 98) \\ &= 275\underline{4}x_1^2x_2^2 - 585\underline{3}x_1^2x_2 + 31\underline{08}x_1^2 - 511\underline{9}x_1x_2^2 \\ &\qquad + 529\underline{4}x_1x_2 + 35\underline{0}x_1 + 2275x_2^2 - 109\underline{9}x_2 - 3822. \end{aligned}$$

In this case, $\Delta_f = 2x_1^2x_2 + 2x_1x_2^2 + 2x_1x_2 + x_2^2 + 2x_2 + 1$, $\Delta_g = x_1^2x_2^2 + 2x_1^2x_2 + 2x_1^2 + x_1x_2^2 + 2x_1x_2 + x_1 + x_2$ and $\varepsilon = 2$ in the ∞-norm. ◁

We note that for polynomials over the complex numbers, there are many studies and various algorithms ([12, 6, 4, 15, 31, 30, 5, 32, 23, 34, 33, 25, 13, 22, 9, 24, 8, 16, 21, 26, 27, 20, 2, 3, 7]). Hence one may think that we can compute an approximate GCD over integers by rounding the result by those algorithms since they compute approximate GCDs over complex numbers. However, it is difficult to make them as polynomials over integers since the resulting tolerance easily becomes large and far from the given polynomials (see [18]). Therefore, we need algorithms designed for polynomials over integers.

For computing approximate GCD over integers, there are two known algorithms. One is based on the result from approximate integer common divisors by Howgrave-Graham ([11]). The other is based on the well-known subresultant mapping and the lattice basis reduction (the LLL algorithm [14]). The former algorithm is originally proposed by von zur Gathen and Shparlinski ([29]) at LATIN 2008 and revised by von zur Gathen et al ([28]). Their algorithm only works for very tiny tolerances and one of input polynomials $f(\vec{x})$ and $g(\vec{x})$ must be given exactly and can not be perturbed. However, the algorithm always can compute an approximate GCD over integers if the given polynomials satisfy the certain conditions. The latter algorithm is proposed by the present author ([17]) at ISSAC 2008 and revised ([18]). In contrast with that by von zur Gathen et al., this algorithm works for not only very tiny but also small tolerances and all the given polynomials can be perturbed (as described in the definition). However, any theoretical condition which guarantees that the algorithm can compute an approximate GCD over integers, is not given.

1.1 The problem to be solved

In this paper, we give an improved algorithm with a new lattice construction process by which we can restrict the range of perturbations in some cases. This helps us for computing approximate

polynomial GCD over integers of the input erroneous polynomials having a priori errors on some digits of their coefficients. For example, the known methods can not compute any approximate polynomial GCD over integers for the following polynomials.

$$\begin{aligned} f(x) &= -302260x^4 - 174933528x^3 + 45943440x^2 + 231047900996x - 143756712 \\ &\approx \left(889x^2 + 512701x - 319\right)\left(-340x^2 - 692x + 450648\right) - 2 \times 10^3 x^2, \\ g(x) &= 526407460x^4 + 303589900698x^3 - 690875197x^2 - 323202349x + 205289 \\ &\approx \left(889x^2 + 512701x - 319\right)\left(592140x^2 - 978x - 631\right) - 5 \times 10^3 x^4 + 4 \times 10^3. \end{aligned}$$

In this case, the tolerance (the absolute error) is 5×10^3 in the ∞-norm and the relative error is not small in relation to the smallest coefficients hence computing an approximate GCD over integers for this pair of polynomials is not so easy. In fact, the known algorithms ([17],[18]) can not detect any expected result.

One may think that this example seems to be odd. However, this situation possibly occurs in some computations with multi-precision integers (each integer is represented as an array of word size integers). For example, transmission errors on some elements of the array, computing lower and higher digits separately and so on. In fact, the above pair of polynomials has perturbations on the second digit only (as an array of 10^3 integers) hence they are in this case. Moreover, this is also useful for simplifying algebraic expressions (e.g. each simplicity of expression is heavily depending on the number of terms not the magnitude of coefficients in general) as in the following polynomial.

$$\begin{aligned} f(x_1, x_2) &= \left(286x_2^2 - 54821x_2 - 3907787\right) x_1^2 \\ &\quad + \left(203830x_2^2 + 11276643x_2 + 35293\right) x_1 - 17930x_2^2 - 990865x_2 + 54765 \\ &= (22x_2 + 1217)((13x_2 - 3211)x_1^2 + (9265x_2 + 29)x_1 - 815x_2 + 45) + 5 \times 10^2 x_2 x_1. \end{aligned}$$

For this problem, we review the algorithm given by the present author ([18]) in Section **2**. We give a new lattice construction process in Section **3**, including various numerical examples. In Section **4**, we give some remarks for this extension. We note that the present article is an extended work of the presentation ([19]) with the extended abstract at SNC 2011 (Symbolic-Numeric Computation, June 7-9, 2011, San Jose, California), and the ideal of this paper is based on the preliminary presentation about computing approximate GCD of integers (not polynomials) by the present author in Research Institute for Mathematical Sciences, Kyoto University in 2010.

2 Approximate GCD by Lattice Basis Reduction

We review the known result ([17],[18]) briefly. Let $f(\vec{x})$ and $g(\vec{x})$ have total degrees $n = \mathrm{tdeg}(f)$ and $m = \mathrm{tdeg}(g)$, respectively. We call the following mapping $\mathcal{S}_r(f, g)$ the subresultant mapping of $f(\vec{x})$ and $g(\vec{x})$ of order r.

$$\mathcal{S}_r(f,g): \begin{array}{ccc} \mathcal{P}_{m-r-1} \times \mathcal{P}_{n-r-1} & \to & \mathcal{P}_{n+m-r-1} \\ (s(\vec{x}), t(\vec{x})) & \mapsto & s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \end{array}$$

where $r = 0, \ldots, \min\{n, m\} - 1$ and $\mathcal{P}_d$ denotes the set of polynomials in variables $x_1, \ldots, x_\ell$, of total degree d or less. We denote the coefficient vector of polynomial $p(\vec{x})$ by $\mathrm{vect}(p)$ w.r.t. the lexicographic ascending order in this article. We note that any term order can be used for representing coefficient vectors since the order is not essential. To see the number of elements of a coefficient vector, we define the notation: $\beta_{d,r} = \binom{d-r+\ell}{\ell}$ hence the number of terms $x_1^{i_1} \cdots x_\ell^{i_\ell}$

satisfying $i_1 + \cdots + i_\ell \leq d$ can be denoted by $\beta_{d,0}$. The k-th convolution matrix $C_k(f)$ is defined to satisfy $C_k(f)\mathrm{vect}(p) = \mathrm{vect}(fp)$ for any polynomial $p(\vec{x})$ of total degree $k-1$ or less, where $\mathrm{vect}(p) \in \mathbb{Z}^{\beta_{k-1,0}\times 1}$ and $C_k(f) \in \mathbb{Z}^{\beta_{n+k-1,0}\times\beta_{k-1,0}}$. We have the matrix representation of the subresultant mapping: $Syl_r(f,g) = (C_{m-r}(f)\ C_{n-r}(g))$ of size $(\beta_{n+m-1,r}) \times (\beta_{m-1,r} + \beta_{n-1,r})$, satisfying

$$\mathcal{S}_r(f,g) : \begin{array}{ccc} \mathcal{P}_{m-r-1} \times \mathcal{P}_{n-r-1} & \to & \mathcal{P}_{n+m-r-1} \\ (\ \mathrm{vect}(s)^{\ t}\ \mathrm{vect}(t)^{\ t}\ )^t & \mapsto & \mathrm{vect}(sf+tg) = Syl_r(f,g)(\ \mathrm{vect}(s)^{\ t}\ \mathrm{vect}(t)^{\ t}\ )^t. \end{array}$$

This mapping is the same as in [10], and has the same property that $f(\vec{x})/t(\vec{x})$ and $g(\vec{x})/s(\vec{x})$ is the GCD of $f(\vec{x})$ and $g(\vec{x})$ if r is the greatest integer such that this mapping is not injective. Hence by computing null vectors of $Syl_r(f,g)$ approximately for the given coprime polynomials, we can compute candidate vectors of approximate cofactors over integers. This procedure can be done by finding short vectors by the well-known LLL algorithm ([14]). For this, we construct the lattice generated by the row vectors of $\mathcal{L}(f,g,r,c)$ which is defined as the following matrix where r denotes the order of the subresultant mapping.

$$\mathcal{L}(f,g,r,c) = (E_{\beta_{n-1,r}+\beta_{m-1,r}} \mid c \cdot Syl_r(f,g)^t)$$

where E_i denotes the identity matrix of size $i \times i$ and $c \in \mathbb{Z}$. The size of $\mathcal{L}(f,g,r,c)$ is $(\beta_{n-1,r} + \beta_{m-1,r}) \times (\beta_{n-1,r} + \beta_{m-1,r} + \beta_{n+m-1,r})$. We note that we mark a block matrix with a vertical bar to distinguish the identity matrix representing a collection of linear combinations from the matrix formed by the coefficient vectors.

However, the short vectors found are only candidate cofactors $t(\vec{x})$ and $s(\vec{x}) \in \mathbb{Z}[\vec{x}]$ such that $s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \approx 0$, and $f(\vec{x})$ and $g(\vec{x})$ may not be divisible by $h(\vec{x})$. To compute an approximate GCD from the candidate cofactors, we apply the LLL algorithm again to the lattice generated by the row vectors of the following matrix $\mathcal{H}(f,g,r,c,t,s)$ of size $(\beta_{r+1,0}+1)\times(\beta_{n,0}+\beta_{m,0}+\beta_{r+1,0}+1)$.

$$\mathcal{H}(f,g,r,c,t,s) = \left(E_{\beta_{r+1,0}+1} \;\middle|\; \begin{array}{cc} c\cdot\mathrm{vect}(f)^t & c\cdot\mathrm{vect}(g)^t \\ c\cdot C_{r+2}(-t)^t & c\cdot C_{r+2}(s)^t \end{array}\right).$$

We have the following lemmas in [18].

Lemma 3
Let B be a bound of maximum absolute value of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$. For the lattice generated by the rows of $\mathcal{L}(f,g,r,c_{\mathcal{L}})$ with $c_{\mathcal{L}} = 2^{(\beta_{n-1,r}+\beta_{m-1,r}-1)/2}\ \sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$, the LLL algorithm can find a short vector whose first $\beta_{n-1,r} + \beta_{m-1,r}$ elements are a multiple of the transpose of the coefficient vectors of cofactors of $f(\vec{x})$ and $g(\vec{x})$ by their GCD, if r is the greatest integer such that the subresultant mapping is not injective. ◁

Lemma 4
Let B be a bound of maximum absolute value of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$. For the lattice generated by the row vectors of $\mathcal{H}(f,g,r,c_{\mathcal{H}},t,s)$ with $c_{\mathcal{H}} = 2^{\beta_{r+1,0}/2}\ \sqrt{\beta_{r+1,0}+1}B + 1$, the LLL algorithm can find a short vector whose 2-nd, ..., $(\beta_{r+1,0}+1)$-th elements are a multiple of the transpose of the coefficient vector of the GCD of $f(\vec{x})$ and $g(\vec{x})$, if r is the greatest integer such that the subresultant mapping is not injective. ◁

For example, we consider the following pair of erroneous polynomials.

$$\begin{array}{rcccl} f(x) & = & 20x^2 + 18x - 27 & = & (4x+7)(5x-4) - x + 1, \\ g(x) & = & 29x^2 + 61x + 19 & = & (4x+7)(7x+3) + x^2 - 2. \end{array}$$

We construct the following matrix $\mathcal{L}(f, g, r, c)$ with $r = 0$ and $c = 1$, and apply the LLL algorithm to the lattice generated by the row vectors of $\mathcal{L}(f, g, r, c)$.

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 19 & 61 & 29 & 0 \\ 0 & 1 & 0 & 0 & 0 & 19 & 61 & 29 \\ 0 & 0 & 1 & 0 & -27 & 18 & 20 & 0 \\ 0 & 0 & 0 & 1 & 0 & -27 & 18 & 20 \end{array}\right) \rightarrow \left(\begin{array}{cccc|cccc} \underline{-4} & \underline{5} & \underline{-3} & \underline{-7} & 5 & -14 & 3 & 5 \\ -5 & 6 & -3 & -9 & -14 & -2 & -1 & -6 \\ -7 & 9 & -5 & -13 & 2 & 5 & 12 & 1 \\ -4 & 5 & -3 & -8 & 5 & 13 & -15 & -15 \end{array}\right).$$

We take the first row vector as candidate cofactors (we note that we have to seek the candidate through all the short vectors). We construct the following matrix $\mathcal{H}(f, g, r, c, t, s)$ with $c = 1$ and apply the LLL algorithm, to compute an approximate GCD.

$$\left(\begin{array}{ccc|cccccc} 1 & 0 & 0 & -27 & 18 & 20 & 19 & 61 & 29 \\ 0 & 1 & 0 & -4 & 5 & 0 & 3 & 7 & 0 \\ 0 & 0 & 1 & 0 & -4 & 5 & 0 & 3 & 7 \end{array}\right) \rightarrow \left(\begin{array}{ccc|cccccc} 1 & \underline{-7} & \underline{-4} & 1 & -1 & 0 & -2 & 0 & 1 \\ 0 & 1 & 0 & -4 & 5 & 0 & 3 & 7 & 0 \\ 0 & 0 & 1 & 0 & -4 & 5 & 0 & 3 & 7 \end{array}\right).$$

Hence, we get $4x + 7$ as an approximate polynomial GCD over integers and $5x - 4$ and $7x + 3$ as approximate cofactors. We note that there are more complicated examples, some lemmas and techniques for decreasing the computing-time (see [17],[18]) though we do not show them here.

3 Digits-wise Lattice

The algorithms introduced in [17] and [18] work well for nearby polynomials having polynomial GCD, according to the numerical experiments therein. However, they can not detect any approximate GCD for the following type of polynomials as noted in the introduction. We note again that this problem is not so special in practice (multi-precision integers, simplifying algebraic expressions and so on). It could be more general word sizes (e.g. 2^{32}) though the word size we use here is 10^1 since this is easy to understand and does not exceed the paper width.

$$\begin{aligned} f(x) &= 32x^3 + \underline{7}6x^2 + \underline{2}2x + 15 &&= (4x + 5)(8x^2 + 4x + 3) + 20x^2 - 10x, \\ g(x) &= \underline{1}0x^3 + 53x^2 + 59x + \underline{4}0 &&= (4x + 5)(5x^2 + 7x + 6) - 10x^3 + 10. \end{aligned}$$

To extend the algorithms for the above case (all the coefficients have a priori errors on only the limited number of digits), we introduce the following digits-wise lattice instead of $\mathcal{L}(f, g, r, c)$ by extending the coefficient vector to the digits-wise.

$$\mathcal{L}(f, g, r, c) = \left(\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 40 & 59 & 53 & 10 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 40 & 59 & 53 & 10 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 40 & 59 & 53 & 10 \\ 0 & 0 & 0 & 1 & 0 & 0 & 15 & 22 & 76 & 32 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 15 & 22 & 76 & 32 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 15 & 22 & 76 & 32 \end{array}\right) \Rightarrow \left(\begin{array}{cccccc|cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 5 & 9 & 5 & 3 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 5 & 9 & 5 & 3 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 5 & 9 & 5 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 5 & 2 & 2 & 7 & 6 & 3 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 5 & 2 & 2 & 7 & 6 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 5 & 2 & 2 & 7 & 6 \end{array}\right).$$

However, the row spaces of the above matrices are not the same and they are essentially different since digit-wise operations can not follow the carrying and borrowing operations for integers. For computing an approximate GCD we need to guarantee that the row space has the coefficient vectors corresponding to their cofactors, hence we have to perform some artificial carrying and borrowing operations in this row space. To do this, we add some extra row vectors representing carry and

borrow digits to the matrix as follows.

$$\left(\begin{array}{cccccc|cccccccccc}
1&0&0&0&0&0&4&0&5&9&5&3&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&4&0&5&9&5&3&1&0&0&0\\
0&0&1&0&0&0&0&0&0&0&4&0&5&9&5&3&1&0\\
0&0&0&1&0&0&1&5&2&2&7&6&3&2&0&0&0&0\\
0&0&0&0&1&0&0&0&1&5&2&2&7&6&3&2&0&0\\
0&0&0&0&0&1&0&0&0&0&1&5&2&2&7&6&3&2\\
\hline
0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10
\end{array}\right)\left.\begin{array}{c}\\\\\\\\\\\\\\\\\\\\\\\\\end{array}\right\}\text{ extra rows.}$$

Moreover, in this case, if we can assume that only the second digit has a priori error hence we multiple the columns except ones corresponding to the second digit by 100 as a penalty weight. The LLL algorithm gives the following result for the lattice generated by row vectors of this scaled matrix.

$$\left(\begin{array}{cccccc|cccccccccc}
1&-2&6&-2&1&-5&1&0&-5&0&-3&0&16&0&-5&0&-10&0\\
\underline{3}&\underline{4}&\underline{8}&\underline{-6}&\underline{-7}&\underline{-5}&3&0&10&0&3&0&-12&0&-14&0&-8&0\\
0&-10&0&10&0&0&15&0&-18&0&17&0&-21&0&-10&0&0&0\\
2&-4&2&-4&2&0&2&0&-10&0&-31&0&-5&0&13&0&2&0\\
6&-2&-4&-2&6&0&21&0&32&0&2&0&11&0&-4&0&-4&0\\
-1&2&-16&2&9&5&-1&0&20&0&-15&0&1&0&-16&0&0&0\\
2&4&2&-6&-3&0&-1&0&10&100&-10&0&-7&0&5&0&2&0\\
-3&-1&-4&6&1&0&-3&0&-7&0&10&0&-5&-100&-19&0&-4&0\\
1&-2&4&-2&-1&0&1&0&-8&0&-8&100&0&0&16&0&4&0\\
3&-5&-1&4&1&0&18&0&8&0&15&0&-9&0&-7&-100&-1&0\\
-3&-2&-4&6&3&1&-3&0&-8&0&10&0&7&0&-6&0&-1&200\\
2&-4&2&1&2&0&9&500&1&0&7&0&11&0&13&0&2&0
\end{array}\right).$$

We can see that the resulting matrix has the row vector corresponding to the coefficient vectors of expected approximate cofactors ($8x^2 + 4x + 3$, $5x^2 + 7x + 6$) on the second row underlined. In the following subsections, we formalize this process into definitions and an algorithm.

3.1 Definitions of Digits-wise Representation

We denote the canonical form of length w of the base b digits in the integer a as

$$\forall a \in \mathbb{Z},\ \mathrm{digits}_{b,w}(a) = \{a_{w-1}, \ldots, a_1, a_0\} \text{ such that}$$
$$a = \textstyle\sum_{i=0}^{w-1} a_i b^i \text{ and } \begin{cases} 0 \le \mathrm{sign}(a) a_i < b & (i = 0, \ldots, w-2) \\ \mathrm{sign}(a) = \mathrm{sign}(a_i) & (i = w-1). \end{cases}$$

For example, we have $\mathrm{digits}_{10,2}(123) = \{12, 3\}$, $\mathrm{digits}_{10,3}(123) = \{1, 2, 3\}$, $\mathrm{digits}_{10,4}(123) = \{0, 1, 2, 3\}$ and $\mathrm{digits}_{10,3}(-123) = \{-1, -2, -3\}$. We extend the coefficient vector of polynomial $p(\vec{x})$ to the digits-wise operations and denote it by $\mathrm{vect}_{b,w}(p)$ where b and w are the base number and the length of the list of digits, respectively, such that

$$\mathrm{vect}_{b,w}(p) = \{\mathrm{digits}_{b,w}(p_e)\ \ldots\ \mathrm{digits}_{b,w}(p_0)\}^t \text{ where } \mathrm{vect}(p) = \{p_e\ \ldots\ p_0\}^t.$$

For example, we have $\mathrm{vect}_{10,2}(32x^3 + 76x^2 + 22x + 15) = \{1, 5, 2, 2, 7, 6, 3, 2\}^t$. We note that the sizes of the coefficient vectors $\mathrm{vect}_{b,w}(f)$ and $\mathrm{vect}_{b,w}(g)$ of $f(\vec{x})$ and $g(\vec{x})$ in the digits-wise form are $w \times \beta_{n,0}$ and $w \times \beta_{m,0}$, respectively. Therefore, their inverse mappings $\mathrm{digits}_{b,w}^{-1}(\cdot)$ and $\mathrm{vect}_{b,w}^{-1}(\cdot)$ can be defined as follows.

$$\mathrm{digits}_{b,w}^{-1}(\vec{d}) = \sum_{i=0}^{w-1} a_i b^i, \quad \mathrm{vect}_{b,w}^{-1}(\vec{p}) = \mathrm{vect}^{-1}(\mathrm{digits}_{b,w}^{-1}(\vec{p}_{w\times\beta_{n,0}}), \ldots, \mathrm{digits}_{b,w}^{-1}(\vec{p}_0))$$

where $\vec{d} = \{a_{w-1}, \ldots, a_1, a_0\}^t \in \mathbb{Z}^w$ and $\vec{p} = \{\vec{p}_{w\times\beta_{n,0}}^t, \ldots, \vec{p}_0^t\}^t \in \mathbb{Z}^{w\times\beta_{n,0}}$, and $\mathrm{vect}^{-1}(\cdot)$ is the conventional mapping from the coefficient vector to the polynomial.

We also extend the k-th convolution matrix and the matrix representation of the subresultant mapping to the digits-wise operations in the same manner and denote them by $C_{k,b,w}(f)$ and $Syl_{r,b,w}(f, g)$, respectively. We note that in general they do not satisfy $C_{k,b,w}(f)\mathrm{vect}_{b,w}(p) = \mathrm{vect}_{b,w}(fp)$ for any polynomial $p(\vec{x})$ of total degree $k-1$, however this is not the matter in our approach. Moreover, we have $\mathrm{vect}_{b,1}(f) = \mathrm{vect}(f)$, $C_{k,b,1}(f) = C_k(f)$ and $Syl_{r,b,1}(f, g) = Syl_r(f, g)$.

For the digits-wise lattice introduced in the beginning of this section, the carrying and borrowing are important hence we define the following carry-borrow vectors $\vec{z}_{b,w,i}$ $(i = 0, 1, \ldots, w-2)$ and matrix $\mathcal{Z}_{b,w}$, satisfying $\mathrm{digits}_{b,w}^{-1}(\vec{z}_{b,w,i}) = 0$ $(i = 0, 1, \ldots, w-2)$.

$$\vec{z}_{b,w,i} = \{\underbrace{0, \ldots, 0}_{i}, -1, b, \underbrace{0, \ldots, 0}_{w-i-2}\}^t \in \mathbb{Z}^w, \ \mathcal{Z}_{b,w} = \{\vec{z}_{b,w,0} \ \ldots \ \vec{z}_{b,w,w-2}\}^t \in \mathbb{Z}^{(w-1)\times w}.$$

We also extend $\mathcal{L}(f, g, r, c)$ and $\mathcal{H}(f, g, r, c, t, s)$ as follows and denote them by $\mathcal{L}_{b,w}(f, g, r, c)$ and $\mathcal{H}_{b,w}(f, g, r, c, t, s)$, respectively.

$$\mathcal{L}_{b,w}(f, g, r, c) = \left(\begin{array}{c|ccc} E_{\beta_{n-1,r}+\beta_{m-1,r}} & c \cdot Syl_{r,b,w}(f, g)^t & & \\ \hline & c \cdot \mathcal{Z}_{b,w} & & \\ & & c \cdot \mathcal{Z}_{b,w} & \\ & & & \ddots \\ & & & c \cdot \mathcal{Z}_{b,w} \end{array}\right),$$

$$\mathcal{H}_{b,w}(f, g, r, c, t, s) = \left(\begin{array}{c|ccc} E_{\beta_{r+1,0}+1} & c \cdot \mathrm{vect}_{b,w}(f)^t & c \cdot \mathrm{vect}_{b,w}(g)^t & \\ & c \cdot C_{r+2,b,w}(-t)^t & c \cdot C_{r+2,b,w}(s)^t & \\ \hline & c \cdot \mathcal{Z}_{b,w} & & \\ & & c \cdot \mathcal{Z}_{b,w} & \\ & & & \ddots \\ & & & c \cdot \mathcal{Z}_{b,w} \end{array}\right).$$

The sizes of $\mathcal{L}_{b,w}(f, g, r, c)$ and $\mathcal{H}_{b,w}(f, g, r, c, t, s)$ are $((\beta_{n-1,r} + \beta_{m-1,r}) + (w-1)\beta_{n+m-1,r}) \times (\beta_{n-1,r} + \beta_{m-1,r} + w\beta_{n+m-1,r})$ and $(\beta_{r+1,0} + 1 + (w-1)(\beta_{n,0} + \beta_{m,0})) \times (\beta_{r+1,0} + 1 + w(\beta_{n,0} + \beta_{m,0}))$, respectively.

Example 5
We show some examples of $\mathcal{L}_{b,w}(f, g, r, c)$ and $\mathcal{H}_{b,w}(f, g, r, c, t, s)$ for

$$\begin{aligned} f(x) &= 32x^3 + 56x^2 + 32x + 15 = (4x+5)(8x^2+4x+3), \quad t(x) = -8x^2 - 4x - 3, \\ g(x) &= 20x^3 + 53x^2 + 59x + 30 = (4x+5)(5x^2+7x+6), \quad s(x) = 5x^2 + 7x + 6. \end{aligned}$$

We have the following matrices for the base number $b = 10$ *and length* $w = 2$ *if we assume that the order of subresultant mapping is* 0 *and* $c = 1$.

$$\mathcal{L}_{10,2}(f, g, 0, 1) = \left(\begin{array}{cccccc|cccccccccccc} 1&0&0&0&0&0&3&0&5&9&5&3&2&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&3&0&5&9&5&3&2&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&3&0&5&9&5&3&2&0\\ 0&0&0&1&0&0&1&5&3&2&5&6&3&2&0&0&0&0\\ 0&0&0&0&1&0&0&0&1&5&3&2&5&6&3&2&0&0\\ 0&0&0&0&0&1&0&0&0&0&1&5&3&2&5&6&3&2\\ \hline 0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10 \end{array}\right),$$

$$\mathcal{H}_{10,2}(f, g, 0, 1, t, s) = \left(\begin{array}{ccc|cccccccccccccccc} 1&0&0&1&5&3&2&5&6&3&2&3&0&5&9&5&3&2&0\\ 0&1&0&0&3&0&4&0&8&0&0&0&6&0&7&0&5&0&0\\ 0&0&1&0&0&0&3&0&4&0&8&0&0&0&6&0&7&0&5\\ \hline 0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-1&10 \end{array}\right).$$

◁

For any fixed non-negative integer n, $\mathrm{vect}_{b,w}(\cdot)$ and $\mathrm{vect}_{b,w}^{-1}(\cdot)$ can be thought as linear mappings over $\mathbb{Z}$ between $\mathcal{P}_n$ and $\mathbb{Z}^{w\times\beta_{n,0}}$ where $\mathcal{P}_n$ is a submodule of $\mathbb{Z}[\vec{x}]$ defined in the previous section. However, $\mathcal{P}_n$ and $\mathbb{Z}^{w\times\beta_{n,0}}$ are not isomorphic by these mappings. We define the quotient module of $\mathbb{Z}^{w\times\beta_{n,0}}$ by the equivalence relation "$\vec{f} \equiv \vec{g}$ iff $\mathrm{vect}_{b,w}^{-1}(\vec{f}) = \mathrm{vect}_{b,w}^{-1}(\vec{g})$" or its subspace generated by the row vectors of block diagonal matrix of $\{\mathcal{Z}_{b,w}, \ldots, \mathcal{Z}_{b,w}\}$, and we denote this quotient module by $\mathbb{Z}_{b,w}^{w\times\beta_{n,0}}$. By these definitions, $\mathcal{P}_n$ is isomorphic to $\mathbb{Z}_{b,w}^{w\times\beta_{n,0}}$ by $\mathrm{vect}_{b,w}(\cdot)$ and $\mathrm{vect}_{b,w}^{-1}(\cdot)$.

Lemma 6
Let B *be a bound of maximum absolute value of coefficients of any factors of* $f(\vec{x})$ *and* $g(\vec{x})$*. For the lattice generated by the row vectors of* $\mathcal{L}_{b,w}(f, g, r, c_{\mathcal{L}})$ *with* $c_{\mathcal{L}} = 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2}\sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$*, the LLL algorithm can find a short vector whose first* $\beta_{n-1,r}+\beta_{m-1,r}$ *elements are a multiple of the transpose of the coefficient vectors of cofactors of* $f(\vec{x})$ *and* $g(\vec{x})$ *by their GCD, if* r *is the greatest integer such that the subresultant mapping is not injective.* ◁

Proof There are cofactors $t(\vec{x})$ and $s(\vec{x})$ of $f(\vec{x})$ and $g(\vec{x})$ by their GCD, respectively, if r is the greatest integer such that the subresultant mapping is not injective. Hence, the lattice generated by row vectors of $\mathcal{L}_{b,w}(f, g, r, c_{\mathcal{L}})$ has the following vector $\vec{u}_{min}$ since $\mathbb{Z}_{b,w}^{w\times\beta_{n+m-1,r}}$ is isomorphic to $\mathcal{P}_{n+m-r-1}$ as shown above.

$$\vec{u}_{min} = (\text{the transpose of the coefficient vectors of } s(\vec{x}) \text{ and } t(\vec{x}),\ \underbrace{0\ \cdots\ 0}_{w\times\beta_{n+m-1,r}}).$$

The LLL algorithm can find a short vector $\vec{u}$ satisfying

$$||\vec{u}||_2 \leq 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2}\ ||\vec{u}_{min}||_2\ .$$

Since all the non-zero elements of right $w \times \beta_{n+m-1,r}$ columns of any row vectors in the lattice which is generated by the row vectors of $\mathcal{L}_{b,w}(f,g,r,c_{\mathcal{L}})$ must be larger than or equal to $c_{\mathcal{L}} = 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2}\ \sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$ in absolute value, the right $w \times \beta_{n+m-1,r}$ columns of the found short vector $\vec{u}$ must be zeros. This means that the transpose of the vector formed by the first $\beta_{n-1,r}+\beta_{m-1,r}$ elements of $\vec{u}$ is in the null space of $Syl_{r,b,w}(f,g)$ hence in that of $Syl_r(f,g)$ and the lemma is proved. ∎

Lemma 7
Let B be the maximum absolute value of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$. For the lattice generated by the row vectors of $\mathcal{H}_{b,w}(f,g,r,c_{\mathcal{H}},t,s)$ with $c_{\mathcal{H}} = 2^{(\beta_{r+1,0}+(w-1)(\beta_{n,0}+\beta_{m,0}))/2}\ \sqrt{\beta_{r+1,0}+1}B+1$, the LLL algorithm can find a short vector whose 2-nd, …, $(\beta_{r+1,0}+1)$-th elements are a multiple of the transpose of the coefficient vector of the GCD of $f(\vec{x})$ and $g(\vec{x})$, if r is the greatest integer such that the subresultant mapping is not injective. ◁

Proof The proof is similar to that of Lemma 6. ∎

We note that in Lemma 7 the short vectors corresponding to the GCD must have ± 1 on the first element since this means the number of coefficient vectors of $f(\vec{x})$ and $g(\vec{x})$ reduced by the coefficient vectors of cofactors. Moreover, this can be thought as the closest vector problem (CVP) hence it may be possible to use Babai's nearest plane algorithm ([1]) instead of the method based on the lattice in Lemma 7.

Example 8
For polynomials in Example 5, we have the following matrices with the base number $b = 10$, length $w = 2$, order $r = 0$, $c_{\mathcal{L}} = 9658$ and $c_{\mathcal{H}} = 4829$ if we use the Landau-Mignotte bound of $f(x)$ and $g(x)$.

$$\mathcal{L}_{10,2}(f,g,0,9658) =$$
$$\left(\begin{array}{cccccc|cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 28974 & 0 & 48290 & 86922 & \cdots & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 28974 & 0 & \cdots & 19316 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 48290 & 28974 & 19316 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 9658 & 48290 & 28974 & 19316 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 9658 & 48290 & \cdots & 28974 & 19316 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 48290 & 57948 & 28974 & 19316 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & -9658 & 96580 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9658 & 96580 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -9658 & 96580 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -9658 & 96580
\end{array}\right),$$

$$\mathcal{H}_{10,2}(f,g,0,4829,t,s) =$$
$$\left(\begin{array}{ccc|ccccccccccc}
1 & 0 & 0 & 4829 & 24145 & 14487 & 9658 & 24145 & \cdots & 43461 & 24145 & 14487 & 9658 & 0 \\
0 & 1 & 0 & 0 & 14487 & 0 & 19316 & 0 & \cdots & 33803 & 0 & 24145 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 14487 & 0 & \cdots & 28974 & 0 & 33803 & 0 & 24145 \\
\hline
0 & 0 & 0 & -4829 & 48290 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -4829 & 48290 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -4829 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 48290 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & -4829 & 48290 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & -4829 & 48290
\end{array}\right).$$

By the LLL algorithm we found the following short vectors and in fact their first rows are corresponding to the coefficient vectors of cofactors and GCD of $f(x)$ and $g(x)$.

$$\mathcal{L}_{10,2}(f,g,0,9658) \Rightarrow$$
$$\left(\begin{array}{cccccc|ccccccccc}
3 & 4 & 8 & -6 & -7 & -5 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\
-1 & 2 & -2 & 0 & 0 & 1 & -28974 & 0 & 0 & 9658 & \cdots & -9658 & 0 & -9658 & 19316
\end{array}\right),$$
$$\mathcal{H}_{10,2}(f,g,0,4829,t,s) \Rightarrow$$
$$\left(\begin{array}{ccc|ccccccccccc}
1 & -5 & -4 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 14487 & 0 & 4829 & 0 & \cdots & 4829 & 0 & -9658 & 0 & -24145
\end{array}\right).$$

Note that 1) we show only the first and second shortest short vectors found though there are more short vectors that are not corresponding to approximate cofactors and GCD, and 2) the LLL algorithm can find the expected short vectors with much smaller c_L and $c_{\mathcal{H}}$ in most cases. In fact, short vectors in this example can be computed from $\mathcal{L}_{10,2}(f,g,0,\underline{10})$ and $\mathcal{H}_{10,2}(f,g,0,\underline{10},t,s)$. ◁

3.2 Algorithm in Digits-wise Representation

We consider the case introduced in the beginning of this section hence we assume that all the coefficients have a priori errors on only the limited number of digits. For such polynomials, the resulting tolerance ε defined in Definition 1 easily becomes large even though the norm of errors in the digits-wise representation is small. We need to adapt the definition to the digits-wise representation. By the following definition, we have digits-wise tolerances $\varepsilon_{10,1} = \varepsilon = 20$, $\varepsilon_{10,2} = 2$ and $\varepsilon_{5,2} = 4$ in the ∞-norm for the pair of $f(x) = (4x+5)(8x^2+4x+3) + 20x^2 - 10x$ and $g(x) = (4x+5)(5x^2+7x+6) - 10x^3 + 10$ for example.

Definition 9 (Digits-wise Approximate Polynomial GCD Over Integers)
*Let $f(\vec{x})$ and $g(\vec{x})$ be polynomials in variables $\vec{x} = x_1, \ldots, x_\ell$ over $\mathbb{Z}$, and let ε be a small positive integer. If they satisfy $f(\vec{x}) = t(\vec{x})h(\vec{x}) + \Delta_f(\vec{x})$, $g(\vec{x}) = s(\vec{x})h(\vec{x}) + \Delta_g(\vec{x})$ and $\varepsilon_{b,w} = \max\{\|\mathrm{vect}_{b,w}(\Delta_f)\|, \|\mathrm{vect}_{b,w}(\Delta_g)\|\}$ for some polynomials $\Delta_f, \Delta_g \in \mathbb{Z}[\vec{x}]$, then we say that the above polynomial $h(\vec{x})$ is an **digits-wise approximate GCD over integers** w.r.t. the base number b and length w. We also say that $t(\vec{x})$ and $s(\vec{x})$ are **digits-wise approximate cofactors over integers**, and we say that their **tolerance** is $\varepsilon_{b,w}$. ($\|p\|$ denotes a suitable vector norm.)* ◁

For computing digits-wise approximate GCD over integers, the lemmas introduced above do not guarantee that we can find the coefficient vectors of approximate cofactors and approximate GCD by the LLL algorithm. However, as same as the algorithms in [18], the short vectors found have a possibility that corresponding polynomials $t(\vec{x})$ and $s(\vec{x}) \in \mathbb{Z}[\vec{x}]$ satisfy $s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \approx$

0, and they can be candidate approximate cofactors. Moreover, in the digits-wise representation, we have to distinguish correct digits from erroneous digits in the digits-wise lattice. We define the following diagonal weight matrix $\mathcal{W}_{b,w}(k_{id}, k_{cf}, c, \mathcal{E}, c_{\mathcal{E}})$ to distinguish them.

$$\mathcal{W}_{b,w}(k_{id}, k_{cf}, c, \mathcal{E}, c_{\mathcal{E}}) = \mathrm{diag}(\underbrace{1, \ldots, 1}_{k_{id}} \underbrace{\vec{w}, \ldots, \vec{w}}_{k_{cf}}),\ \vec{w} = \{c_{w-1}, \ldots, c_0\},\ c_i = \begin{cases} c_{\mathcal{E}} & (i \in \mathcal{E}) \\ c & (i \notin \mathcal{E}) \end{cases}$$

where we assume that the coefficients have a priori error on the i-th digits in the base b representation for any $i \in \mathcal{E} \subset \mathbb{Z}_{>0}$, and c and $c_{\mathcal{E}}$ are penalty weights that force the LLL algorithm to reduce more correct digits (columns) than other digits and reduce more erroneous digits than coefficient digits of candidate factors, respectively in the lattice basis. With this diagonal weight matrix, we define the following matrices that are based on $\mathcal{L}_{b,w}(f, g, r, 1)$ and $\mathcal{H}_{b,w}(f, g, r, 1, t, s)$, respectively.

$$\begin{aligned} \tilde{\mathcal{L}}_{b,w}(f, g, r, c, \mathcal{E}, c_{\mathcal{E}}) &= \mathcal{L}_{b,w}(f, g, r, 1)\mathcal{W}_{b,w}(\beta_{n-1,r} + \beta_{m-1,r}, w\beta_{n+m-1,r}, c, \mathcal{E}, c_{\mathcal{E}}), \\ \tilde{\mathcal{H}}_{b,w}(f, g, r, c, t, s, \mathcal{E}, c_{\mathcal{E}}) &= \mathcal{H}_{b,w}(f, g, r, 1, t, s)\mathcal{W}_{b,w}(\beta_{r+1,0} + 1, w(\beta_{n,0} + \beta_{m,0}), c, \mathcal{E}, c_{\mathcal{E}}). \end{aligned}$$

Lemma 10
Let B be the maximum absolute value of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$ with perturbations. For the lattice generated by the row vectors of $\tilde{\mathcal{L}}_{b,w}(f, g, r, c_{\tilde{\mathcal{L}}}, \mathcal{E}, c_{\mathcal{E}})$ with the following $c_{\tilde{\mathcal{L}}}$, the LLL algorithm can find a short vector whose first $\beta_{n-1,r} + \beta_{m-1,r}$ elements are a multiple of the transpose of the coefficient vectors of candidate approximate cofactors of $f(\vec{x})$ and $g(\vec{x})$.

$$c_{\tilde{\mathcal{L}}} = 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2} \sqrt{(\beta_{n-1,r} + \beta_{m-1,r})B^2 + (\#\mathcal{E} \times \beta_{n+m-1,r})(b-1)^2 c_{\mathcal{E}}^2}$$

where $\#\mathcal{E}$ is the number of elements in $\mathcal{E}$. ◁

Proof Let $t(\vec{x})$ and $s(\vec{x})$ be one of candidate approximate cofactors of $f(\vec{x})$ and $g(\vec{x})$, respectively, satisfying $\| \mathrm{vect}_{b,w}(s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x})) \| \approx 0$. In this case, the lattice generated by rows of $\tilde{\mathcal{L}}_{b,w}(f, g, r, c_{\tilde{\mathcal{L}}}, \mathcal{E}, c_{\mathcal{E}})$ has the following vector $\vec{u}_{cac}$ for some integer r.

$$\vec{u}_{cac} = (\text{the transpose of the coefficient vectors of } s(\vec{x}) \text{ and } t(\vec{x}),\ \underbrace{* \ \cdots \ *}_{w \times \beta_{n+m-1,r}})$$

where all the correct digits are 0 on the right $w \times \beta_{n+m-1,r}$ elements denoted by $*$. The shortest vector of this lattice must be smaller than or equal to $\vec{u}_{cac}$ hence the LLL algorithm can find a short vector $\vec{u}$ satisfying

$$\begin{aligned} \|\vec{u}\|_2 &\leq 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2} \|\vec{u}_{cac}\|_2 \\ &\leq 2^{(\beta_{n-1,r}+\beta_{m-1,r}+(w-1)\beta_{n+m-1,r}-1)/2} \sqrt{(\beta_{n-1,r} + \beta_{m-1,r})B^2 + (\#\mathcal{E} \times \beta_{n+m-1,r})(b-1)^2 c_{\mathcal{E}}^2} \end{aligned}$$

since the left $\beta_{n-1,r} + \beta_{m-1,r}$ elements of $\vec{u}_{cac}$ are bounded by B and the erroneous digits on the right $w \times \beta_{n+m-1,r}$ elements of $\vec{u}_{cac}$ are bounded by $(b-1)c_{\mathcal{E}}$.

Therefore, all the correct digits on the right $w \times \beta_{n+m-1,r}$ elements of the found short vector $\vec{u}$ must be zeros since all the non-zero correct digits on the right $w \times \beta_{n+m-1,r}$ elements of row vectors in the lattice generated by the row vectors of $\tilde{\mathcal{L}}_{b,w}(f, g, r, c_{\tilde{\mathcal{L}}}, \mathcal{E}, c_{\mathcal{E}})$ are larger than or equal to $c_{\tilde{\mathcal{L}}}$ in absolute value. This means that the polynomials $t(\vec{x})$ and $s(\vec{x})$ whose coefficient vectors are the first $\beta_{n-1,r} + \beta_{m-1,r}$ elements of $\vec{u}$ satisfy

$$\|\text{all the correct digits of } \mathrm{vect}_{b,w}(s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}))\| = 0$$

hence they are candidate approximate cofactors of $f(\vec{x})$ and $g(\vec{x})$ though we may not guarantee $\|\mathrm{vect}_{b,w}(s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}))\| \approx 0$. ∎

Lemma 11
Let B be the same maximum in Lemma 10. For the lattice generated by the row vectors of $\tilde{\mathcal{H}}_{b,w}(f, g, r, c_{\tilde{\mathcal{H}}}, t, s, \mathcal{E}, c_{\mathcal{E}})$ with the following $c_{\tilde{\mathcal{H}}}$, the LLL algorithm can find a short vector whose 2-nd, …, $(\beta_{r+1,0}+1)$-th elements are a multiple of the transpose of the coefficient vector of a candidate approximate GCD of $f(\vec{x})$ and $g(\vec{x})$.

$$c_{\tilde{\mathcal{H}}} = 2^{(\beta_{r+1,0}+1+(w-1)(\beta_{n,0}+\beta_{m,0})-1)/2}\sqrt{(\beta_{r+1,0}+1)B^2 + (\#\mathcal{E}\times(\beta_{n,0}+\beta_{m,0}))(b-1)^2c_{\mathcal{E}}^2}$$

where $\#\mathcal{E}$ is the number of elements in $\mathcal{E}$. ◁

Proof The proof is similar to that of Lemma 10. ∎

In general, there are short vectors that are not corresponding to approximate cofactors nor approximate GCD with small perturbations (small tolerance) hence the above lemmas can not guarantee that our algorithm always can find such a good approximate GCD. However, in most cases, according to our numerical experiment in Section **4**, the following algorithm works well, in which we use $c_{\mathcal{E}} = \sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$ and $c_{\mathcal{E}} = \sqrt{\beta_{r+1,0}+1}B$ for $\tilde{\mathcal{L}}_{b,w}(f, g, r, c, \mathcal{E}, c_{\mathcal{E}})$ and $\tilde{\mathcal{H}}_{b,w}(f, g, r, c, t, s, \mathcal{E}, c_{\mathcal{E}})$, respectively. We again note that $c_{\mathcal{E}}$ is a scaling weight to make the LLL algorithm do reducing more erroneous digits than coefficient digits of candidate cofactors and GCD, as in the proofs of Lemma 6 and Lemma 10.

Algorithm 12 (digits-wise approximate GCD over integers)
Input*: $f, g \in \mathbb{Z}[\vec{x}], n = \mathrm{tdeg}(f), m = \mathrm{tdeg}(g),\ b, w \in \mathbb{Z}_{>0},\ \mathcal{E} \subset \{0, 1, \ldots, w-1\}$.*
Output*: $h, t, s \in \mathbb{Z}[\vec{x}]$ satisfying $f(\vec{x}) \approx t(\vec{x})h(\vec{x})$ and $g(\vec{x}) \approx s(\vec{x})h(\vec{x})$, or "not found".*

1. $\varepsilon \leftarrow 1$ *and while* $\varepsilon < \min\{\|\mathrm{vect}_{b,w}(f)\|, \|\mathrm{vect}_{b,w}(g)\|\}$ *do* **2–14**
(or do once for the possible smallest ε)
2. $r \leftarrow \min\{n, m\} - 1$ *and while* $r \geq 0$ *do* **3–13** *(or do once for $r = 0$)*
3. $c \leftarrow \max\{\|f\|, \|g\|\}$ *and construct a matrix* $\tilde{\mathcal{L}}_{b,w}(f, g, r, c, \mathcal{E}, c_{\mathcal{E}})$
4. *while* $c \leq c_{\tilde{\mathcal{L}}}$ *do* **5–12** *(or do once for $c = \max\{\|f\|, \|g\|\}$)*
5. *apply the LLL algorithm to the lattice generated by the row vectors of* $\tilde{\mathcal{L}}_{b,w}(f, g, r, c, \mathcal{E}, c_{\mathcal{E}})$
6. *for each basis vector sorted by the norm of right $w\beta_{n+m-1,r}$ columns, do* **7–11**
7. $c' \leftarrow \max\{\|f\|, \|g\|\}$ *and construct a matrix* $\tilde{\mathcal{H}}_{b,w}(f, g, r, c, t, s, \mathcal{E}, c_{\mathcal{E}})$
8. *while* $c' \leq c_{\tilde{\mathcal{H}}}$ *do* **9–11** *(or do once for $c' = \max\{\|f\|, \|g\|\}$)*
9. *apply the LLL algorithm to the lattice generated by the row vectors of* $\tilde{\mathcal{H}}_{b,w}(f, g, r, c, t, s, \mathcal{E}, c_{\mathcal{E}})$
10. *let $h(\vec{x}), t(\vec{x}), s(\vec{x})$ be candidate approximate GCD and cofactors, and output $h(\vec{x}), t(\vec{x}), s(\vec{x})$ if* $\max\{\|\mathrm{vect}_{b,w}(f - th)\|, \|\mathrm{vect}_{b,w}(g - sh)\|\} \leq \varepsilon$
11. $c' \leftarrow c' \times \max\{\|f\|, \|g\|\}$ *(or multiply some positive integer)*
12. $c \leftarrow c \times \max\{\|f\|, \|g\|\}$ *(or multiply some positive integer)*
13. $r \leftarrow r - 1$
14. $\varepsilon \leftarrow \varepsilon \times 10$ *(or multiply/add some positive integer)*
15. *output "not found".*

Example 13
Algorithm 12 works for polynomials $f(x_1, x_2)$ and $g(x_1, x_2)$ below as follows.

$$\begin{aligned} f(x_1, x_2) &= 15336x_1^2 - 3651x_1x_2 - 11673x_1 - 1271x_2^2 + 11618x_2 - 15979, \\ g(x_1, x_2) &= 23184x_1^2 - 15094x_1x_2 + 53046x_1 + 2425x_2^2 - 19493x_2 + 26112. \end{aligned}$$

We assume that these polynomials have a priori errors on their 3*rd* (4^2) *and* 4*th* (4^3) *digits of coefficients in the base* $b = 4$ *representation (note:* $\log_4(\max\{\| f \|_\infty, \| g \|_\infty\}) \approx 7.85$*). By the algorithm, we reduce the lattice generated by the row vectors of the following matrix of size* 76×86 *with* $c_{\tilde{\mathcal{L}}} = 6986206386174202099$ *and* $c_{\mathcal{E}} = 2671636$*.*

$\tilde{\mathcal{L}}_{4,8}(f, g, 0, c_{\tilde{\mathcal{L}}}, \{2, 3\}, c_{\mathcal{E}}) =$

$$\left(\begin{array}{cccccc|cc|c} 1 & 0 & 0 & 0 & 0 & 0 & 6986206386174202099 & 139724...48404198 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -209586...22606297 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & -6986206386174202099 & 279448...96808396 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -69862...74202099 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 27944825544696808396 \end{array}\right).$$

We found the following short vectors that are sorted by the norm of right columns.

$$\left(\begin{array}{cccccc|cccc|ccc} 313 & -41 & -213 & 512 & -71 & 322 & 0 & 0 & 0 & 0 & -320596320 & -85492352 & 0 \; 0 \; \cdots \; 0 \\ 165 & -21 & -113 & 272 & -43 & 170 & 0 & 0 & 0 & 0 & -1485429616 & -371357404 & 0 \; 0 \; \cdots \; 0 \\ 295 & -39 & -203 & 480 & -73 & 302 & 0 & 0 & 0 & 0 & 1301086732 & 325939592 & 0 \; 0 \; \cdots \; 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \; \vdots \; \ddots \; \vdots \end{array}\right).$$

We construct the following matrix of size 88×100 *for the first short vector in the step* ***9*** *with* $c_{\tilde{\mathcal{H}}} = 399729686425627882725$ *and* $c_{\mathcal{E}} = 2181382$*.*

$\tilde{\mathcal{H}}_{4,8}(f, g, 0, c_{\tilde{\mathcal{H}}}, t, s, \{2, 3\}, c_{\mathcal{E}}) =$

$$\left(\begin{array}{cccc|cc|c} 1 & 0 & 0 & 0 & 0 & -119918...83648175 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \cdots & -799459372851255765450 \\ \hline 0 & 0 & 0 & 0 & -399729686425627882725 & 159891...11530900 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & -39972...27882725 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 1598918745702511530900 \end{array}\right).$$

We found the following short vectors that are sorted by the norm of right columns. We show only short vectors having ± 1 *on their first elements as noted just after Lemma 7.*

$$\left(\begin{array}{cccc|cccc|ccc} 1 & 51 & -31 & 72 & 0 & 0 & 0 & 0 & 0 & -2181382 & 0 \; \cdots \; 0 \\ 1 & 51 & -27 & 72 & 0 & 0 & 0 & 0 & 0 & -2181382 & 0 \; \cdots \; 0 \\ 1 & 51 & -29 & 72 & 0 & 0 & 0 & 0 & 0 & -2181382 & 0 \; \cdots \; 0 \\ 1 & 49 & -31 & 72 & 0 & 0 & 0 & 0 & -19632438 & -8725528 & -399729686425627882725 \; \cdots \; 0 \end{array}\right).$$

Hence, we get $213x_1+41x_2-313$ *and* $322x_1-71x_2+512$ *as approximate cofactors,* $72x_1-31x_2+51$ *as an approximate GCD of* $f(x_1,x_2)$ *and* $g(x_1,x_2)$*, and* $\varepsilon_{4,8} = \sqrt{38} \approx 6.16$ *in the Euclidean norm. Moreover, the perturbation polynomials are* $(3\times 4^2-4^3)+(-3\times 4^2-2\times 4^3)x_2$ *and* $(-3\times 4^2-3\times 4^3)x_1+(2\times 4^2+3\times 4^3)x_2^2$*.* ◁

Example 14
Though the discussions above and Algorithm 12 are only for the case of two polynomials, it is easy to extend them to several polynomials, using the generalized subresultant mapping (see also [25],[18]). We show some example of the case of three polynomials below as follows.

$$\begin{aligned} f(x_1,x_2) &= 23112x_1^2-6999x_1x_2-6117x_1-1271x_2^2+11730x_2-15963,\\ g(x_1,x_2) &= 2304x_1^2-6104x_1x_2+38432x_1+2201x_2^2-19493x_2+26224,\\ h(x_1,x_2) &= -3744x_1^2+24724x_1x_2+6060x_1-9951x_2^2+12700x_2+6139. \end{aligned}$$

We assume that these polynomials have a priori errors on their 2*nd* (16^1) *digits of coefficients in the base* $b=16$ *representation (note:* $\log_{16}(\max\{\|f\|_\infty,\|g\|_\infty,\|h\|_\infty\})\approx 3.81$*). We construct a matrix of size* 69×89 *which is similar to* $\tilde{\mathcal{L}}_{b,w}(f,g,r,c_{\tilde{\mathcal{L}}},\mathcal{E},c_{\mathcal{E}})$ *with* $c_{\tilde{\mathcal{L}}} = 2816708953910864585$ *and* $c_{\mathcal{E}} = 2443811$ *and found the following short vectors that are sorted by the norm of right columns.*

$$\left(\begin{array}{ccccccccc|cccccc} -313 & 41 & 321 & -512 & 71 & -32 & -121 & -321 & 52 & 0 & 0 & -5354389901 & \cdots & 0\\ 496 & -64 & -512 & 816 & -112 & 48 & 192 & 512 & -80 & 0 & 0 & -2856815059 & \cdots & 0\\ -205 & 29 & 213 & -336 & 51 & -16 & -77 & -213 & 36 & 0 & 0 & -1886622092 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right).$$

For the first short vector found, corresponding to candidate three cofactors, we construct a matrix of size 58×76 *which is similar to* $\tilde{\mathcal{H}}_{b,w}(f,g,r,c_{\tilde{\mathcal{H}}},t,s,\mathcal{E},c_{\mathcal{E}})$ *with* $c_{\tilde{\mathcal{H}}} = 39365206313183407$ *and* $c_{\mathcal{E}} = 1629208$ *and found the following short vectors that are sorted by the norm of right columns.*

$$\left(\begin{array}{cccc|cccccccccccccccccccccccccc} -1 & 51 & -31 & 72 & 0 & 0 & 0 & 0 & 0 & 0 & 6516832 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4887624 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0\\ -1 & 51 & -31 & 72 & 0 & 0 & 0 & 0 & 0 & 0 & 6516832 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4887624 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0\\ -1 & 51 & -31 & 72 & 0 & 0 & 0 & 0 & 0 & 0 & 6516832 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -4887624 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right).$$

Hence, we get $321x_1+41x_2-313$*,* $32x_1-71x_2+512$ *and* $-52x_1+321x_2+121$ *as approximate cofactors,* $72x_1-31x_2+51$ *as an approximate GCD of* $f(x_1,x_2)$*,* $g(x_1,x_2)$ *and* $h(x_1,x_2)$*, and* $\varepsilon_{16,4} = \sqrt{65}\approx 8.06$ *in the Euclidean norm. Moreover, the perturbation polynomials are* $3\times 16^1x_1-4\times 16^1x_2$*,* $-4\times 16^1x_1+7\times 16^1$ *and* $5\times 16^1x_2-2\times 16^1$*.* ◁

4 Remarks

To see the efficiency of Algorithm 12, we have generated several sets of 100 pairs of polynomials: A pair of bivariate polynomials of total degree randomly chosen from $[2,6]$, having their GCD of total degree randomly chosen from $[1,3]$, coefficients of their factors randomly chosen from $[-100,100]$ and added noise bivariate polynomials of the same total degree, whose coefficients are randomly chosen from $[-9,9]\times 10^k$ but 0 at α probability, for randomly chosen erroneous digit k within the coefficient size. For example, the following pair of polynomials is one of them ($\alpha = 0.0$

and $k = 4$).

$$\begin{cases} (-85x_1^3 + 21x_2x_1^2 + 88x_1^2 + 18x_2^2x_1 - 99x_2x_1 + 17x_1 + 95x_2^3 - 49x_2^2 - 89x_2 - 96) \\ \quad \times(46x_1 + 92x_2 + 47) + (8 \times 10^4 x_1^4 + 1 \times 10^4 x_2x_1^3 - 1 \times 10^4 x_1^3 + 4 \times 10^4 x_2x_1^2 \\ \quad -7 \times 10^4 x_1^2 - 6 \times 10^4 x_2^3x_1 + 8 \times 10^4 x_2^2x_1 - 6 \times 10^4 x_2x_1 - 8 \times 10^4 x_1 \\ \quad -5 \times 10^4 x_2^4 - 7 \times 10^4 x_2^3 + 7 \times 10^4 x_2^2 + 7 \times 10^4 x_2 - 6 \times 10^4), \\ (-85x_1^3 + 21x_2x_1^2 + 88x_1^2 + 18x_2^2x_1 - 99x_2x_1 + 17x_1 + 95x_2^3 - 49x_2^2 - 89x_2 - 96) \\ \quad \times(-80x_1 + 83x_2 + 62) + (-8 \times 10^4 x_1^4 + 7 \times 10^4 x_2x_1^3 + 5 \times 10^4 x_1^3 + 9 \times 10^4 x_2^2x_1^2 \\ \quad -6 \times 10^4 x_2x_1^2 - 5 \times 10^4 x_1^2 + 8 \times 10^4 x_2^3x_1 + 4 \times 10^4 x_2^2x_1 - 9 \times 10^4 x_2x_1 \\ \quad -4 \times 10^4 x_1 + 5 \times 10^4 x_2^4 - 2 \times 10^4 x_2^3 + 6 \times 10^4 x_2^2 - 9 \times 10^4 x_2). \end{cases}$$

We have computed their approximate GCDs by the algorithm with $\varepsilon = 10$ in the step **1**, $r = 0$ in the step **2** and $c_{\tilde{\mathcal{L}}} = c_{\tilde{\mathcal{H}}} = 10^{10}$ and $c_{\mathcal{E}} = 10^5$ in the steps **3** and **7**. Note that all the experiments have been computed by our preliminary implementation on Mathematica 8.0, and we use the max norm for polynomials. Table 1 shows the results where "#success" denotes the number of pairs for which we got the expected digits-wise approximate polynomial GCD over integers and "#failure" denotes otherwise. According to the result, our algorithm works well for most of pairs of polynomials. However, the computation time is not good since the time-complexity of the lattice basis reduction is heavily depending on the number of bases that is the number of rows of matrices in our algorithm. Therefore, our algorithm works well but any faster algorithm is required to be used in the practical situation.

probability α	0.75		0.5		0.0	
	1st set	2nd set	1st set	2nd set	1st set	2nd set
#success:#failure	99:1	99:1	93:7	96:4	97:3	91:9

Table 1: The result of our experiments

Although we consider about only polynomials over integers in this paper, the digits-wise representation can be extended to polynomials over reals or complexes. For example, we can construct the Sylvester matrix of the given polynomials over reals in the digits-wise representation: dividing mantissae of coefficients into several elements if the given polynomials do not have both of small and large exponential parts. This may help us to treat erroneous coefficients having errors on only higher bits and should be studied as a further work.

The preliminary implementation on Mathematica 8.0, of our algorithm introduced in this paper with some examples can be found at the following URL: `http://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/snc2011plus.nb`.

Acknowledgments

The author would like to thank Prof. Kaltofen for having the personal conversation on approximate polynomial GCD over integers which is very helpful for the ideal of digit-wise lattice. Moreover, this work was supported in part by Japanese Ministry of Education, Culture, Sports, Science and Technology under Grant-in-Aid for Young Scientists, MEXT KAKENHI (22700011).

References

[1] L. Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[2] K. Batselier, P. Dreesen, and B. De Moor. A geometrical approach to finding multivariate approximate LCMs and GCDs. *Linear Algebra Appl.*, 438(9):3618–3628, 2013.

[3] G. Chèze, A. Galligo, B. Mourrain, and J.-C. Yakoubsohn. A subdivision method for computing nearest gcd with certification. *Theoret. Comput. Sci.*, 412(35):4493–4503, 2011.

[4] D. Christou and M. Mitrouli. Estimation of the greatest common divisor of many polynomials using hybrid computations performed by the ERES method. *Appl. Numer. Anal. Comput. Math.*, 2(3):293–305, 2005.

[5] R. M. Corless, S. M. Watt, and L. Zhi. *QR* factoring to compute the GCD of univariate approximate polynomials. *IEEE Trans. Signal Process.*, 52(12):3394–3402, 2004.

[6] G. M. Diaz-Toca and L. Gonzalez-Vega. Computing greatest common divisors and square-free decompositions through matrix methods: the parametric and approximate cases. *Linear Algebra Appl.*, 412(2-3):222–246, 2006.

[7] M. Elkadi, A. Galligo, and T. L. Ba. Approximate GCD of several univariate polynomials with small degree perturbations. *J. Symbolic Comput.*, 47(4):410–421, 2012.

[8] I. Z. Emiris, A. Galligo, and H. Lombardi. Numerical univariate polynomial GCD. In *The mathematics of numerical analysis (Park City, UT, 1995)*, volume 32 of *Lectures in Appl. Math.*, pages 323–343. Amer. Math. Soc., Providence, RI, 1996.

[9] I. Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure Appl. Algebra*, 117/118:229–251, 1997. Algorithms for algebra (Eindhoven, 1996).

[10] S. Gao, E. Kaltofen, J. May, Z. Yang, and L. Zhi. Approximate factorization of multivariate polynomials via differential equations. In *ISSAC 2004*, pages 167–174. ACM, New York, 2004.

[11] N. Howgrave-Graham. Approximate integer common divisors. In *Cryptography and lattices (Providence, RI, 2001)*, volume 2146 of *Lecture Notes in Comput. Sci.*, pages 51–66. Springer, Berlin, 2001.

[12] N. Karcanias, S. Fatouros, M. Mitrouli, and G. H. Halikias. Approximate greatest common divisor of many polynomials, generalised resultants, and strength of approximation. *Comput. Math. Appl.*, 51(12):1817–1830, 2006.

[13] N. K. Karmarkar and Y. N. Lakshman. On approximate GCDs of univariate polynomials. *J. Symbolic Comput.*, 26(6):653–666, 1998. Symbolic numeric algebra for polynomials.

[14] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[15] T. Y. Li and Z. Zeng. A rank-revealing method with updating, downdating, and applications. *SIAM J. Matrix Anal. Appl.*, 26(4):918–946 (electronic), 2005.

[16] M. Mitrouli and N. Karcanias. Computation of the GCD of polynomials using Gaussian transformations and shifting. *Internat. J. Control*, 58(1):211–228, 1993.

[17] K. Nagasaka. Approximate polynomial gcd over integers. *ACM Communications in Computer Algebra*, 42(3):124–126, 2008. (ISSAC 2008 poster session).

[18] K. Nagasaka. Approximate polynomial gcd over integers. *J. Symbolic Comput.*, 46(12):1306–1317, 2011.

[19] K. Nagasaka. An improvement in the lattice construction process of approximate polynomial gcd over integers. In *Proceedings of Symbolic-Numeric Computation (SNC2011)*, pages 63–64. 2011. (extended abstract).

[20] K. Nagasaka and T. Masui. Extended qrgcd algorithm. In V. Gerdt, W. Koepf, E. Mayr, and E. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, volume 8136 of *Lecture Notes in Computer Science*, pages 257–272. Springer International Publishing, 2013.

[21] M. Ochi, M. Noda, and T. Sasaki. Approximate greatest common divisor of multivariate polynomials and its application to ill-conditioned systems of algebraic equations. *J. Inform. Process.*, 14(3):292–300, 1991.

[22] V. Y. Pan. Approximate polynomial gcds, Padé approximation, polynomial zeros and bipartite graphs. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (San Francisco, CA, 1998)*, pages 68–77, New York, 1998. ACM.

[23] V. Y. Pan. Computation of approximate polynomial GCDs and an extension. *Inform. and Comput.*, 167(2):71–85, 2001.

[24] C. Rössner and J.-P. Seifert. The complexity of approximate optima for greatest common divisor computations. In *Algorithmic number theory (Talence, 1996)*, volume 1122 of *Lecture Notes in Comput. Sci.*, pages 307–322. Springer, Berlin, 1996.

[25] D. Rupprecht. An algorithm for computing certified approximate GCD of *n* univariate polynomials. *J. Pure Appl. Algebra*, 139(1-3):255–284, 1999. Effective methods in algebraic geometry (Saint-Malo, 1998).

[26] T. Sasaki and M. Noda. Approximate square-free decomposition and root-finding of ill-conditioned algebraic equations. *J. Inform. Process.*, 12(2):159–168, 1989.

[27] A. Schönhage. Quasi-gcd computations. *J. Complexity*, 1(1):118–137, 1985.

[28] J. von zur Gathen, M. Mignotte, and I. E. Shparlinski. Approximate polynomial gcd: Small degree and small height perturbations. *J. Symbolic Comput.*, 45(8):879–886, 2010.

[29] J. von zur Gathen and I. E. Shparlinski. Approximate polynomial gcd: small degree and small height perturbations. In *LATIN 2008: Theoretical informatics*, volume 4957 of *Lecture Notes in Comput. Sci.*, pages 276–283. Springer, Berlin, 2008.

[30] C. J. Zarowski, X. Ma, and F. W. Fairman. QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. *IEEE Trans. Signal Process.*, 48(11):3042–3051, 2000.

[31] Z. Zeng and B. H. Dayton. The approximate GCD of inexact polynomials. II. A multivariate algorithm. In *ISSAC 2004*, pages 320–327. ACM, New York, 2004.

[32] L. Zhi. Displacement structure in computing approximate GCD of univariate polynomials. In *Computer mathematics*, volume 10 of *Lecture Notes Ser. Comput.*, pages 288–298. World Sci. Publ., River Edge, NJ, 2003.

[33] L. Zhi and M. Noda. Approximate GCD of multivariate polynomials. *Sūrikaisekikenkyūsho Kōkyūroku*, (1138):64–76, 2000. Research on the theory and applications of computer algebra (Japanese) (Kyoto, 1999).

[34] L. H. Zhi and M. Noda. Approximate GCD of multivariate polynomials. In *Computer mathematics (Chiang Mai, 2000)*, volume 8 of *Lecture Notes Ser. Comput.*, pages 9–18. World Sci. Publ., River Edge, NJ, 2000.

Communications of JSSAC (2016)
Vol. 2, pp. 33 – 42

Practice of Drawing Graphs of Implicit Functions of Three Variables

Noriko Hyodo
Salesian Polytechnic

Yuji Kondoh
National Institute of Technology, Kagawa College

Hirokazu Murao
The University of Electro-Communications

Tomokatsu Saito
AlphaOmega Inc.

Tadashi Takahashi
Konan University

(Received 31/Oct/2014 Accepted 16/Feb/2015)

Abstract

We have long been working on developing algorithms for drawing graphs of implicit functions. In this paper, we investigate methods for trivariate cases as a simple extension of the existing methods for bivariate cases. The basic strategy of our methods consists of division of the 3D space in the display area into a contiguous sequence of tiny cubes, and the check for every cube of the existence of the solutions to the polynomial equation defining an implicit function. The method itself is characterized by the exact treatment with mathematical preciseness of numeric calculation and formula manipulation provided by computer algebra system. The underlying mathematics used in our most precise algorithm for bivariate cases cannot be extended direct to trivariate cases, and there remains a small chance even for our current most rigorous algorithm to miss the detection of a closed curve of solutions if it is isolated inside a cube. We explain how we can simply extend our method to trivariate cases, and investigate the situations of the solution graphs that each algorithm may miss. We also show some sample results obtained by our several experimental implementations.

1 Introduction

There have been developed many mathematical software systems which support the functionality of graph drawing. Visualization is useful for roughly grasping the behavior of mathematical functions. It is important with visualization to present the overall behavior exactly and also not to lose critical nature at some distinct points. Despite this, how to establish an accurate drawing method or the preciseness of drawing methods are rarely discussed. Drawing the figure of the real zeros of the equation of a multivariate polynomial, in other words, the graph of a real algebraic function, is an old problem hard to solve[1]. In general, the problem to obtain the real solution curves of a polynomial exactly still be very difficult, because they may have isolated points or curves. Actually,

not a few existing software systems may fail without any notice. In this paper, we treat polynomial equations with rational coefficients, and describe a method to plot the solution curves precisely.

The computer algebra system Risa/Asir[2] equips with a function, called *ifplot*, for plotting the curves of implicit functions defined by bivariate polynomials. The method of plotting is based on a principle for a graph to be mathematically precise, and on this principle, an algorithm is developed for obtaining the solution curves. The effectiveness of the proposed principle and the practicality of the algorithms has been demonstrated by examples. More concretely, the method makes, internally, use of an evaluation function of a square region to tell whether it contains any solution(s) of an implicit function. We call this evaluation function *character* function (or *character* for short). Every region covering the display area is checked by a character function. Character function can be implemented in various ways, depending on the requirement for efficiency and mathematical correctness in practice. The ultimate condition expected for character is to guarantee the existence or non-existence of solutions. As described in [4] and implemented with `ifplot`, such an ultimate character can be realized for 2D cases by making use of Gröbner basis calculation. This method cannot be directly extended to 3D cases.

Nowadays, various types of devices for 3D plotting are available, such as 3D display terminals and 3D printers, and they are becoming very common in recent years. Furthermore, processors widely used have obtained excessive high-performance, and it is expected that a huge amount of calculations required for 3D graphics can be processed with reasonable efficiency by utilizing the performance. With this recognition in mind, we shall treat implicit functions of three variables in our plotting context with Risa/Asir, and we investigate our method toward the 3D extension in this paper.

2 Fundamentals

2.1 Basic Concepts and Principle

Drawing the graph of mathematical functions is commonly recognized as an easy problem, despite the difficulty in mathematical precise treatment. Every existing display device of any kind consists of a large amount of “spots”, each of which has finite area or volume determined by the resolution. A curve or a graph is usually drawn, regardless of connected or non-connected, as a set of spots, thus, having some area or volume, although a point or a curve cannot have a width or thickness in mathematical senses. Spot is often termed as *pixel* in 2D cases and *voxel* in 3D cases. Then, in what sense and to what extent the drawn figure is precise? It is difficult to give an answer to this question in a mathematical well-defined manner. Notice that a graph shows only a sketch of the true mathematical curve. By employing algebraic computation, however, we can give a mathematically definite meaning of the drawn figure of the solution curves. In [5, 4], the authors described, starting from the definitions of mathematical basic notions, the abstract meaning of plotting, and how we can improve the mathematical preciseness in the computing methods actually used in practice, in the context of plotting in the 2D space. As a preparation for the 3D-extension, we give a brief review of the previous works by quoting some of the descriptions in [4] in the following.

Let D be a connected compact subset of $\mathbb{R}^n$, and let $f : D \to \mathbb{R}$ be a function defined on D and continuous. As in [5], we propose the following principle for plotting the solutions to the equation $f = 0$. We shall use a *cell* as a terminology of the generalization of spot. The required properties for cells will be explained later.

Plotting principle 1. A *cell* has an n-dimensional volume.

2. A cell is *plotted* if it has a point in $\mathbb{R}^n$ in common with the solutions, and any non-plotted cell has no common point with the solutions.

Here, *plotted* means being displayed with a foreground color on the displaying device.

A plotting algorithm that satisfies the above principle is not easy to develop in general. The difficulty lies in constructing effective procedure to decide whether a solution exists or not in a given area. Required functionalities for an algorithms are outlined below.

- division of the domain of plotting area into a family of small sets of points, called *cells*.
- method to determine whether the given equation has isolated singular points, and if it does, the location of the cells in which the isolated singular points exist.
- method to determine whether a specified cell has any solution on its boundary.
- method to determine whether a specified cell contains any solutions which have no intersections on the boundary, i.e., entirely contained closed curve/surface.

2.2 Mathematical Description of Plotting

We describe the meaning of plotting with mathematical preciseness. On the above principle, plotting the solution of $f = 0$ is to decide for every cell defined on D whether or not it intersects the solution set $\{(x_1, x_2, \ldots, x_n) \in D \mid f(x_1, x_2, \ldots, x_n) = 0\}$. Let $\{C_j \mid j = 1, ..., m\}$ be a family of m subsets of D.

Definition 1 *We call $\{C_j \mid j = 1, ..., m\}$ a family of cells, or resolution, defined on D if every C_j is a connected closed subset of D, $D = \bigcup_{i=j}^{m} C_j$, and ${C_j}^{\mathrm{I}} \cap {C_k}^{\mathrm{I}} = \emptyset$ for $j \neq k$ where ${C_j}^{\mathrm{I}}$ and ${C_k}^{\mathrm{I}}$ denote sets of all interior points of C_j and C_k respectively.*

Our main concern in plotting the zeros of a function is to compute the following function called a *character*.

Definition 2 *We call a function $\chi : C_j \to \{0, 1\}$ a character of f with resolution $\{C_j\}$ defined on D if $\chi(C_j) = 0$ implies $f(x_1, x_2, \ldots, x_n) \neq 0$ for every point $(x_1, x_2, \ldots, x_n)$ in C_j.*

A character function χ guarantees that if $\chi(C_j) = 0$ then the given function f never vanish all over the cell C_j. It does not guarantee the existence of zeros of f in the cell C_j when $\chi(C_j) = 1$, however. The condition for character is not sufficiently tight to be used in practice for plotting the zeros of a function, in general. We propose a strong property of character as follows.

Definition 3 *A character χ of f with resolution $\{C_j\}$ on D is faithful if $\chi(C_j) = 1$ implies that there exists a point $(x_1, x_2, \ldots, x_n)$ in C_j such that $f(x_1, x_2, \ldots, x_n) = 0$.*

The faithful character provides desired functionality for exact plotting. For general bivariate functions, there are no known algorithms to compute faithful character, except for the one which applies to bivariate polynomials with rational coefficients [3].

We consider how we can implement a character function concretely. Hereinafter, we limit our concern to such cases that the family $\{C_j\}$ is composed of rectangular grid points, and we let C_k be $\{(x_1, ..., x_n) \mid a_{k,l} \leq x_l \leq b_{k,l},\ 1 \leq l \leq n,\ a_{k,l} \in \mathbb{Q}, b_{k,l} \in \mathbb{Q}\}$. Let $I_{k,l}$ denote an interval $[a_{k,l}, b_{k,l}]$.

Interval character. The function that checks if zero is contained in the interval value $f(I_{k,1}, ..., I_{k,n})$ satisfies the condition for character, but cannot be faithful in most cases. This function is called an *interval character*.

In general, it is quite difficult to give a concrete computing method for a character function, even without requiring the faithfulness. We let ourselves get pragmatic. We focus on the detection of zeros of a given polynomial, and introduce a notion of functions, called *weak character*, which are not necessarily character (in the sense of Definition 2) but satisfy the property of Definition 3. Also, in the following, we use the same term "character" referring to the functions used for the determination of cell plotting.

2.3 Plotting in 2D-space

We describe our basic ideas used for 2D cases, i.e., for plotting zeros of a bivariate polynomial $f(x, y)$. Without loss of generality, we assume f is square-free for simplicity.

According to its definition, for a specified cell, character must detect and must not miss the existence of any solution to the equation $f(x, y) = 0$ in the cell. Consider the case when a certain cell contains the solution of $f = 0$, and consider how to construct a function to determine the existence of any solutions. We consider what could likely to happen with the cells containing the zeros, and consider how it can be detected. In most cases, the curve would intersect with any of the edges of the cell, and the existence of any solution on a line segment can be easily checked, e.g., using the signs of the both endpoints or Sturm's theorem. These observations lead to the development of a series of character functions actually used in our implementation of `ifplot` for 2D cases as follows.

Sign (weak) character. Consider a certain cell C_k containing the solutions to $f = 0$. Then, it is likely that the signs of f are not all equal at four corners of C_k, i.e., $(a_{k,1}, a_{k,2})$, $(b_{k,1}, a_{k,2})$, $(a_{k,1}, b_{k,2})$ and $(b_{k,1}, b_{k,2})$. Our simplest character function, called sign (weak) character, computes and checks those signs.

Even if the curve $f = 0$ has any common point with any of the edges, this character may have a chance to miss their existence as in the case that the curve intersects with one edge an even number of times by taking their multiplicities into account. This type of omission can be avoided simply by computing Sturm's sequence.

Boundary (weak) character. If a certain cell C_k contains the zeros of f, then it is very likely that f has its zeros on any of the four edges. Boundary (weak) character is designed to detect the existence of the zeros on the edges of a cell. According to Sturm's theorem, we can determine the number of zeros of a univariate polynomial existing in a specified interval, and we shall use this theorem to detect the existence of zeros. More concretely, for each cell C_k, we check whether at least one of univariate polynomials $f(a_{k,1}, y)$, $f(b_{k,1}, y)$, $f(x, a_{k,2})$ and $f(x, b_{k,2})$ have zeros in the intervals $I_{k,1}$ for x and $I_{k,2}$ for y. In practice, we may recursively apply the bisection method and the detection by using Sturm's theorem to each of full line segments of the grid in the display area, until the interval of the bisected segments gets smaller than the resolution or non-existence of zeros in the segment is confirmed, as in [6].

Both of the above two characters will fail to detect the existence of zeros in a cell if all the zeros are of singular points or closed curves completely isolated inside the cell.

Implementation approach to a faithful character. We assume that f is not only square-free but irreducible. Singular points, if exist, can be obtained as solutions to the zero-dimensional system of polynomial equations $f = \partial f/\partial x = \partial f/\partial y = 0$. Also, on every closed curve, there must

exist finitely many points that satisfy the system of equations $\partial f/\partial x = 0$ or $\partial f/\partial y = 0$. Therefore, we can determine the locations of any cells containing isolated zeros by computing the solutions via the Gröbner basis of the system with sufficient accuracy that makes the precision of the locations smaller than the resolution.

This way of algebraic treatment plus the previous method for the detection on boundary can establish a faithful character.

3 Extension for Plotting in 3D-space

We now consider the case with three variables, i.e., to plot zeros of trivariate implicit functions $f(x,y,z) = 0$, in 3D space. Most of the arguments and the algorithms for 3D cases parallel those of 2D cases. Our overall strategies and the algorithms derived from the intermediate value theorem and Sturm's theorem are made so simple as the mathematical correctness and the accuracy of numeric calculation may rely on computer algebraic computation, and can be easily expanded to 3D cases. However, there arises a difficult problem in 3D cases, if there exists an isolated closed surface inside a voxel and we need to identify the precise location of the voxel. The rest of the section is devoted to investigating the algorithms using various characters, towards the 3D-extension, to some details from the practical point of view.

3.1 Character Functions for Graphs in 3D-space

The basic strategy of our algorithms is to check the existence of solutions against all cells or voxels in the drawing area, using character function, and the algorithms differ in how character determines the (non-)existence of solutions. We assume that all numeric calculations are done exactly or with sufficient accuracy.

3.2 Sign (Weak) Character

The simplest algorithm uses sign (weak) character. Based on the intermediate value theorem, sign (weak) character determines the existence of zero from the signs of $f(x,y,z)$ at the corners of a voxel. The procedure of the algorithm using sign (weak) character consists of the following steps.

1. fix grids defining a set of voxels and fix the coordinates of all grid points in a display area
2. evaluate $f(x,y,z)$ at all the grid points, and determine the zeroness or the signs of the values
3. if the values at all corner points of one voxel are non-zero and have the same sign, the voxel is regarded containing no zeros.

   This algorithm has such merits as

- required calculations can be done efficiently, and especially, can be performed in parallel, and
- it can be applied not only to algebraic equations but any continuous functions.

Notice that the algorithm is valid if the function to plot behaves gently, i.e., the value of function changes slowly in the range of the coordinates of a single voxel. On the other hand, the algorithm may miss the existence of zeros in such cases as the solution exists as an isolated singular point in a voxel, the solution curve/surface in a voxel is completely isolated or contained in a voxel, the solution curve/surface in a voxel is closed and its outward extension to the neighboring voxel, if any, circumvent the corner points, and so on.

3.3 Boundary Character

Boundary character is designed to detect the existence of solutions on the boundary edge or face. In 2D cases, Sturm's theorem can be used to detect the solution on the edge, a boundary of cell [3]. The boundary of each voxel consists of six faces of a cube. If a voxel contains any solution, the solution surface/curve is very likely to intersect with at least one of the six faces of the voxel cube. Every voxel face is a grid square of some grid plane. We consider the intersection curve(s)/point(s) of the surface/curve and each grid plane. On each grid plane, we consider the polynomial $f(x, y, z)$, i.e., the bivariate polynomial $f(x, y, z)$ obtained by the substitution corresponding to the plane, and we determine a set of squares of the plane containing its zeros. For this determination, we can use our method for 2D cases, and especially for completeness, we have only to use faithful character algorithm. Notice that faithful character for 2D cases can determine the existence or non-existence of solutions in a cell exactly. Therefore, boundary character for 3D cases can never miss the existence of a solution as long as the surface/curve has a point in common with any of the boundaries of voxel. The only case that this algorithm may miss is those when the graph of the solution is completely isolated inside a voxel. Treatment for those cases will be considered next.

3.4 Faithful Character

The role of character function is ideally the exact determination of the existence of solutions in a specified one of regions, cells or voxels, equally divided by grids. We want detect even those 3D cases when the curve/surface is isolated completely inside a single voxel. The isolated curve/surface must be closed, and as in 2D cases, the condition satisfied other than $f = 0$ differs depending on the shape of the curve/surface. If the isolated curve/surface is a single point, it is singular at the point and therefore, the following must be satisfied at the point:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial z} = 0. \tag{1}$$

In the case of closed surface, there always exists a point at which the tangent is parallel to the axis of x, y or z, say x afterwards, and then, the partial derivative of f in that direction must be equal to 0 at the point. Usually, it is expected that there exist only a finite number of such points, in which

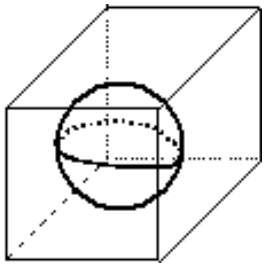


Fig. 1: Example: a closed surface completely isolated inside a voxel.

case the ideal $< f, \partial f/\partial x >$ is zero-dimensional and the solutions of the polynomial system of the ideal determine the voxel position containing the surface. The above discussion can be summarized as follows.

1. Taking the direction of x-axis as one direction, we compute the Gröbner basis of polynomials f and $\partial f/\partial x$.

2. If the ideal turned out to be zero-dimensional, solve the polynomial system to obtain its zeros.

This process can always detect a single point of the graph component isolated inside a single voxel, and is very likely to detect a closed surface inside a single voxel.

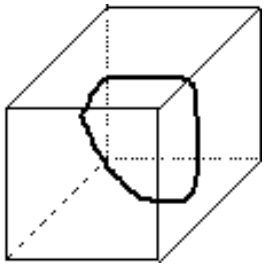


Fig. 2: Example difficult to detect: a closed curve completely isolated inside a voxel.

Very special is the case of closed curves, where the sign of f does not change around the curve, and therefore the condition (1) is satisfied on every point of the curve. This means that the ideal obtained by computing the Gröbner basis of f and some of the partial derivatives of f w.r.t. x, y and z is of positive dimension, and the method for detection mentioned above cannot be used. As a very simple example of this, we give the following equation

$$f(x,y,z) \quad = \quad (x^2+y^2+z^2-1)^2+x^2=0$$

representing the unit circle of the intersection of a sphere and a plane, and we assume the voxel has such a wide range that the circle is completely contained in a single voxel. The Gröbner basis will give a set of polynomials of the plane and the circle. In such rare and simple cases of closed curves, the location of the voxel containing a closed curve can be easily determined by some means, however, in general, more advanced algorithm as cylindrical algebraic decomposition(CAD) will be necessary. Further investigation is left for future study.

4 Empirical Study

As described in the previous section, it is difficult to guarantee the mathematical preciseness in plotting trivariate implicit functions, and even with our most precise algorithm using faithful character, there remains a chance to miss a closed curve complete isolated inside a single voxel. However, from a practical point of view, there may be a chance that such a tiny graphical component need not be drawn because it is not visible, and if being very casual is allowed, we may say that such a tiny component does not affect on the total appearance in most cases. We therefore, being very pragmatic, started to implement the 3D-extension of our algorithm using Risa/Asir for empirical study. In what follows, we report a part of our study, and show some examples.

Using Risa/Asir language The user language of Risa/Asir is equipped with a function to plot a given set of coordinate data, in order to facilitate experimentation of drawing method. Our first attempt was done by using this facility, and the algorithm of boundary character was implemented. Figures 3 and 4 are the sample output of this implementation. The defining polynomials used are as follows:

$$f(x,y,z) \quad = \quad (x^2+y^2+z^2)^2-10(x^2+y^2)+6z^2-10=0 \tag{2}$$

$$f(x,y,z) \quad = \quad (x^2+y^2+z^2-1)^2+(x-1/32)^2=0 \tag{3}$$

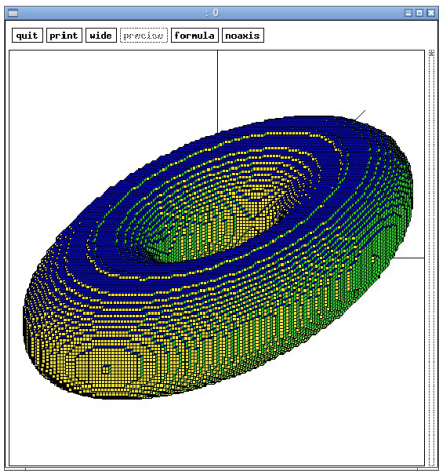


Fig. 3: Drawing example of Eq. (2)

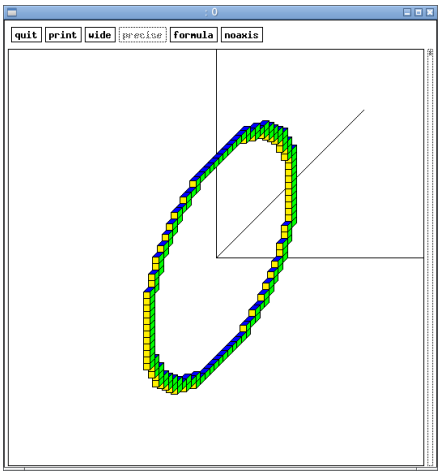


Fig. 4: Drawing example of Eq. (3)

Fig.3 represents the graph of Eq. (2) in $128 \times 128 \times 128$ grid, and Fig.4 does the graph of Eq. (3) in $64 \times 64 \times 64$ grid. These examples indicate that boundary character is sufficiently useful for this level of preciseness.

Using OpenGL for Drawing Graphical capability provided by the user language of Risa/Asir is quite limited; even standard capabilities such as for scaling or for rotation are not available. Also, the drawing speed is as fast as being tolerable for one shot drawing, but is quite slow by today's graphics standard. To remedy these defects, we developed a new drawing function which makes use of OpenGL. OpenGL provides rich functions for graphics manipulation, which will ease the development of standard graphics capabilities. We still use the implementation in Risa/Asir user language for calculations with character functions, and we simply replace the drawing part with our new implementation using OpenGL. Two types of character functions are implemented, boundary character and sign (weak) character. We use the boundary character in the following examples.

Figures 5, 6 and 7 are the examples drawn by our new implementation in $128 \times 128 \times 128$ grid, where the implicit functions used are defined respectively by the following equations:

$$f(x,y,z) = 16(x^2+y^2+z^2)^2 - 40(x^2+y^2) + 24z^2 - 9 = 0, \quad (4)$$

$$f(x,y,z) = (x^2+y^2+2z^2-1)^3 - x^2y^3 = 0, \quad (5)$$

$$f(x,y,z) = (x^2+y^2+2z^2-1)^2 + (x-1/32)^2 = 0. \quad (6)$$

With our new implementation, remarkable speedup has been attained, and also, we recognized again that boundary character is sufficient for practical use. Additionally, we should mention that the use of OpenGL, rather than the direct use of X as in the current implementation in Risa/Asir, makes it easier to port to a wide variety of platforms and to develop various graphical capabilities. The experience and the result of our empirical study strongly support that the hard problem of drawing 3D graphs of trivariate implicit functions can be treated with sufficient preciseness and reasonable amount of computing time, and that our 3D-extended method is useful in practice.

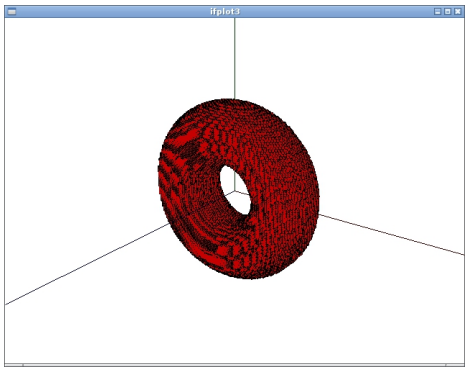


Fig. 5: Drawing example of Eq. (4) by OpenGL

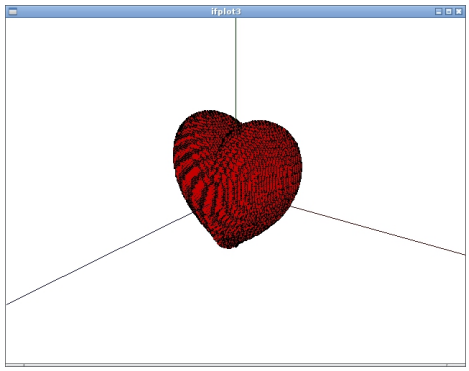


Fig. 6: Drawing example of Eq. (5) by OpenGL

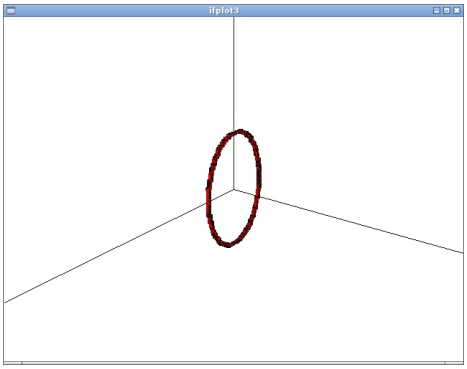


Fig. 7: Drawing example of Eq. (6) by OpenGL

5 Conclusion

In this paper, we investigated how we can draw 3D graphs of trivariate implicit functions exactly, and reported our empirical study with some experimental implementation. The methods used and implemented are the straightforward 3D-extension of the existing methods for 2D cases, each of which uses a different type of character function. Our voxel-based algorithm checks every voxel in the display area whether it contains the zero of the implicit function by character function. For each type of character function, we explained what situation of point/curve/surface may have a chance to be not detected. Even faithful character, our sharpest character, may miss a closed curve isolated inside a voxel, in which case the existence of such curve itself can be detected but its (voxel) location cannot be identified by our simple algorithm.

Finally, we would like to point out the similarity and the affinity of our voxel-based algorithm with 3D printers. The use of voxel as a smallest of plotting will be convenient if 3D printers are targeted as an output device, because a pixel on printing device can be treated as one voxel. Some typical type of 3D printers construct printed object by stacking layers of material, and the processing structure of repetition of the voxel-based algorithm is same as the layer-by-layer printing process and can be applied to the process. Notice that actual printing has such a problem as how to add supports for points or parts of object floating in the air.

As explained before, our current algorithm is still incomplete. Investigation of appropriate

methods for detecting a closed curve isolated inside a single voxel and development of mathematically correct and complete algorithm are left for further study.

References

[1] R. Fateman. Honest plotting, global extrema, and interval arithmetic. In Wang [7], pages 216–223.

[2] M. Noro and T. Takeshima. Risa/Asir — a computer algebra system. In Wang [7], pages 387–396.

[3] T. Saito. An extension of Sturm's theorem to two dimensions. *Proceedings of the Japan Academy, Ser. A Mathematical Sciences*, 73(1):18–19, 1997.

[4] T. Saito. *Displaying Zeros of Mathematical Equations*. PhD thesis, 2000. (in Japanese).

[5] T. Saito, Y. Kondoh, Y. Miyoshi, and T. Takeshima. Displaying real solution of mathematical equations. *Journal of JSSAC*, 6(2):2–21, 1998. (in Japanese).

[6] T. Saito, T. Takeshima, and T. Hilano. *Practice and Application of Gröbner Basis Computation*. University of Tokyo Press, 2003. (in Japanese).

[7] P. S. Wang, editor. *Proceedings of ISSAC '92*, Berkeley, CA, July 27–29 1992.

Communications of Jssac Vol. 2 2016
Publisher Japan Society for Symbolic and Algebraic Computation
Office zip 124–0011
Katsushika-ku Yotsugi 1–26–2 AlphaOmega Inc.