Minimum Converging Precision of the QR-Factorization Algorithm for Real Polynomial GCD

Pramook Khungurn
Massachusetts Institute of Technology
Cambridge, MA 02139-4307
pramook@mit.edu

Hiroshi Sekigawa
NTT Communication Science Laboratories
Nippon Telegraph and Telephone Corporation
Kanagawa, 243-0198, Japan
sekigawa@theory.brl.ntt.co.jp

Kiyoshi Shirayanagi
Department of Mathematical Sciences
Tokai University
Kanagawa, 259-1292 Japan
shirayan@ss.u-tokai.ac.jp

ABSTRACT

Shirayanagi and Sweedler proved that a large class of algorithms over the reals can be modified slightly so that they also work correctly on fixed-precision floating-point numbers. Their main theorem states that, for each input, there exists a precision, called the minimum converging precision (MCP), at and beyond which the modified "stabilized" algorithm follows the same sequence of instructions as that of the original "exact" algorithm. Bounding the MCP of any non-trivial and useful algorithm has remained an open problem.

This paper studies the MCP of an algorithm for finding the GCD of two univariate polynomials based on the QR-factorization. We show that the MCP is generally incomputable. Additionally, we derive a bound on the minimal precision at and beyond which the stabilized algorithm gives a polynomial with the same degree as that of the exact GCD, and another bound on the minimal precision at and beyond which the algorithm gives a polynomial with the same support as that of the exact GCD.

Categories and Subject Descriptors: I.1.2 [Computing Methodologies]: Symbolic and Algebraic Manipulation – *algorithms.*
General Terms: Algorithms, theory.
Keywords: Algebraic algorithm stabilization, polynomial greatest common divisor.

1. INTRODUCTION

Shirayanagi and Sweedler [19] showed that a large class of algorithms over the reals can be "mimicked" by algorithms over fixed-precision floating-point numbers in the following sense. Let $\mathcal{A}$ be an algorithm in the class. Then, there exists a family of algorithms $\{\mathcal{A}'_\tau\}_{\tau\in\mathbb{N}}$ such that:

1. $\mathcal{A}'_\tau$ is a slight modification of $\mathcal{A}$ that operates over floating-point numbers with precision τ.
2. For all input $\mathbf{x}$ in $\mathcal{A}$'s domain, $\lim_{\tau\to\infty} \mathcal{A}'_\tau(\mathrm{fl}_\tau(\mathbf{x})) = \mathcal{A}(\mathbf{x})$, where $\mathrm{fl}_\tau(\mathbf{x})$ denotes $\mathbf{x}$ approximated to precision τ.
3. For any input $\mathbf{x}$, there exists an integer $\Gamma_\mathbf{x}$, called the *minimum converging precision* (MCP), such that, for all $\tau \geq \Gamma_\mathbf{x}$, the computation of $\mathcal{A}'_\tau(\mathrm{fl}_\tau(\mathbf{x}))$ follows the same sequence of instructions as that of $\mathcal{A}(\mathbf{x})$.

(Following [19], we regard the family as a single algorithm that receives the precision as another input, and call this algorithm the *stabilized* algorithm.)

Their result implies that computation over the reals can generally be approximated by computation over floating-point numbers, and outlines a simple and general technique that transforms legacy exact algorithms to numerical algorithms that behaves *absolutely correctly* given that the input is accurate enough. The technique has been applied to many computer algebra algorithms including Buchburger's algorithm for Gröbner bases [18], Sturm's algorithm for counting real roots of polynomials [17], and Greville's algorithm for Moore-Penrose generalized inverses [14, 20]. Shirayanagi and Sweedler also suggested a number of potential applications such as (1) computation of approximate outputs, (2) computation of the "shape" of the output without exact computation, and (3) using the stabilized algorithm to aid the computation of the exact output [20].

Nevertheless, bounding the MCP, the precision where the stabilized algorithm starts behaving correctly, of any non-trivial and useful algorithm has been an open problem for at least 10 years [20]. We find the problem interesting as it provides a new perspective on the effect of roundoff errors on numerical computation. Bounding the MCP would also allow us to assess the practicality and effectiveness of the above potential applications.

This paper studies the MCP of algorithms for finding the greatest common divisor (GCD) of univariate polynomial with real coefficients. We show that, for a class of the algorithms solving the problem, the MCP is incomputable if the coefficients of the input polynomials can be any computable numbers. This result illustrates the hardness of the problem.

Additionally, we examine a GCD algorithm based on QR-factorization, and derive asymptotic bounds specific to the algorithm on two related numbers: the *minimal correct degree precision* (MCDP) and the *minimal same support precision* (MSSP). The MCDP is the minimal precision at and beyond which the stabilized algorithm gives a polynomial

ISSAC'07, July 29–August 1, 2007, Waterloo, Ontario, Canada.

with the same degree as that of the exact GCD. The MSSP is the minimal precision at and beyond with the algorithm gives a polynomial with the same support, i.e., non-zero terms, as that of the exact GCD.

For the MCDP, we show it is of order

$$O\big(d(\log d + \log M) - \sum_{i=1}^{r} \log |R[i,i]|\big),$$

where d is the sum of the degrees of the input polynomials, M is the largest absolute value of the coefficients, r is the rank of the Sylvester matrix of the input polynomials, and R is the upper triangular matrix in the QR-factorization of the Sylvester matrix. For the MSSP, we show that it is of order

$$O\big(d(\log d + \log M) - \sum_{i=1}^{r} \log |R[i,i]| - \log |\mu|\big),$$

where μ is the smallest non-zero coefficient of the (monic) GCD of the input polynomials. These bounds show that, for some algorithm, it is possible to bound precisions at which the algorithm gives "correct" output without bounding the MCP, a task we consider more difficult. They also suggest that the impact of roundoff errors on GCD computation can be significant.

Lastly, we show that we can obtain a simpler bound if we restrict the domain of the coefficients of to $\mathbb{Z}[\xi]$, where ξ is an algebraic integer of degree n. We prove an $O(dn(\log d + \log N + n \log C))$ bound on the MCDP, where N is the largest absolute value of the integers in the coefficients of the input polynomials, and C is the largest absolute value of the coefficients of ξ's minimal polynomial. This bound implies an $\tilde{O}(d^4 n(\log d + \log N) + d^3 n^4 \log^2 C)$ algorithm that can compute the degree of the GCD of polynomials with algebraic integer coefficients.

The rest of the paper is organized as follows. Section 2 discusses previous works on real polynomial GCDs and compare and contrast our work to them. Section 3 defines notations and gives backgrounds on the Shirayanagi–Sweedler stabilization technique. Section 4 proves that the MCP is incomputable. Section 5 bounds the MCDP and the MSSP of the QR-factorization algorithm. Section 6 derives the bound on the MCDP if input coefficients are members of $\mathbb{Z}[\xi]$. Section 7 concludes.

2. RELATED WORK

GCD algorithms for univariate real polynomials have been extensively studied. Since computation over the reals is unrealizable on conventional computers, researchers instead gave algorithms that compute various "approximate" GCDs.

Schönhage [16] introduced *quasi-GCDs* and a fast algorithm to compute them assuming that input coefficients can be approximated to any precision. Most other researchers assumed that the coefficients are inexact and are provided *as is*. Sasaki and Noda [15], for example, extended the Euclidean algorithm so that it produces a polynomial that, when divided from the input polynomials, produces remainders with small ∞-norm.

A popular notion of approximate GCD is the *ε-GCD*. Polynomial $\mathbf{G}$ is said to be the ε-GCD of $\mathbf{P}$ and $\mathbf{Q}$ if it is the polynomial such that (1) for some norm $\|\cdot\|_*$, there exist $\hat{\mathbf{P}}$ and $\hat{\mathbf{Q}}$ with $\|\mathbf{P} - \hat{\mathbf{P}}\|_*, \|\mathbf{Q} - \hat{\mathbf{Q}}\|_* < \varepsilon$ such that $\mathbf{G} = \gcd(\hat{\mathbf{P}}, \hat{\mathbf{Q}})$, and (2) $\mathbf{G}$ has the highest degree. Hribernig and Stetter [9] sought the ε-GCD where the norm is the 1-norm. Karmarkar and Lakshman [10] gave an algorithm that finds pairs of polynomials close to the input polynomials with non-trivial GCD, and apply the algorithm to find the 2-norm ε-GCD such that $\|\mathbf{P} - \hat{\mathbf{P}}\|^2 + \|\mathbf{Q} - \hat{\mathbf{Q}}\|^2$ is minimized. Emiris, Galligo, and Lombardi [4, 5] gave not only an algorithm to compute the ε-GCD, but also conditions on the Sylvester matrix of the input polynomial that guarantee that the degree of the ε-GCD is equal to a particular value.

Additionally, there are many results on numerical GCD algorithms. Corless et al. [3] described an algorithm based on the SVD. Zarowski, Ma, and Fairman [22] gave an algorithm based on QR factorization, which is later improved by Corless, Watt, and Zhi [2]. These algorithms are numerically stable and have good backward error bounds. Zhi [23] also proposed a fast algorithm based on displacement structure of the Sylvester matrix, but its stability is unknown.

In general, the above studies focus on *short-term behaviors* of algorithms as they provide some guarantees on the outputs regardless of how precise the inputs are. Also, they mainly deal with input errors while roundoff errors are often not addressed. On the other hand, our study focuses on *asymptotic behaviors* as we seek the machine precision required to guarantee that the stabilized algorithm behave like the exact algorithm. We exclusively deal with roundoff errors, and, like Schönhage, assume that inputs are arbitrarily precise.

3. PRELIMINARIES

Scalars are denoted by lowercase letters, vectors by boldfaced lowercase letters, and matrices by uppercase letters. We let $\mathbf{a}[k]$ denote the kth component of vector $\mathbf{a}$, and $A[i,j]$ denote the (i,j)-element of matrix A. We let $\mathbf{a}[i:j]$ denote the vector $(\mathbf{a}[i], \mathbf{a}[i+1], \ldots, \mathbf{a}[j])$. Symbols such as $A[i_1 : i_2, j_1 : j_2]$, $A[i_1 : i_2, j]$, and $A[i, j_1 : j_2]$ hold similar meanings for matrices. Moreover, we let $A[*, j]$ denote the jth column of A, and $A[i, *]$ denote the ith row of A.

Let $\|\mathbf{a}\|$ denote the ℓ_2-norm of $\mathbf{a}$, and let $|\mathbf{a}|$ denote the vector whose entries are the absolute values of the corresponding entries of $\mathbf{a}$. We define $|A|$ similarly.

Throughout this paper, every polynomial is a real polynomial, and is denoted by a boldfaced uppercase letter: for examples, $\mathbf{P}$, $\mathbf{Q}$, and $\mathbf{R}$. The degree of polynomial $\mathbf{P}$ is denoted by $\deg(\mathbf{P})$, the coefficient of x^r in $\mathbf{P}$ by $\mathbf{P}[r]$. The *support* of polynomial $\mathbf{P}$ is the set $\{r : \mathbf{P}[r] \neq 0\}$. For example, the support of $x^5 + x + 1$ is $\{0, 1, 5\}$.

3.1 Floating-point Numbers

A floating-point number a with precision τ is an ordered pair $(\mathfrak{M}(a), \mathfrak{e}(a))$, where $\mathfrak{M}(a)$ is an integer with τ digits in base 2, and $\mathfrak{e}(a)$ is another integer. The real value of a is defined to be $a = \mathfrak{M}(a) \times 2^{\mathfrak{e}(a)-\tau}$. We require that the leftmost digit of $\mathfrak{M}(a)$ is 1 unless $a = 0$. As a convention, τ denotes the precision of every floating-point number.[1]

Let x be a real number. We define the following functions:

- Let $\mathrm{fl}_\tau(x)$ be the floating-point number with precision τ closest to x.

[1]Our definition of floating-point numbers is very close to the IEEE floating-point standard, except for the absence of the sign bit, and the requirement that the leftmost bit of the mantissa is not zero.

- Let $\mathrm{up}_\tau(x)$ be x rounded up to precision τ away from zero.
- For $x \neq 0$, let $\mathfrak{e}(x)$ be the integer such that $x = y \times 2^{\mathfrak{e}(x)}$ for some y such that $0.5 \leq |y| < 1$.
- Let $\epsilon_\tau(x)$ be the value $2^{\mathfrak{e}(x)-1-\tau}$. This number is one half of the unit in the last place (ULP) of x.

3.2 Bracket Coefficients

The Shirayanagi–Sweedler stabilization technique relies on computing with intervals whose endpoints are floating-point numbers. We refer the reader to [1] for a more detailed treatment of interval computation. Following [19], we represent intervals by objects called bracket coefficients.

A *bracket coefficient* $[\![x]\!]$ is an ordered pair of floating-point numbers $(\langle x\rangle, \lfloor x \rceil)$ with $\lfloor x \rceil \geq 0$. It represents the closed interval $[\ \langle x\rangle - \lfloor x \rceil, \langle x\rangle + \lfloor x \rceil\ ]$. We refer to $\langle x\rangle$ as the *approximation term*, and $\lfloor x \rceil$ the *error term*. $[\![x]\!]$ is said to *approximate* a real number x if x falls within its interval. We denote this fact by $[\![x]\!] = x$. As a convention, $[\![x]\!]$ denotes a bracket coefficient that approximates the real number x. The *floating-point bracket approximation* to a real number x, denoted by $[\![x]\!]_\tau$, is the bracket coefficient $(\mathrm{fl}_\tau(x), \epsilon_\tau(\mathrm{fl}_\tau(x)))$.

Bracket coefficient $[\![x]\!]$ is said to be *equal* to zero if $|\langle x\rangle| \leq \lfloor x \rceil$, or, in other words, if its interval contains zero. $[\![x]\!]$ is said to be *greater than* zero if it is not equal to zero, and $\langle x\rangle > 0$. Similarly, $[\![x]\!]$ is said to be *less than* zero if it is not equal to zero, and $\langle x\rangle < 0$. This trichotomy law of bracket coefficients is based on the zero-rewriting, an idea crucial to the Shirayanagi-Sweedler stabilization technique. Arithmetic operations on bracket coefficients are defined as follows:

1. Addition: $[\![s]\!] = [\![a]\!] + [\![b]\!]$ if
   (a) $\langle s\rangle = \mathrm{fl}_\tau(\langle a\rangle + \langle b\rangle)$, and
   (b) $\lfloor s \rceil = \mathrm{up}_\tau(\lfloor a \rceil + \lfloor b \rceil + \epsilon_\tau(\langle s\rangle))$.
2. Subtraction: $[\![d]\!] = [\![a]\!] - [\![b]\!]$ if
   (a) $\langle d\rangle = \mathrm{fl}_\tau(\langle a\rangle - \langle b\rangle)$, and
   (b) $\lfloor d \rceil = \mathrm{up}_\tau(\lfloor a \rceil + \lfloor b \rceil + \epsilon_\tau(\langle d\rangle))$.
3. Multiplication: $[\![p]\!] = [\![a]\!][\![b]\!]$ if
   (a) $\langle p\rangle = \mathrm{fl}_\tau(\langle a\rangle\langle b\rangle)$, and
   (b) $\lfloor p \rceil = \mathrm{up}_\tau(\lfloor a \rceil \lfloor b \rceil + |\langle a\rangle| \lfloor b \rceil + |\langle b\rangle| \lfloor a \rceil + \epsilon_\tau(\langle p\rangle))$.
4. Inverse: $[\![i]\!] = [\![a]\!]^{-1}$ or $[\![i]\!] = 1/[\![a]\!]$ if
   (a) $\langle i\rangle = \mathrm{fl}_\tau(1/\langle a\rangle)$, and
   (b) $\lfloor i \rceil = \mathrm{up}_\tau\Big(\frac{\lfloor a \rceil}{|\langle a\rangle|(|\langle a\rangle| - \lfloor a \rceil)} + \epsilon_\tau(\langle i\rangle)\Big)$.
5. Square root: for $[\![a]\!] > 0$, $[\![r]\!] = [\![a]\!]^{1/2}$ if
   (a) $\langle r\rangle = \mathrm{fl}_\tau(\langle a\rangle^{1/2})$, and
   (b) $\lfloor r \rceil = \mathrm{up}_\tau\Big(\frac{\langle a\rangle^{1/2}}{2}\Big(\frac{\lfloor a \rceil}{\langle a\rangle - \lfloor a \rceil}\Big) + \epsilon_\tau(\langle r\rangle)\Big)$.

We note that, for any operation $\star$ above, it holds that $[\![a]\!] \star [\![b]\!] \cong a \star b$.

Lastly, for $\mathbf{a} = (a_1, \ldots, a_n)$, we let $[\![\mathbf{a}]\!] = ([\![a_1]\!], \ldots, [\![a_n]\!])$ be a vector of bracket coefficients whose entries approximate the corresponding entries of $\mathbf{a}$, and say that $[\![\mathbf{a}]\!]$ approximates $\mathbf{a}$. Also, we define $\langle \mathbf{a}\rangle = (\langle a_1\rangle, \ldots, \langle a_n\rangle)$ and $\lfloor \mathbf{a} \rceil = (\lfloor a_1 \rceil, \ldots, \lfloor a_n \rceil)$. These notations extend to matrices in a straightforward way.

3.3 Model of Computation

We are interested in two types of machines: EXACT, and $\mathsf{BRACKET}_\tau$. Both machines have a countably infinite number of registers $x_1, x_2, \ldots$. Each register of EXACT holds a real number, and each register of $\mathsf{BRACKET}_\tau$ a bracket coefficient with floating-point numbers of precision τ. Arithmetic operations in EXACT are the canonical operations on real numbers, and those in $\mathsf{BRACKET}_\tau$ are as defined in the last section.

An *algebraic algorithm* is a finite sequence of instructions which can be any one of the following forms:

1. $x_i \leftarrow x_j + x_k$
2. $x_i \leftarrow x_j - x_k$
3. $x_i \leftarrow x_j \times x_k$
4. $x_i \leftarrow x_j^{-1}$
5. $x_i \leftarrow \sqrt{x_j}$
6. $x_i \leftarrow c$ for some $c \in \mathbb{Z}$ ($[\![c]\!]_\tau$ in $\mathsf{BRACKET}_\tau$)
7. **goto line** ℓ
8. **if** $x_i = 0$ **goto line** ℓ
9. **if** $x_i < 0$ **goto line** ℓ
10. **halt**

Note that any algebraic algorithm can run on both EXACT and $\mathsf{BRACKET}_\tau$. The following is an example of an algebraic algorithm:

```
1  x1 ← x1 − x2
2  if x1 = 0 goto line 5
3  x3 ← 0
4  goto line 6
5  x3 ← 1
6  halt
```

We regard the state of the vector $\mathbf{x} = (x_1, x_2, \ldots)$ before the algorithm runs as the input of the algorithm, and the state of $\mathbf{x}$ after the algorithm terminates as the output of the algorithm. Thus, an algebraic algorithm is a function on an infinite dimensional Euclidean space.

An *execution path* of an algebraic algorithm is the order of instructions the algorithm follows when running on a particular input. For example, if $x_1 = x_2$, then the execution path of the above algorithm is (1, 2, 5, 6), but if $x_1 \neq x_2$, the execution path is (1, 2, 3, 4, 6).

To provide distinction between registers of EXACT and $\mathsf{BRACKET}_\tau$, we shall let $x_1, x_2, \ldots$ be registers of EXACT, and $[\![x_1]\!], [\![x_2]\!], \ldots$ be registers of $\mathsf{BRACKET}_\tau$. This choice is made to convey the fact that $[\![x_i]\!]$ approximates x_i when the two machines run the same algorithm on the "same" input.

3.4 Shirayanagi–Sweedler Stabilization

Let $\mathcal{A}$ be an algebraic algorithm, and let $\mathbf{x} = (x_1, x_2, \ldots)$ be a countably infinite dimensional real vector. We let $\mathcal{A}(\mathbf{x})$ denote output (i.e., the state of the registers after $\mathcal{A}$ terminates) of running $\mathcal{A}$ in EXACT with input $\mathbf{x}$, and let $\mathcal{A}([\![x]\!]_\tau)$ denote the output of running $\mathcal{A}$ on $\mathsf{BRACKET}_\tau$ with input $[\![\mathbf{x}]\!]_\tau = ([\![x_1]\!]_\tau, [\![x_2]\!]_\tau, \ldots)$.

Shirayanagi's and Sweedler's main result can be restated formally as follows:

PROPOSITION 1. *Let $\mathcal{A}$ be an algebraic algorithm. Let $\mathbf{y} = (y_1, y_2, \ldots) = \mathcal{A}(\mathbf{x})$ and $[\![\mathbf{y}_\tau]\!] = ([\![y_{\tau,1}]\!], [\![y_{\tau,2}]\!], \ldots) = \mathcal{A}([\![x]\!]_\tau)$. Then, for all input $\mathbf{x}$ in $\mathcal{A}$'s domain:*

1. $\lim_{\tau\to\infty}\langle \mathbf{y}_\tau\rangle = \mathbf{y}$.
2. *There exists a positive integer* $\Gamma_{\mathbf{x}}$ *such that, for all* $\tau \geq \Gamma_{\mathbf{x}}$*, the execution path of* $\mathcal{A}$ *on* $\mathsf{BRACKET}_\tau$ *with input* $[\![\mathbf{x}]\!]_\tau$ *is the same as the execution path of* $\mathcal{A}$ *on* EXACT *with input* $\mathbf{x}$.

We say that $\Gamma_{\mathbf{x}}$ is a *converging precision* of $\mathcal{A}$ on $\mathbf{x}$. The smallest $\Gamma_{\mathbf{x}}$ is the *minimum converging precision* (MCP). In the context of algorithm $\mathcal{A}$ that outputs a polynomial, Shirayanagi and Sweedler also show that, for every input $\mathbf{x}$, there exists an integer $\Gamma^{SS}_{\mathbf{x}}$ such that, if $\tau \geq \Gamma^{SS}_{\mathbf{x}}$, then $\mathcal{A}([\![\mathbf{x}]\!]_\tau)$ has the same support as that of $\mathcal{A}(\mathbf{x})$. We call the least $\Gamma^{SS}_{\mathbf{x}}$ the *minimum same support precision* (MSSP) of $\mathcal{A}$ on $\mathbf{x}$. Also, define the *minimum correct degree precision* (MCDP) of $\mathcal{A}$ on $\mathbf{x}$ to be the least integer Γ such that, if $\tau \geq \Gamma$, then the degree of $\mathcal{A}([\![\mathbf{x}]\!]_\tau)$ is the same as that of $\mathcal{A}(\mathbf{x})$. Clearly, the MCDP does not exceed the MSSP, and therefore exists.

The result implies that the stabilized algorithm $\{\mathcal{A}'_\tau\}_{\tau\in\mathbb{N}}$ mentioned in the introduction can be constructed as follows. For input $\mathbf{x}$ and parameter τ, we compute its floating-point brakcet approximation $[\![\mathbf{x}]\!]_\tau$ and then run $\mathcal{A}$ on input $[\![\mathbf{x}]\!]_\tau$ in $\mathsf{BRACKET}_\tau$. Let $[\![\mathbf{y}_\tau]\!]$ be the corresponding output. Then, we return $\langle \mathbf{y}_\tau\rangle$ to the user. Note that, if we rewrite $\langle y_{\tau,i}\rangle$ to zero for every $[\![y_{\tau,i}]\!] = 0$, then the output polynomial has the same support as the exact output polynomial if $\tau > MSSP$. As such, we perform this zero-rewriting by default.

4. INCOMPUTABILITY RESULT

In this section, we show that the MCP of a large class of GCD algorithms is incomputable provided that the coefficients of the input polynomials are allowed to be any computable numbers. Recall that a *computable number* x is a real number such that there exists a Turing machine that produces the nth digit of the binary (or decimal) expansion of x, given any positive integer n. Equivalently, x is a computable number if there is a Turing machine that, when given an integer τ, outputs $\mathrm{fl}_\tau(x)$.

DEFINITION 1. *A GCD-finding algorithm* $\mathcal{A}$ *is said to be* degree-aware *if, for any polynomials* $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, *and* $\mathbf{D}$ *such that* $\deg(\gcd(\mathbf{A},\mathbf{B})) \neq \deg(\gcd(\mathbf{C},\mathbf{D}))$*, it is true that* $\mathcal{A}(\mathbf{A},\mathbf{B})$ *follows different execution path from that of* $\mathcal{A}(\mathbf{C},\mathbf{D})$.

THEOREM 1. *Let* $\mathcal{A}$ *be any degree-aware GCD-finding algebraic algorithm. There exists no Turing machine* $\mathcal{D}_\mathcal{A}$ *such that, for every pair of polynomials* $\mathbf{A}$ *and* $\mathbf{B}$ *whose coefficients are computable numbers,* $\mathcal{D}_\mathcal{A}(\mathbf{A},\mathbf{B})$ *gives the MCP of* $\mathcal{A}$ *on* $\mathbf{A}$ *and* $\mathbf{B}$.

PROOF. Suppose by way of contradiction that such $\mathcal{D}_\mathcal{A}$ exists. Let Γ be the MCP of $\mathcal{A}$ when the input polynomials are $x+1$ and $x+1$. For any Turing machine $\mathcal{M}$ and any input s to $\mathcal{M}$, we define another Turing machine $\mathcal{X}_{\mathcal{M},s}$ as follows:

$\mathcal{X}_{\mathcal{M},s}(\tau)$
1 If $\tau \leq \Gamma$
2 **then** Output $\mathrm{fl}_\tau(1)$.
3 **else** Run $\mathcal{M}$ on s for $\tau - \Gamma$ steps.
4 **if** $\mathcal{M}$ does not terminate by $\tau - \Gamma$ steps
5 **then** Output $\mathrm{fl}_\tau(1)$.
6 **else** Let k be the step at which $\mathcal{M}$ terminates.
7 Output $\mathrm{fl}_\tau(1 + 2^{-\Gamma-k})$.

Note that, if $\mathcal{M}$ on input s does not terminate, then $\mathcal{X}_{\mathcal{M},s}$ computes the number 1. Otherwise, it computes the number $1 + 2^{-\Gamma-k}$ where k is the number of steps $\mathcal{M}$ runs when fed with input s. Next, we define Turing machine $\mathcal{E}$ as follows:

$\mathcal{E}(\mathcal{M}, s)$
1 Construct $\mathcal{X}_{\mathcal{M},s}$.
2 **if** $\mathcal{D}_\mathcal{A}(x+1, x+\mathcal{X}_{\mathcal{M},s}) = \Gamma$
3 **then reject**
4 **else accept**

We claim that $\mathcal{E}$ rejects if and only if $\mathcal{M}$ does not halt on s. If $\mathcal{M}$ does not halt on s, then $\mathcal{X}_{\mathcal{M},s}$ computes the number 1. Therefore, $\mathcal{D}_\mathcal{A}(x+1, x+\mathcal{X}_{\mathcal{M},s})$ returns Γ, and $\mathcal{E}$ rejects.

If $\mathcal{M}$ halts on s, then $\mathcal{X}_{\mathcal{M},s}$ computes a number different from 1, and $\deg(\gcd(x+1, x+\mathcal{X}_{\mathcal{M},s})) = 0 \neq \deg(\gcd(x+1, x+1))$. As a result, the execution path of the exact computation $\mathcal{A}(x+1, x+1)$ must be different from that of $\mathcal{A}(x+1, x+\mathcal{X}_{\mathcal{M},s})$ because $\mathcal{A}$ is degree-aware. Let $\mathcal{A}'$ be $\mathcal{A}$'s stabilized version. Since $\mathrm{fl}_\Gamma(\mathcal{X}_{\mathcal{M},s}) = \mathrm{fl}_\Gamma(1)$, we have that the computation of $\mathcal{A}'_\Gamma(\mathrm{fl}_\Gamma(x+1), \mathrm{fl}_\Gamma(x+\mathcal{X}_{\mathcal{M},s}))$ must follow exactly the execution path of $\mathcal{A}(x+1, x+1)$ by the definition of Γ. Since this execution path is different from that of $\mathcal{A}(x+1, x+\mathcal{X}_{\mathcal{M},s})$, it must be the case that the MCP of $\mathcal{A}$ on $x+1$ and $x+\mathcal{X}_{\mathcal{M},s}$ is greater than Γ. Hence, $\mathcal{E}$ accepts.

Because $\mathcal{E}$ is a Turing machine that solves the halting problem, we arrive at a contradiction. □

5. QR-FACTORIZATION ALGORITHM

In this section, we describe the QR-factorization algorithm and derive bounds on its MCDP and MSSP.

5.1 Algorithm Description

The algorithm first forms the Sylvester matrix of the input polynomials, and then performs QR-factorization. The last non-zero row of R gives the coefficients of the GCD. A proof of this algorithm can be found in [2].

Our QR-factorization algorithm is a slight variation of the Householder QR algorithm given in [7]. Unlike the algorithm in [7], our HOUSEHOLDER effectively produces a permutation matrix P that swaps Row j with a row under it, and a Householder reflection Q that zeroes out elements under the main diagonal in Column j of PA. Moreover, elements on the main diagonal of the resulting triangular matrix R can either be positive or negative, unlike the algorithm in [7], which produces R such that all of its diagonal entries are nonnegative. Clearly, our version of HOUSEHOLDER still gives a correct QR-factorization although the factorization might be different from those given by other implementations. The pseudocode of the algorithm is given in Figure 1.

Finally, the pseudocode of the algorithm for finding GCD is given in Figure 2.

Clearly, the QR-factorization algorithm is an algebraic algorithm. It is degree-aware because the degree of the GCD determines the number of iterations it has to go through the loop on Line 4 to 6 of QR-GCD.

5.2 Proof Strategy

In general, a worst-case upper bound on the MCP of any algebraic algorithm can be determined by the following three-step process.

```
HOUSEHOLDER-QR(A)
    ▷ A is a matrix of dimension d × d.
 1  A_1 ← A
 2  for j ← 1 to d
 3       do (v_j, α_j, β_j) ← HOUSEHOLDER(A_j, j)
 4          A_{j+1} ← A_j
 5          if α_j = 0
 6             then continue
 7          for i ← j to d
 8               do c ← v_j · A[j : d, i]
 9                  for k ← j to d
10                       do A_{j+1}[k, i] ← A_j[k, i]−
                                2β_j c v_j[k − j + 1]
11  return A_{d+1}

HOUSEHOLDER(A_j, j)
    ▷ A_j is a matrix of dimension d × d, and 1 ≤ j ≤ d.
 1  σ_1 ← Σ_{i=j}^{d} A_j[j, i]^2
 2  if σ_1 = 0
 3     then α_j ← 0
 4     else  for k ← j to d
 5                do if A_j[k, j] ≠ 0
 6                      then break
 7           for i ← j to d
 8                do Swap A_j[j, i] with A_j[k, i].
 9           μ ← √σ_1
10           v_j ← A_j[j : d, j]
11           if A_j[j, j] < 0
12              then v_j[1] ← A_j[j, j] − μ
13              else  v_j[1] ← A_j[j, j] + μ
14           σ_2 ← ‖v_j‖^2
15           β_j ← 1/σ_2
16           α_j ← 1
17  return (v_j, α_j, β_j)
```

Figure 1: Pseudocode of the QR-factorization.

1. Find an upper bound on how large the error term of each register can become as the stabilized algorithm evolves, assuming that the stabilized algorithm follows the exact execution path.
2. Find a lower bound on the size of the smallest non-zero numerical value that arises during the evolution of the exact algorithm.
3. Find a precision at and beyond which the upper bound in Step 1 never exceeds the lower bound in Step 2. This precision is a converging precision, and therefore is greater than or equal to the MCP.

The rationale behind this method is as follows:

1. The first place that the execution path of the stabilized algorithm differs from the execution path of the exact algorithm must be at a conditional statement.
2. In our model of computation, a conditional statement involves comparing a bracket coefficient to zero.
3. Comparison can give a wrong answer only when the bracket coefficient actually approximates a non-zero real number, but its error term is so large that the bracket coefficient is equal to zero [18].

```
QR-GCD(A, B)
1  A ← Syl(A, B)
2  R ← HOUSEHOLDER-QR(A)
3  d, i ← deg(A) + deg(B)
4  while R[i, i] = 0
5        do i ← i − 1
6  return Σ_{j=i}^{d−i+1} R[i, j] x^{d−1−j}
```

Figure 2: Pseudocode of the main algorithm.

4. Therefore, if the precision is large enough so that no error term is larger than the smallest numerical value, then any bracket coefficient that approximates a non-zero value cannot be equivalent to zero. As a result, no comparison gives a wrong result, and the stabilized algorithm follows the same execution path as the exact algorithm.

Bounding the MCDP is easier because we only have to make sure that the stabilized algorithm branches correctly at *some* conditional statements. For the QR-factorization algorithm, we shall see that ensuring correct branching only at Line 4 of QR-GCD and Line 2 of HOUSEHOLDER is enough. As a result, we only need to concern ourselves with the lower bounds on the sizes of the quantities relevant to the two conditional statements: the diagonals of R. Bounding the MSSP additionally requires that no error term exceeds the smallest absolute value of the coefficients of the exact GCD.

5.3 Error Analysis

For Step 1 of the three-step process, we trace the execution of the QR-factorization and derive upper bounds on the error terms of entries of the matrix A_j in HOUSEHOLDER-QR as j increases from 1 to d, using the definitions of arithmetic operations of bracket coefficients in Section 3.2. Our analysis is a forward error analysis as opposed to backward error analyses in [21] and [8]. We remark that one might able to use the backword errors to bound the forward errors, but we have not explored the option and chose the direct approach. Since the derivation is technical and tedious, we state the bounds here without proof and refer the reader to [11] of the full proofs.

The following lemma gives an upper bound on the error terms of HOUSEHOLDER's output: (As a convention, if a is a variable in the exact algorithm, then $[\![a]\!]$ denotes the corresponding bracket coefficient in the stabilized version, $\langle a \rangle$ its approximation term, and $\lfloor a \rceil$ its error term.)

LEMMA 1. *For any j, let $\mathbf{x}_j$ denote the vector $A_j[j : m, j]$. If $\lfloor A_j[i, j] \rceil \le \varepsilon \|\mathbf{x}_j\|$ for all $j \le i \le d$ and for some constant ε such that $2^{-\tau} < \varepsilon < 1/\Theta(d^2)$, then, after* HOUSEHOLDER *runs to completion, the followings are true:*

(i) The stabilized algorithm branched correctly at Line 2,
(ii) $\lfloor \mathbf{v}_j[i] \rceil = O(d\varepsilon\|\mathbf{v}_j\|)$ for all $1 \le i \le d - j + 1$, and
(iii) $\lfloor \beta_j \rceil = O(d^2\varepsilon\beta_j)$.

We would like to comment on (1) why (i) is true, and (2) why we include the row swap in our algorithm. If $\|\mathbf{x}_j\| = 0$, then $[\![\sigma_1]\!]$ must approximate 0, and the stabilized algorithm branched correctly. If $\|\mathbf{x}_j\| \neq 0$, it is easy (but rather tedious) to show that $\lfloor \sigma_1 \rceil = O(d\varepsilon\|\mathbf{x}_j\|^2) < \frac{O(d)}{\Theta(d^2)}\|\mathbf{x}_j\|^2 <$

$\frac{1}{2}\|\mathbf{x}_j\|^2$. So, $|\langle\sigma_1\rangle| \geq |\sigma_1| - \lfloor\sigma_1\rceil > \frac{1}{2}\|\mathbf{x}_j\|^2 > \lfloor\sigma_1\rceil$. Hence, $[\![\sigma_1]\!] \neq 0$, and the stabilized algorithm also branched correctly in this case.

As for the row swap, we first note that the comparison in Line 11 of HOUSEHOLDER serves to prevent catastrophic cancellation in the computation of $\mathbf{v}_j[1]$. (See [7] for details.) However, it also makes the error analysis much more difficult. Imagine removing Line 4 to Line 8 from HOUSEHOLDER, and that $[\![A_j[j,j]]\!]$ is equal to zero although $A_j[j,j]$ is actually negative. The stabilized algorithm would branch to Line 13 and compute $[\![\mathbf{v}_j[1]]\!] \leftarrow [\![A_j[j,j]]\!] + [\![\mu]\!]$, but the exact algorithm would branch to Line 12 and compute $\mathbf{v}_j[1] \leftarrow A_j[j,j] - \mu$. This makes us unable to bound $\lfloor\mathbf{v}_j[1]\rceil$ correctly in terms of $\mathbf{v}_j[1]$. We can actually require that the precision to be high enough so that this mismatch does not occur, but this requirement would lead to a worst bound on the MCDP as $A_j[j,j]$ can be very small relative to $\|\mathbf{v}_j\|$, the length of the output vector. The row swap makes sure that $[\![A[j,j]]\!] \neq 0$ before Line 11 to prevent the mismatch, and thus sharpens the bound.

The next lemma gives an upper bound on errors incurred by HOUSEHOLDER-QR:

LEMMA 2. *Let $M = \max\{1, \max_{k,\ell}\{|A_1[k,\ell]|\}\}$. Let γ be such that and $\lfloor A_1[k,\ell]\rceil \leq \gamma$ for any k,ℓ. Let j be an integer such that $1 \leq j \leq m+1$, and $\|\mathbf{x}_j\| \neq 0$. Let ε be such that $2^{-\tau} < \varepsilon < 1/\Theta(m^2)$. If, for all i such that $1 \leq i < j$ and $\|\mathbf{x}_i\| \neq 0$, it is true that $\lfloor A_i[k,\ell]\rceil \leq \varepsilon\|\mathbf{x}_i\|$, then*

$$\lfloor A_j[k,\ell]\rceil \leq \frac{O(m^3M)^{j-1}}{\prod_{\substack{1\leq i<j \\ \|\mathbf{x}_i\|\neq 0}} \|\mathbf{x}_i\|} \cdot \gamma$$

for all k,ℓ.

We are now done with Step 1, and will proceed to Step 2 and Step 3 in the next two sections.

5.4 The MCDP

QR-GCD gives a polynomial whose degree is less than the exact GCD if $[\![R[i,i]]\!]$ is not equal to zero for some $i \geq \deg(\mathbf{A}) + \deg(\mathbf{B}) - \deg(\gcd(\mathbf{A},\mathbf{B}))$. The following lemma states that this cannot happen if HOUSEHOLDER does not branch off incorrectly at the **if** statement in Line 2.

LEMMA 3. *Let $\mathbf{A}$ and $\mathbf{B}$ be real polynomials, and let A be their Sylvester matrix. Let r be the rank of A. If, in the first r times* HOUSEHOLDER *is called during the execution of the stabilized version of* HOUSEHOLDER-QR *on $[\![A]\!]$, it is true that* HOUSEHOLDER *never decides that $[\![\sigma_1]\!] = 0$ at the **if** statement on Line 2, then all elements in all rows below Row r in the output of* HOUSEHOLDER-QR *are equal to zero.*

PROOF. The Sylvester matrix A has the property that, if $A = QR$ is the QR-factorization of A, then $R[1,1], R[2,2], \ldots, R[r,r]$ are not zero, and the remaining $\deg(\mathbf{A})+\deg(\mathbf{B})-r$ rows below are all zero rows [2]. Hence, the exact computation must apply r orthogonal transformations to A in the first r iterations of the main loop of HOUSEHOLDER-QR. After that, the exact computation will not apply any other transformations, and the return value is equal to A_{r+1}.

Assume then that the stabilized HOUSEHOLDER never decides $[\![\sigma_1]\!] = 0$ in the first r times it is called. The assumption implies that the stabilized version also applies a transformation to A in each of the first r iterations of the main loop as well. It follows that any elements in Row $r+1$ to Row $\deg(\mathbf{A}) + \deg(\mathbf{B})$ of A_{r+1} must be equal to zero because r transformations are applied just like in the exact algorithm. After that, the stabilized algorithm will also not apply any more transformation, and return the matrix $[\![A_{r+1}]\!]$. □

THEOREM 2. *Let $\mathbf{A}$ and $\mathbf{B}$ be two real polynomials whose coefficients have absolute values not larger than $M \in \mathbb{R}$. Let $d = \deg(\mathbf{A})+\deg(\mathbf{B})$, and let A be the Sylvester matrix of the two polynomials. Let r be the rank of A, and let $A = QR$ be the QR-factorization of A computed by* HOUSEHOLDER-QR*. If*

$$\tau = \Omega\Big(d(\log d + \log M) - \sum_{i=1}^{r} \log|R[i,i]|\Big),$$

then the matrix $[\![R]\!]$ returned by the stabilized HOUSEHOLDER-QR *with input $[\![A]\!]_\tau$ has the following properties:*

(a) $[\![R]\!]$ is upper triangular, and Row r is the last non-zero row.

(b) $\lim_{\tau\to\infty} \sum_{i=0}^{d-r} \langle R[r,d-i]\rangle x^i = R[r,r]\gcd(\mathbf{A},\mathbf{B})$. (Here, $\gcd(\mathbf{A},\mathbf{B})$ is monic.)

In other words, the MCDP of the QR factionization algorithm is $O\Big(d(\log d + \log M) - \sum_{i=1}^{r}\log|R[i,i]|\Big)$.

The proof of the theorem relies on the following lemma. We omit the algebraic manipulations and refer the reader to [11] for the full proof.

LEMMA 4. *There exists a constant C such that, for any j such that $1 \leq j \leq r$, if*

$$2^{-\tau} \leq \lambda \frac{\big(\prod_{i=1}^{r}|R[i,i]|\big)^2}{\Theta(d^2(Cd^5M^3)^r)},$$

then

$$\lfloor A_j[k,\ell]\rceil \leq \lambda \frac{|R[j,j]|}{\Theta(d^2(Cd^3M)^{r-j+1})}$$

for any constant λ, and for any $1 \leq k,\ell \leq d$.

PROOF (SKETCH). We choose C to the constant of the function $O(m^3M)$ in Lemma 2. Note that, in Lemma 2, $\|\mathbf{x}_i\| = |R[i,i]|$ for all i, and $\prod_{\substack{1\leq i<j \\ \|\mathbf{x}_i\|\neq 0}} \|\mathbf{x}_i\| = \prod_{i=1}^{j-1} R[i,i]$ because the first r diagonal entries are non-zero. The inequality on $\lfloor A_j[k,\ell]\rceil$ can be shown by strong induction on j. □

PROOF OF THEOREM 2. We set τ so that $2^{-\tau}$ is less than the quantity specified in Lemma 4 with $\lambda = |R[r,r]|/(dM)$. With this, $\tau = O\Big(d(\log d + \log M) - \sum_{i=1}^{r}\log|R[i,i]|\Big)$. By Lemma 1, we have that no faulty branching at the **if** statement in Line 2 of HOUSEHOLDER can take place because $\lfloor A_j[k,\ell]\rceil \leq \frac{|R[j,j]|}{\Theta(d^2)}$ for all $1 \leq j \leq r$. This implies that r transformations are applied, and all elements below $[\![R[1,1]]\!], [\![R[2,2]]\!], \ldots, [\![R[r,r]]\!]$ are equal to zero. Moreover, Lemma 3 implies that all rows of $[\![R]\!]$ below Row r are zero rows. Therefore, $[\![R]\!]$ is upper triangular.

Lemma 4 and Lemma 2 together imply that $[\![R[r,r]]\!]$ is not zero because $\lfloor R[r,r]\rceil < \lambda \cdot \frac{|R[r,r]|}{\Theta(d^2(Cd^3M))} \cdot \frac{Cd^3M}{|R[r,r]|} < \frac{|R[r,r]|}{\Theta(d^3M)}$. So, Row r is the last non-zero row of R. We have proven (a).

In the exact algorithm, the polynomial $\sum_{i=0}^{d-r} R[r, d-i]x^i$ is similar to the GCD of $\mathbf{A}$ and $\mathbf{B}$. So, $\sum_{i=0}^{d-r} A_{d+1}[r, d-i]x^i$ is equal to $R[r, r]\gcd(\mathbf{A}, \mathbf{B})$. The Shirayanagi–Sweedler stabilization technique implies that Row r of $[\![R]\!]$ will converge to Row r of R, and (b) follows as a result. □

The theorem also implies that the absolute error of every coefficient is less than $\frac{|R[r,r]|}{\Theta(d^3M)}$.

Our crude analysis showed that the constant C is at most 80000, and the constant of the $\Theta(d^2(Cd^5M^3)^r)$ term to be at most 8000. This implies that the MCP can be as large as $d(5\log_2 d + 3\log_2 M + \log_2 80000) + \log_2 8000 + 2\log_2 d$ even without the $\sum_{i=1}^{r} \log|R[i,i]|$ term. Nevertheless, this upper bound is very loose. For example, let $\mathbf{P} = x + \sqrt{2}$, $\mathbf{Q} = x+\sqrt{3}$, and $\mathbf{R} = x+\sqrt{5}$. The formula would imply that the MCDP of $\mathbf{P}^5\mathbf{Q}^5$ and $\mathbf{PQR}^5$ can be as large as $10 \times 5 \times \log_2 10 + 10 \times \log_2 80000 \approx 370$. However, by increasing the precision used in calculation until the degree of the output seem to stabilize, we found empirically that the MCPD was only 32.

5.5 The MSSP

The output of QR-GCD($[\![\mathbf{A}]\!]_\tau, [\![\mathbf{B}]\!]_\tau$) has the same support as $\gcd(\mathbf{A}, \mathbf{B})$ if and only if, for all entries $R[r, i]$ such that $R[r, i] \neq 0$, we have that $[\![R[r, i]]\!]$ is not equal to zero. We can use Lemma 4 to bound the MSSP in terms of the smallest non-zero coefficient of the GCD as follows:

THEOREM 3. *Let μ be the non-zero coefficient of $\gcd(\mathbf{A}, \mathbf{B})$ with the smallest absolute value. Then, the MSSP of the QR-factorization is $O\big(d(\log d+\log M)-\log|\mu|-\sum_{i=1}^{r}\log|R[i,i]|\big)$.*

PROOF. We use Lemma 4 with $\lambda = |R[r, r]\mu|$. This choice of λ gives $\tau = O\big(d(\log d+\log M)-\log|\mu|-\sum_{i=1}^{r}\log|R[i,i]|\big)$ as claimed. By Lemma 4, we have that

$$\lfloor R[r, i]\rceil < |R[r, r]\mu| \cdot \frac{|R[r, r]|}{\Theta(d^2(Cd^3M))} \cdot \frac{Cd^3M}{|R[r, r]|} = \frac{|R[r, r]\mu|}{\Theta(d^3M)},$$

for all i. So, if the coefficient of x^k of $\gcd(\mathbf{A}, \mathbf{B}) \neq 0$, then

$$\begin{aligned}\lfloor R[r, d-k]\rceil &\leq \frac{|R[r, r]\mu|}{\Theta(d^3M)} \leq \frac{|R[r, r]\gcd(\mathbf{A}, \mathbf{B})[k]|}{\Theta(d^3M)} \\ &\leq \frac{|R[r, d-k]|}{\Theta(d^3M)} < |\langle R[r, d-k]\rangle|.\end{aligned}$$

if d is sufficiently large. □

6. SIMPLER BOUND IN THE ALGEBRAIC INTEGERS

The bounds are still incomputable and do not allow one to estimate the MCDP or the MSSP without running the algorithm itself. We demonstate in this section that the MCDP bound can be simplified if the coefficients of the input polynomials belong to $\mathbb{Z}[\xi]$, where ξ is an algebraic integer of degree n. We start by proving a lemma that allow us to estimate the $\sum_{i=1}^{r}\log|R[i,i]|$ term.

Let $\mathbf{A}$ and $\mathbf{B}$ be real polynomials, and let $d = \deg(\mathbf{A}) + \deg(\mathbf{B})$. Let A be their Sylvester matrix, and let $A = QR$ be the matrix's QR-factorization obtained through our algorithm. Let r be the rank of A, and let $\mathbf{a}_i$ denote the ith column of A. Let $\mathbf{e}_1$, $\mathbf{e}_2$, …, and $\mathbf{e}_d$ denote the standard basis of $\mathbb{R}^d$.

LEMMA 5. *We can choose $d-r$ vectors $\mathbf{e}_{i_1}, \mathbf{e}_{i_2}, \ldots, \mathbf{e}_{i_{d-r}}$ out of $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d$ so that the matrix*

$$A' = \begin{bmatrix}\mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_r & \mathbf{e}_{i_1} & \mathbf{e}_{i_2} & \cdots & \mathbf{e}_{i_{d-r}}\end{bmatrix}$$

has non-zero determinant and $\big|\prod_{i=1}^{r} R[i,i]\big| \geq |\det A'|$.

PROOF. Since, A is a Sylvester matrix of rank r, $R[i, i] \neq 0$ for all $i = 1, \ldots, r$. This implies that the first r columns of A are linearly independent.

By the Steinitz exchange lemma, we can choose $d - r$ vectors $\mathbf{e}_{i_1}$, $\mathbf{e}_{i_2}$, …, and $\mathbf{e}_{i_{d-r}}$ out of $\mathbf{e}_1$, $\mathbf{e}_2$, …, $\mathbf{e}_d$ such that these vectors, together with the first r columns of A, form a basis of $\mathbb{R}^d$. So, $\det A' \neq 0$.

Consider the QR-factorization $A' = Q'R'$ computed by our QR-factorization algorithm. Since the first r columns of A and A' are the same, the first r columns of R and R' must be the same as well because the value of the ith column in the upper triangular matrix only depends on the first i columns of the input matrix. Therefore,

$$|\det A'| = \left|\prod_{i=1}^{d} R'[i,i]\right| = \left|\prod_{i=1}^{r} R[i,i]\right|\left|\prod_{i=r+1}^{d} R'[i,i]\right|.$$

We claim that, $|R'[i,i]| \leq 1$ for all i such that $r+1 \leq i \leq d$. Otherwise, the norm of the ith column of R' exceeds 1. However, since the ith column of R' is equal to Q^T multiplied by the ith column of A', its norm must be exactly 1, a contradiction. Hence,

$$\left|\prod_{i=1}^{r} R[i,i]\right| \geq \left|\prod_{i=1}^{r} R[i,i]\right|\left|\prod_{i=r+1}^{d} R'[i,i]\right| = |\det A'|$$

as required. □

Next, we bound the MCDP. Let $g(x) = x^n + c_{n-1}x^{n-1} + \cdots + c_0$ be ξ's minimal polynomial. Let C be a positive integer such that $C \geq |c_i|$ for all i. For positive integer N, let $\mathbb{Z}[\xi]\langle\langle N\rangle\rangle$ denote the set $\{a_0 + \cdots + a_{n-1}\xi^{n-1} : a_0, \ldots, a_{n-1} \in \langle\langle N\rangle\rangle\}$.

THEOREM 4. *Let $\mathbf{A}$ and $\mathbf{B}$ be polynomials whose coefficients are members of $\mathbb{Z}[\xi]\langle\langle N\rangle\rangle$, and $d = \deg(\mathbf{A}) + \deg(\mathbf{B})$. Then, the MCDP of the QR-factorization algorithm on input $\mathbf{A}$ and $\mathbf{B}$ is $O(dn(\log d + \log N + n\log C))$.*

The proof relies on the following lemma, whose proof can be found in [11].

LEMMA 6. *Let A be an $d \times d$ matrix with entries from $\mathbb{Z}[\xi]\langle\langle N\rangle\rangle$. If $\det(A) \neq 0$, then*

$$|\det(A)| \geq \frac{1}{(2C)^{O(dn^2)}(ndN)^{O(dn)}D},$$

where $D = 1 + |\xi| + \ldots + |\xi|^{n-1}$.

PROOF. (Theorem 4) Consider the matrix A' defined in Lemma 5. Every entry of A' is a member of $\mathbb{Z}[\xi]\langle\langle N\rangle\rangle$. So,

$$\left|\prod_{i=1}^{r} R[i,i]\right| \geq \frac{\alpha}{(2C)^{O(dn^2)}(nmN)^{O(dn)}}.$$

In other words,

$$\begin{aligned}-\sum_{i=1}^{r}|\log R[i,i]| &\leq \log\left((2C)^{O(dn^2)}(ndN)^{O(dn)}\right) + \log D \\ &= O(dn(\log d + \log N + n\log C)).\end{aligned}$$

Moreover, since the coefficients are members of $\mathbb{Z}[\xi]\langle\langle N\rangle\rangle$, we have that the coefficient with the largest absolute value cannot have absolute value greater than $N + N|\xi| + \cdots + N|\xi|^{n-1} = ND$. Therefore, the $\log M$ term in Theorem 2 is of order $O(\log N)$. So, the MCDP is $O(d(\log d + \log N) + dn(\log d + \log N + n \log C)) = O(dn(\log d + \log N + n \log C))$. □

We claim that the MSSP also admits the same bound, but we will not discuss the result due to space constraint.

The theorem implies that our algorithm can find the degree of the GCD in $\mathbb{Z}[\xi]$ using $O(d^3)$ additions and multiplications, $O(d)$ divisions and square roots. The distinct feature of our algorithm is that it reduces coefficients to floating-point numbers, and therefore is very simple to implement. Assuming that we can do multiplication of floating-point numbers of precision τ in $O(\tau \log \tau)$ time, division in $O(\tau^2)$ (using long division), square roots in $O(\tau^2 \log \tau)$ time (using Newton's method which performs $O(\log \tau)$ long divisions), our algorithm has bit complexity $\tilde{O}(d^4 n(\log d + \log N) + d^3 n^4 \log^2 C)$. This bound compares favorably to the $\tilde{O}((d^5 n^4 + d^3 n^5) \log^2(nNC^d))$ bound of the non-modular subresultant algorithm in [12]. However, there are modular algorithms with vastly superior worst-case and randomized running time [13, 6]. As a result of Lemma 4, our algorithm can approximate the coefficients of the GCD to any given precision, but the above bit complexity does not hold if the demanded precision is $\omega(d(\log d + \log N))$. We believe coefficients of the exact GCD can be recovered by integer relation algorithms, but have yet to confirm the possibility.

7. CONCLUSION AND DISCUSSION

This paper provides a preliminary theoretical study of the MCP of algorithms for finding the GCD of univariate real polynomials. We demonstrate the hardness of the problem of bounding the MCP by showing that the MCP is generally incomputable. We also outline a general technique for bounding the MCP, and use it to bound the MCDP and MSSP of the QR-factorization algorithm. The incomputability result and the upper bounds together suggest that roundoff errors alone can have significant impact on accuracy of GCD algorithms. Additionally, although the bounds we gave are still incomputable, we show that they can be simplified if coefficients of the input polynomials are algebraic integers.

Though it would appear that no practical applications have emerged from our study, we believe it provides a new perspective on stability of numerical algorithms as the MCP and similar numbers capture the hardness of simulating the algorithm with fixed-precision floating-point numbers. Future research directions include finding connections between traditional numerical stability and the MCP, and proving lower bounds and upper bounds on the MCP of other algorithms.

REFERENCES

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations, Computer Science, and Applied Mathematics.* Academic Press, 1983.

[2] R. Corless, S. Watt, and L. Zhi. QR factoring to compute the GCD of univariate approximate polynomials. *IEEE Trans. Signal Processing*, 52, 2004.

[3] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *Proc. 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*, 1995.

[4] I. Z. Emiris, A. Galligo, and H. Lombardi. Numerical univariate polynomial gcd. In *Proc. AMS-SIAM Summer Seminar on Math. of Numerical Analysis*, 1995.

[5] I. Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure and Applied Algebra*, 117 and 118:229–251, 1997.

[6] M. J. Encarnación. On a modular algorithm for computing gcds of polynomials over algebraic number fields. In *Proc. 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC '94)*, 1994.

[7] G. Golub and C. Van Loan. *Matrix Computation*. Johns Hopkins University Press, 1996.

[8] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[9] V. Hribernig and H. J. Stetter. Detection and validation of clusters of polynomial zeros. *J. Symb. Comput.*, 24(6):667–681, 1997.

[10] N. K. Karmarkar and Y. N. Lakshman. On approximate GCDs of univariate polynomials. *J. Symb. Comput.*, 26:653–666, 1998.

[11] P. Khungurn. Shirayanagi-Sweedler algebraic algorithm stabilization and polynomial GCD algorithms. Master's thesis, Massachusetts Institute of Technology, 2007. http://web.mit.edu/pramook/www/meng.pdf.

[12] L. Langemyr. An analysis of the subresultant algorithm over an algebraic number field. In *Proc. 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC '91)*, 1991.

[13] L. Langemyr and S. McCallum. The computation of polynomial greatest common divisors over an algebraic number field. *J. Symb. Comput.*, 8(5):429–448, 1989.

[14] H. Minakuchi, H. Kai, K. Shirayanagi, and M-T. Noda. Algorithm stabilization techniques and their application to symbolic computation of generalized inverses. In *Electronic Proc. of the IMACS Conference on Applications of Computer Algebra (IMACS-ACA'97)*, 1997.

[15] M. Sasaki and M. T. Noda. Approximate gcd and its applications to ill-conditioned algebraic equations. *J. CAM*, 38:335–351, 1991.

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

[17] H. Sekigawa and K. Shirayanagi. Zero in interval computation and its applications to Sturm's algorithm. Poster presentation at International Symposium on Symbolic and Algebraic Computation (ISSAC'95), 1995.

[18] K. Shirayanagi. Floating-point Gröbner bases. *Mathematics and Computers in Simulation*, 42:509–528, 1996.

[19] K. Shirayanagi and M. Sweedler. A theory of stabilizing algebraic algorihms. Technical Report 95-28, Mathematical Sciences Institute, Cornell University, 1995.

[20] K. Shirayanagi and M. Sweedler. Remarks on automatic algorithm stabilization. *J. Symb. Comput.*, 26(6):761–765, 1998.

[21] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1988.

[22] C. Zarowski, X. Ma., and F. Fairman. A QR-factorization method for computing the greatest common divisor of polynomials with real-valued coefficients. *IEEE Trans. Signal Processing*, 48, 2000.

[23] L. Zhi. Displacement structure in computing the approximate gcd of univariate polynomials. In W. Sit and Z. Li, editors, *Mathematics*, Lecture Notes Series on Computing, pages 228–298. World Scentific, 2003.