Approximate polynomial GCD over integers

Approximate Polynomial GCD over Integers

Kosaku Nagasaka

Graduate School of Human Development and Environment, Kobe University, Japan

Abstract

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 polynomials, lots of algorithms have been introduced, approximate univariate GCD and approximate multivariate factorization for example. However, for polynomials over integers having coefficients rounded from empirical data, changing their coefficients over reals does not remain them in the polynomial ring over integers, hence we need several approximate operations 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.

Key words: approximate polynomial GCD, numerical polynomial GCD

1. Introduction

For polynomials with empirical data on their coefficients, we have to use approximate GCD algorithms to find appropriate factors. There are many studies and various algorithms (Karcanias et al., 2006; Diaz-Toca and Gonzalez-Vega, 2006; Christou and Mitrouli, 2005; Li and Zeng, 2005; Zeng and Dayton, 2004; Zarowski et al., 2000; Corless et al., 2004; Zhi, 2003; Pan, 2001; Zhi and Noda, 2000b,a; Rupprecht, 1999; Karmarkar and Lakshman, 1998; Pan, 1998; Emiris et al., 1997; Rössner and Seifert, 1996; Emiris et al., 1996; Mitrouli and Karcanias, 1993; Ochi et al., 1991; Sasaki and Noda, 1989; Schönhage, 1985), if the given polynomial pair and the desired GCD are over the complex field. For polynomials over integers having coefficients rounded from empirical data, we can not use those algorithms since they compute GCDs over complex numbers. We need another type of algorithms. However, such an algorithm has not been studied enough. There are only two known algorithms: the author's poster presentation at ISSAC 2008 and the von zur Gathen and Shparlinski's one at LATIN 2008. In this paper, we introduce algorithms to compute the following approximate GCD over integers.

Email address: `nagasaka@main.h.kobe-u.ac.jp` (Kosaku Nagasaka).

URL: `http://wwwmain.h.kobe-u.ac.jp/~nagasaka/` (Kosaku Nagasaka).

Definition 1 (Approximate 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 $f(\vec{x})$ and $g(\vec{x})$ satisfy

$$f(\vec{x}) = t(\vec{x})h(\vec{x}) + \Delta_f(\vec{x}), \;\; g(\vec{x}) = s(\vec{x})h(\vec{x}) + \Delta_g(\vec{x}), \;\; \varepsilon = \max\{\|\Delta_f\|, \|\Delta_g\|\}, \quad (1)$$

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 $\boldsymbol{\varepsilon}$. ($\|p\|$ denotes a suitable norm of polynomial $p(\vec{x})$.) $\triangleleft$

We note that we compute a GCD of two 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 pair of polynomials still remains over integers. The conventional approximate GCD algorithms can not be used for this problem since determining leading coefficients and content scalars and rounding to integers can not be done easily since such conversions make polynomials far from desired nearest GCDs over integers.

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

$$\begin{aligned} f(x) &= 54x^6 - 36x^5 - 192x^4 + 42x^3 + 76x^2 - 62x + 15, \\ g(x) &= 73x^5 + 36x^4 - 103x^3 - 70x^2 - 48x + 35. \end{aligned}$$

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

$$\begin{aligned} f(x) &\approx (6x^4 - 10x^3 - 8x^2 + 7x - 3)(9x^2 + 9x - 5) \\ &= 54x^6 - 36x^5 - 192x^4 + 4\underline{1}x^3 + 76x^2 - 62x + 15, \\ g(x) &\approx (8x^3 - 4x^2 - 3x - 7)(9x^2 + 9x - 5) \\ &= 7\underline{2}x^5 + 36x^4 - 103x^3 - 70x^2 - 48x + 35. \end{aligned}$$

If we apply the conventional approximate GCD algorithm by Kaltofen et al. (2006) for example, the following approximate GCD is found. However, it is difficult to make them being polynomials over integers although their tolerance is very small: $\varepsilon = 10^{-8}$.

$$\begin{aligned} f(x) &\approx (1.00x^2 + 0.99x - 0.55)(54.34x^4 - 90.20x^3 - 72.20x^2 + 63.65x - 27.07), \\ g(x) &\approx (1.00x^2 + 0.99x - 0.55)(72.84x^3 - 36.20x^2 - 26.83x - 63.33). \end{aligned}$$

$\triangleleft$

Example 3. 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.

$$\begin{aligned}
f(x_1,x_2) &= 89x_1^2x_2^2 - 87x_1x_2^2 - 136x_2^2 + 15x_1^2x_2 \\
&\qquad\qquad + 132x_1x_2 + 119x_2 - 42x_1^2 + 166x_1 + 139, \\
g(x_1,x_2) &= 56x_1^2x_2^2 - 45x_1x_2^2 - 98x_2^2 - 13x_1^2x_2 \\
&\qquad\qquad + 46x_1x_2 + 225x_2 - 12x_1^2 + 80x_1 - 112.
\end{aligned}$$

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

$$\begin{aligned}
f(x_1,x_2) &\approx (18x_1x_2 + 15x_2 + 14x_1 + 10) \times (5x_1x_2 - 9x_2 - 3x_1 + 14) \\
&= \underline{90}x_1^2x_2^2 - 87x_1x_2^2 - 13\underline{5}x_2^2 + 1\underline{6}x_1^2x_2 \\
&\qquad\qquad + 13\underline{1}x_1x_2 + 1\underline{20}x_2 - 42x_1^2 + 166x_1 + 1\underline{40}, \\
g(x_1,x_2) &\approx (11x_1x_2 + 11x_2 + 4x_1 - 8) \times (5x_1x_2 - 9x_2 - 3x_1 + 14) \\
&= 5\underline{5}x_1^2x_2^2 - 4\underline{4}x_1x_2^2 - 9\underline{9}x_2^2 - 13x_1^2x_2 \\
&\qquad\qquad + 4\underline{5}x_1x_2 + 22\underline{6}x_2 - 12x_1^2 + 80x_1 - 112.
\end{aligned}$$

In this case, $\Delta_f = -x_1^2x_2^2 - x_2^2 - x_1^2x_2 + x_1x_2 - x_2 - 1$ and $\Delta_g = x_1^2x_2^2 - x_1x_2^2 + x_2^2 + x_1x_2 - x_2$. Moreover, we note that any approximate GCD is not unique since they may have several GCDs even for the same tolerance. $\triangleleft$

The algorithm by von zur Gathen and Shparlinski (2008) and its revised version (von zur Gathen et al., 2010) are based on the result from approximate integer common divisors by Howgrave-Graham (2001) hence the algorithm only works for very tiny tolerances and one of input polynomials $f(\vec{x})$ and $g(\vec{x})$ must be exact. In this paper, we compute approximate GCDs over integers by using the well-known subresultant mapping and lattice basis reduction (the LLL algorithm by Lenstra et al. (1982)) hence the present algorithm works for not only very tiny but also small tolerances and for two approximate polynomials. In Section **2**, we introduce a simple algorithm and examples with exact and approximate polynomials. However, the simple algorithm is very time-consuming so we give some criteria to improve the algorithm in Section **3**, including various numerical examples. In Section **4**, we extend the algorithm to several polynomials and introduce its application. Some remarks are given in Section **5**.

2. Approximate GCD by Lattice Basis Reduction

Computing a polynomial GCD is usually done by PRS (polynomial remainder sequence by the traditional Euclidean algorithm) in case of exact coefficients, and QR or Singular value decompositions in case of numerical coefficients (we note that some recent algorithms use more complicated matrix decompositions). Such conventional algorithms are related to the well-known subresultant mapping. Our algorithm also uses the several well-known facts of the subresultant mapping. In the below, we simply review them and introduce how to use the lattice basis reduction with the mapping.

2.1. *Subresultant mapping*

Let $f(\vec{x})$ and $g(\vec{x})$ be multivariate polynomials over integers, of total degrees $n = \mathrm{tdeg}(f)$ and $m = \mathrm{tdeg}(g)$, respectively. In this paper, 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. This mapping is the same as in Gao et al. (2004), 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.

We construct the coefficient vector $\vec{p}$ of polynomial $p(\vec{x})$ by the lexicographic ascending order. To see the number of elements of a coefficient vector, we define the notation: $\beta_{d,r} = \binom{d-r+\ell}{\ell}$. The number of terms $x_1^{i_1} \cdots x_\ell^{i_\ell}$ satisfying $i_1 + \cdots + i_\ell \leq d$ is denoted by $\beta_{d,0}$. Hence, the sizes of the coefficient vectors $\vec{f}$ and $\vec{g}$ of $f(\vec{x})$ and $g(\vec{x})$ are $\beta_{n,0}$ and $\beta_{m,0}$, respectively. The k-th convolution matrix $C_k(f)$ is defined to satisfy $C_k(f)\vec{p} = \overrightarrow{fp}$ for any polynomial $p(\vec{x})$ of total degree $k-1$, where $\vec{p} \in \mathbb{Z}^{\beta_{k-1,0} \times 1}$ and $C_k(f) \in \mathbb{Z}^{\beta_{n+k-1,0} \times \beta_{k-1,0}}$. Using this matrix, we also 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

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

Computing $s(\vec{x})$ and $t(\vec{x})$ such that the mapping is not injective is done by computing the null space of matrix $Syl_r(f,g)$. Hence, finding the polynomial GCD is reduced to finding null vectors. Some approximate GCD algorithms (Corless et al., 2004; Zarowski et al., 2000) use the QR decomposition of $Syl_0(f,g)^t$, the transpose of the Sylvester matrix of $f(\vec{x})$ and $g(\vec{x})$ (see also Laidacker (1969) for the exact case). Moreover, including them, most of algorithms are based on this subresultant mapping (see Karcanias et al. (2006); Rupprecht (1999) for more information).

2.2. *Exact GCD by short vectors*

We introduce the exact version of our algorithm using the subresultant mapping and the lattice basis reduction (the LLL algorithm by Lenstra et al. (1982)) which is useful to find null vectors, since the LLL algorithm can find a short vector $\vec{u}$ satisfying

$$\|\vec{u}\|_2 \leq 2^{(d-1)/2} \min\{\|\vec{v}\|_2 \mid \vec{0} \neq \vec{v} \in L\},\ \vec{u} \in L,$$

for the given lattice $L = \{r_1\vec{v}_1 + \cdots + r_d\vec{v}_d | r_i \in \mathbb{Z}\} \subseteq \mathbb{Z}^k$. Moreover, there are lots of research (Schnorr and Euchner, 1994; Lenstra et al., 1982; Backes and Wetzel, 2002) studying the LLL algorithm, and is the notable remark that mostly the LLL algorithm may find very short vectors comparing with the bound $2^{(d-1)/2}$.

We construct the lattice generated by 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 $(\beta_{n-1,r}+\beta_{m-1,r}) \times (\beta_{n-1,r}+\beta_{m-1,r}+\beta_{n+m-1,r})$.

Lemma 4. *Let B be the maximum of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$. For the lattice generated by row vectors of $\mathcal{L}(f,g,r,c_B)$ with $c_B = 2^{(\beta_{n-1,r}+\beta_{m-1,r}-1)/2}$ $\sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$, 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.* $\lhd$

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 $\mathcal{L}(f,g,r,c_B)$ has the following vector $\vec{u}_{min}$.

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

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

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

Since all the non-zero elements of right $\beta_{n+m-1,r}$ columns of row vectors of $\mathcal{L}(f,g,r,c_B)$ are larger than or equal to $c_B = 2^{(\beta_{n-1,r}+\beta_{m-1,r}-1)/2}\sqrt{\beta_{n-1,r}+\beta_{m-1,r}}B$ in absolute value, the right $\beta_{n+m-1,r}$ 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 $Syl_r(f,g)$ and the lemma is proved. $\square$

For example, B would be the maximum of Landau-Mignotte bounds (Mignotte, 1974) of $f(x)$ and $g(x)$ for univariate polynomials, or $B = \max\{2^{n\ell} \parallel f \parallel_2,\ 2^{m\ell} \parallel g \parallel_2\}$ for multivariate polynomials, given by Gel′fond (1960). We note that we can decrease the bound c_B by using the coefficients bounds of factors of $f(\vec{x})$ and $g(\vec{x})$ separately, however, this is not so important since the LLL algorithm may find appropriate short vectors with very small $c \ll c_B$.

Example 5. We compute a polynomial GCD of the following very simple polynomials.

$$\begin{aligned} f(x) &= 28x^2 - x - 15 &&= (7x+5)(4x-3), \\ g(x) &= 42x^2 + 65x + 25 &&= (7x+5)(6x+5). \end{aligned}$$

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 row vectors of $\mathcal{L}(f,g,r,c)$.

$$\left(\begin{array}{cccc|cccc} 1&0&0&0&-15&-1&28&0\\ 0&1&0&0&0&-15&-1&28\\ 0&0&1&0&25&65&42&0\\ 0&0&0&1&0&25&65&42 \end{array}\right) \to \left(\begin{array}{cccc|cccc} -5&-6&-3&4&0&0&0&0\\ -1&-2&-1&1&-10&-9&-3&-14\\ 1&0&0&0&-15&-1&28&0\\ -1&-1&-1&1&-10&-24&-4&14 \end{array}\right).$$

The first row is a short vector corresponding to cofactors, hence we get $4x-3$ and $6x+5$ as cofactors and $7x+5$ as the polynomial GCD of $f(x)$ and $g(x)$. We note that Lemma 4 guarantees only with such a large c_B so we have to enlarge c and apply the LLL algorithm repeatedly if we could not find appropriate short vectors. ◁

Example 6. We compute the polynomial GCD of the following very simple polynomials.

$$\begin{aligned} f(\vec{x}) &= 12x_1^2-10x_1x_2+2x_1-12x_2^2+23x_2-10 &&= (6x_1+4x_2-5)(2x_1-3x_2+2),\\ g(\vec{x}) &= 30x_1^2+62x_1x_2-43x_1+28x_2^2-47x_2+15 &&= (6x_1+4x_2-5)(5x_1+7x_2-3). \end{aligned}$$

We construct the following matrix $\mathcal{L}(f,g,r,c)$ with $r=0$ and $c=1$:

$$\mathcal{L}(f,g,r,c) = \left(\begin{array}{cccccc|cccccccccc} 1&0&0&0&0&0&-10&23&-12&0&2&-10&0&12&0&0\\ 0&1&0&0&0&0&0&-10&23&-12&0&2&-10&0&12&0\\ 0&0&1&0&0&0&0&0&0&0&-10&23&-12&2&-10&12\\ 0&0&0&1&0&0&15&-47&28&0&-43&62&0&30&0&0\\ 0&0&0&0&1&0&0&15&-47&28&0&-43&62&0&30&0\\ 0&0&0&0&0&1&0&0&0&0&15&-47&28&-43&62&30 \end{array}\right),$$

and apply the LLL algorithm to the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$:

$$\to \left(\begin{array}{cccccc|cccccccccc} -3&7&5&-2&3&-2&0&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&-10&23&-12&0&2&-10&0&12&0&0\\ 1&1&0&0&0&0&-10&13&11&-12&2&-8&-10&12&12&0\\ 0&0&1&0&0&0&0&0&0&0&-10&23&-12&2&-10&12\\ 1&-3&-2&1&-1&1&5&-9&-6&8&-6&-4&20&-5&16&6\\ 0&2&0&0&1&-1&0&-5&-1&4&-15&8&14&43&-8&-30 \end{array}\right).$$

The first row is a short vector corresponding to cofactors, hence we get $2x_1-3x_2+2$ and $5x_1+7x_2-3$ as cofactors and $6x_1+4x_2-5$ as the polynomial GCD of $f(\vec{x})$ and $g(\vec{x})$. ◁

2.3. Approximate GCD by short vectors

For finding an approximate GCD, there may not exist any null vector in the lattice since we can use the conventional algorithm if it exists. However, there may exist vectors having small norms instead of null vectors, so we can find appropriate vectors by the lattice basis reduction. This means that we can compute candidate cofactors $t(\vec{x})$ and $s(\vec{x}) \in \mathbb{Z}[\vec{x}]$ such that $s(\vec{x})f(\vec{x}) + t(\vec{x})g(\vec{x}) \approx 0$. If there is an approximate GCD over integers, let it be $h(\vec{x})$, $h(\vec{x})$ may satisfy $f(\vec{x}) \approx t(\vec{x})h(\vec{x})$ and $g(\vec{x}) \approx -s(\vec{x})h(\vec{x})$. Since all the polynomials are over integers, $f(\vec{x})$ and $g(\vec{x})$ may not be divisible by $h(\vec{x})$, hence it is not possible to find $h(\vec{x})$ by dividing by the approximate cofactors computed. We use the LLL algorithm again to get $h(\vec{x})$. Let $\mathcal{H}(f,g,r,c,t,s)$ be the following matrix of size $(\beta_{r+1,0}+1) \times (\beta_{n,0}+\beta_{m,0}+\beta_{r+1,0}+1)$.

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

where $\vec{f}$ and $\vec{g}$ denote the coefficient vectors of $f(\vec{x})$ and $g(\vec{x})$, respectively. Although the following lemma does not guarantee that we can find an approximate GCD, there may exist vectors having small norms so we can find appropriate vectors by the lattice basis reduction, as for finding null vectors.

Lemma 7. *Let B be the maximum of coefficients of any factors of $f(\vec{x})$ and $g(\vec{x})$. For the lattice generated by row vectors of $\mathcal{H}(f,g,r,c_H,t,s)$ with $c_H = 2^{\beta_{r+1,0}/2}\sqrt{\beta_{r+1,0}+1}B+1$, 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.* ◁

Proof. The proof is similar to that of Lemma 4. □

We note that the first element of the short vector whose 2-nd, …, $(\beta_{r+1,0}+1)$-th elements are a multiple of the transpose of the coefficient vector of the polynomial GCD of $f(\vec{x})$ and $g(\vec{x})$ is always 1 representing the given polynomials $f(\vec{x})$ and $g(\vec{x})$ in the exact case.

Example 8. We compute an approximate GCD over integers of the following very simple polynomials which are slightly different from the polynomials in Example 5.

$$\begin{aligned} f(x) &= 28x^2 - x \underline{-14} &&= (7x+5)(4x-3)+1, \\ g(x) &= 42x^2 + 65x + 25 &&= (7x+5)(6x+5). \end{aligned}$$

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 row vectors of $\mathcal{L}(f, g, r, c)$.

$$\left(\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & -14 & -1 & 28 & 0 \\ 0 & 1 & 0 & 0 & 0 & -14 & -1 & 28 \\ 0 & 0 & 1 & 0 & 25 & 65 & 42 & 0 \\ 0 & 0 & 0 & 1 & 0 & 25 & 65 & 42 \end{array}\right) \rightarrow \left(\begin{array}{cccc|cccc} -5 & -6 & -3 & 4 & -5 & -6 & 0 & 0 \\ 4 & 4 & 2 & -3 & -6 & -5 & -3 & -14 \\ 9 & 11 & 5 & -7 & -1 & -13 & -4 & 14 \\ 1 & 0 & 0 & 0 & -14 & -1 & 28 & 0 \end{array}\right).$$

We take the first row vector as candidate cofactors since the right hand part of the first row is the smallest. 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.

$$\left(\begin{array}{ccc|cccccc} 1 & 0 & 0 & -14 & -1 & 28 & 25 & 65 & 42 \\ 0 & 1 & 0 & -3 & 4 & 0 & 5 & 6 & 0 \\ 0 & 0 & 1 & 0 & -3 & 4 & 0 & 5 & 6 \end{array}\right) \rightarrow \left(\begin{array}{ccc|cccccc} 0 & 1 & 0 & -3 & 4 & 0 & 5 & 6 & 0 \\ 0 & 0 & 1 & 0 & -3 & 4 & 0 & 5 & 6 \\ 1 & -5 & -7 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right).$$

Hence, we get $7x + 5$ as an approximate polynomial GCD over integers and $4x - 3$ and $6x + 5$ as approximate cofactors and the tolerance is 1. Moreover, this approximate GCD is the nearest one since $\varepsilon = 1$ always means that the corresponding approximate GCD over integers is the nearest, for relatively prime polynomials. We note again that Lemma 4 and Lemma 7 guarantee only with such large c_B and c_H so we have to enlarge c_B and c_H and apply the LLL algorithm repeatedly if we could not find appropriate short vectors. $\triangleleft$

Example 9. We compute an approximate GCD over integers of the following very simple polynomials which are slightly different from the polynomials in Example 6.

$$\begin{aligned} f(\vec{x}) &= 12x_1^2 - 10x_1x_2 + \underline{2}x_1 - 12x_2^2 + 23x_2 - \underline{10} \\ &= (6x_1 + 4x_2 - 5)(2x_1 - 3x_2 + 2) - x_1 + 1, \\ g(\vec{x}) &= 30x_1^2 + 62x_1x_2 - 43x_1 + 28x_2^2 - \underline{47}x_2 + \underline{15} \\ &= (6x_1 + 4x_2 - 5)(5x_1 + 7x_2 - 3) + 2x_2 - 1. \end{aligned}$$

We construct the following matrix $\mathcal{L}(f, g, r, c)$ with $r = 0$ and $c = 1$:

$$\mathcal{L}(f, g, r, c) = \left(\begin{array}{cccccc|cccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & -9 & 23 & -12 & 0 & 1 & -10 & 0 & 12 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -9 & 23 & -12 & 0 & 1 & -10 & 0 & 12 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9 & 23 & -12 & 1 & -10 & 12 \\ 0 & 0 & 0 & 1 & 0 & 0 & 14 & -45 & 28 & 0 & -43 & 62 & 0 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 14 & -45 & 28 & 0 & -43 & 62 & 0 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 14 & -45 & 28 & -43 & 62 & 30 \end{array}\right),$$

and apply the LLL algorithm to the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$:

$$\rightarrow \left(\begin{array}{cccccc|cccccccccc} 3 & -7 & -5 & 2 & -3 & 2 & 1 & 0 & -6 & 0 & -10 & 11 & 0 & 5 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -9 & 23 & -12 & 0 & 1 & -10 & 0 & 12 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -9 & 23 & -12 & 0 & 1 & -10 & 0 & 12 & 0 \\ 2 & -4 & -4 & 1 & -2 & 1 & -4 & 9 & 2 & -8 & 9 & -13 & -8 & 7 & -6 & -18 \\ -3 & 7 & 6 & -2 & 3 & -2 & -1 & 0 & 6 & 0 & 1 & 12 & -12 & -4 & -10 & 12 \\ -5 & 13 & 9 & -3 & 6 & -4 & 3 & -13 & 5 & 12 & -13 & 6 & 22 & 31 & -2 & -12 \end{array}\right).$$

We take the first row vector as candidate cofactors since the right hand part of the first row is the smallest. We construct the following matrix $\mathcal{H}(f,g,r,c,t,s)$ with $c=1$:

$$\mathcal{H}(f,g,r,c,t,s) = \left(\begin{array}{cccc|cccccccccccc} 1 & 0 & 0 & 0 & -9 & 23 & -12 & 1 & -10 & 12 & 14 & -45 & 28 & -43 & 62 & 30 \\ 0 & 1 & 0 & 0 & 2 & -3 & 0 & 2 & 0 & 0 & -3 & 7 & 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & -3 & 0 & 2 & 0 & 0 & -3 & 7 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & -3 & 2 & 0 & 0 & 0 & -3 & 7 & 5 \end{array}\right),$$

and apply the LLL algorithm, to compute an approximate GCD:

$$\rightarrow \left(\begin{array}{cccc|cccccccccccc} 0 & 1 & 0 & 0 & 2 & -3 & 0 & 2 & 0 & 0 & -3 & 7 & 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 & -3 & 0 & 2 & 0 & 0 & -3 & 7 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & -3 & 2 & 0 & 0 & 0 & -3 & 7 & 5 \\ 1 & 5 & -4 & -6 & 1 & 0 & 0 & -1 & 0 & 0 & -1 & 2 & 0 & 0 & 0 & 0 \end{array}\right).$$

We take the last row vector as an approximate GCD since the first element of the row is 1 meaning the coefficient vectors of $f(\vec{x})$ and $g(\vec{x})$. Hence, we get $2x_1 - 3x_2 + 2$ and $5x_1 + 7x_2 - 3$ as cofactors, $6x_1 + 4x_2 - 5$ as the GCD of $f(\vec{x})$ and $g(\vec{x})$, and $\varepsilon = 2$ in the max norm. $\triangleleft$

Based on the above lemmas and examples, we have the following algorithm.

Algorithm 1. (approximate GCD over integers)

```
  Input: $f, g \in \mathbb{Z}[\vec{x}]$,
Output: $h, t, s \in \mathbb{Z}[\vec{x}]$ satisfying $f(\vec{x}) \approx t(\vec{x})h(\vec{x})$ and $g(\vec{x}) \approx s(\vec{x})h(\vec{x})$.
 1. $\varepsilon \leftarrow 1$ and while $\varepsilon < \min\{\|f\|, \|g\|\}$ do 2–14 (or do once for the possible smallest $\varepsilon$)
 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 $\mathcal{L}(f,g,r,c)$
 4.     while $c \leq c_B$ do 5–12 (or do once for $c = \max\{\|f\|, \|g\|\}$)
 5.       apply the LLL algorithm to the lattice generated by rows of $\mathcal{L}(f,g,r,c)$
 6.       for each basis vector sorted by the norm of right $\beta_{n+m-1,r}$ columns, do 7–11
 7.         $c' \leftarrow \max\{\|f\|, \|g\|\}$ and construct a matrix $\mathcal{H}(f,g,r,c',t,s)$
```

```
8.      while $c' \leq c_H$ do 9–11 (or do once for $c' = \max\{\|f\|, \|g\|\}$)
9.        apply the LLL algorithm to the lattice generated by rows of $\mathcal{H}(f,g,r,c',t,s)$
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 $\|f - th\| + \|g - sh\| \leq 2\varepsilon$
11.       $c' \leftarrow c' \times \max\{\|f\|, \|g\|\}$
12.     $c \leftarrow c \times \max\{\|f\|, \|g\|\}$
13.   $r \leftarrow r - 1$
14. $\varepsilon \leftarrow \varepsilon \times 10$
15. output "not found".
```

3. Further Improvements and Numerical Experiments

Algorithm 1 is time-consuming since we can not determine which lattice basis vector is related to a pair of approximate cofactors in advance and we need to try on all the candidate basis vectors. We note that the number of candidate vectors is $\beta_{n-1,r} + \beta_{m-1,r}$. To decrease the number of trials, we introduce the following criteria by which we can determine which short vector is useless in advance.

Criterion 1 (Zero Constant Term).
Let $t(\vec{x})$ and $s(\vec{x})$ be the polynomials given by a vector $\vec{u}$ in the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$ as in the proof of Lemma 4. $\vec{u}$ is not corresponding to any approximate GCD, if

$$t(\vec{0}) = 0 \text{ while } |f(\vec{0})| > \varepsilon \text{ or } s(\vec{0}) = 0 \text{ while } |g(\vec{0})| > \varepsilon.$$

Proof. For any polynomial $h(\vec{x})$ satisfying the equation (1), $t(\vec{0}) = 0$ means that $t(\vec{x})h(\vec{x})$ does not have any non-zero constant term, and we have $\Delta_f(\vec{0}) = f(\vec{0})$. Therefore, for the given tolerance ε, there is not any polynomial $h(\vec{x})$ such that $\|\Delta_f(\vec{x})\| \leq \varepsilon$ if $|f(\vec{0})| > \varepsilon$. Another condition: $s(\vec{0}) = 0$ and $|g(\vec{0})| > \varepsilon$, is also proved by the same way. □

Criterion 2 (Degenerated Cofactors).
Let $t(\vec{x})$ and $s(\vec{x})$ be the polynomials given by a vector $\vec{u}$ in the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$ as in the proof of Lemma 4. $\vec{u}$ is not corresponding to any approximate GCD, if

$$\begin{aligned} & n - \operatorname{tdeg}(t) > m - \operatorname{tdeg}(s) \text{ while } \|\operatorname{term}_n(f)\| > \varepsilon, \\ \text{or } & n - \operatorname{tdeg}(t) < m - \operatorname{tdeg}(s) \text{ while } \|\operatorname{term}_m(g)\| > \varepsilon, \end{aligned}$$

where $\operatorname{term}_d(p)$ denotes the sum of monomials of total degree d, of polynomial $p(\vec{x})$.

Proof. $n - \operatorname{tdeg}(t) > m - \operatorname{tdeg}(s)$ implies $\operatorname{tdeg}(h) = m - \operatorname{tdeg}(s)$ and $\operatorname{tdeg}(th) < \operatorname{tdeg}(f)$ for any approximate GCD $h(\vec{x})$ satisfying the equation (1). This means that $\Delta_f(\vec{x}) = f(\vec{x}) - t(\vec{x})h(\vec{x})$ has $\operatorname{term}_n(f)$ at least. Therefore, for the given tolerance ε, there is not any polynomial $h(\vec{x})$ such that $\|\Delta_f(\vec{x})\| \leq \varepsilon$ if $\|\operatorname{term}_n(f)\| > \varepsilon$. Another condition: $n - \operatorname{tdeg}(t) < m - \operatorname{tdeg}(s)$ and $\|\operatorname{term}_m(g)\| > \varepsilon$, is also proved by the same way. □

Criterion 3 (Leading Monomials).
Let $t(\vec{x})$ and $s(\vec{x})$ be the polynomials of total degree $n-k$ and $m-k$, respectively, given by a vector $\vec{u}$ in the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$ as in the proof of Lemma 4. $\vec{u}$ is not corresponding to any approximate GCD, if

$$\exists i, \left| \frac{|\mathrm{cf}_{x_i^{n-k}}(t)\mathrm{cf}_{x_i^m}(g)| - |\mathrm{cf}_{x_i^{m-k}}(s)\mathrm{cf}_{x_i^n}(f)|}{|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)|} \right| > \varepsilon$$

or

$$\exists i, \left| |\mathrm{cf}_{x_i^m}(g)| - \frac{|\mathrm{cf}_{x_i^{m-k}}(s)|(|\mathrm{cf}_{x_i^n}(f)| + |\mathrm{cf}_{x_i^m}(g)|)}{|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)|} \right| > \varepsilon$$

in the case of $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| \neq 0$, or if

$$\exists i, \max\{|\mathrm{cf}_{x_i^n}(f)|, |\mathrm{cf}_{x_i^m}(g)|\} > \varepsilon$$

in the case of $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| = 0$, where $\mathrm{cf}_{x_i^d}(p)$ denotes the coefficient of term x_i^d of polynomial $p(\vec{x})$.

Proof. Let $h(\vec{x})$ be any approximate GCD of $f(\vec{x})$ and $g(\vec{x})$ satisfying the equation (1). In the case of $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| = 0$, we have $\deg_{x_i}(th) < \deg_{x_i}(f)$ and $\deg_{x_i}(th) < \deg_{x_i}(g)$. Hence, $\Delta_f(\vec{x})$ and $\Delta_g(\vec{x})$ must have $\mathrm{cf}_{x_i^n}(f)$ and $\mathrm{cf}_{x_i^m}(g)$, respectively. Therefore, for the given tolerance ε, there is not any polynomial $h(\vec{x})$ such that $\| \Delta_f(\vec{x}) \| \leq \varepsilon$ and $\| \Delta_g(\vec{x}) \| \leq \varepsilon$, if $\max\{|\mathrm{cf}_{x_i^n}(f)|, |\mathrm{cf}_{x_i^m}(g)|\} > \varepsilon$. In the case of $|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)| \neq 0$, for any $i \in \{1,\dots,\ell\}$, $h(\vec{x})$ must satisfy the following inequalities.

$$\begin{aligned} ineq_f &:= ||\mathrm{cf}_{x_i^n}(f)| - |\mathrm{cf}_{x_i^{n-k}}(t)\mathrm{cf}_{x_i^k}(h)|| \leq \varepsilon, \\ ineq_g &:= ||\mathrm{cf}_{x_i^m}(g)| - |\mathrm{cf}_{x_i^{m-k}}(s)\mathrm{cf}_{x_i^k}(h)|| \leq \varepsilon. \end{aligned}$$

$\max\{ineq_f, ineq_g\}$ is minimum if $\mathrm{cf}_{x_i^k}(h)$ satisfies

$$\mathrm{cf}_{x_i^k}(h) = \frac{|\mathrm{cf}_{x_i^n}(f)| + |\mathrm{cf}_{x_i^m}(g)|}{|\mathrm{cf}_{x_i^{n-k}}(t)| + |\mathrm{cf}_{x_i^{m-k}}(s)|}$$

which is the solution of $ineq_f + ineq_g = 0$. Substituting the above $\mathrm{cf}_{x_i^k}(h)$ for that of $ineq_f$ and $ineq_g$ proves the criterion. □

We have generated 100 pairs of bivariate polynomials of total degree randomly chosen from $[2,10]$, having their GCD of total degree randomly chosen from $[1,5]$, coefficients of their factors randomly chosen from $[-100,100]$ and added noise polynomials of the same total degree, whose coefficients randomly chosen from $[-10,10]$. Table 1 shows the result where "#success" denotes the number of pairs for which our algorithm found their

	without any criteria	with criteria 1, 2 and 3
average time (#success)	32.1622 sec. (94/100)	9.10895 sec. (94/100)
average time (#failure)	11.2494 sec. (6/100)	5.06832 sec. (6/100)

Table 1. Result of benchmark

approximate GCD. With the above criteria 1, 2 and 3, our algorithm becomes about 3 times faster than the original version.

According to the above lemmas and the result of numerical examples, we insert the following step **6–1** just after the step **6** in Algorithm 1 and rename it Algorithm **2**.

6–1. <u>choose another vector if the chosen vector satisfies any criteria 1, 2 or 3</u>

3.1. Numerical Experiments and Remarks

To see the efficiency of Algorithm 2, we have generated 100 pairs of polynomials satisfying the following conditions, and compute their approximate GCDs by the algorithm with $\varepsilon = 10$ in the step **1** and doing the steps **5**–**12** only once. We note that all the experiments have been computed on Linux with Intel Core 2 6700 2.66GHz and 2GB memory, by our implementation on *Mathematica* 6.0, and we use the max norm for polynomials though the algorithm works for other norms.

No.1) A pair of bivariate polynomials of total degree randomly chosen from $[2, 10]$, having their GCD of total degree randomly chosen from $[1, 5]$, coefficients of their factors randomly chosen from $[-100, 100]$ and added noise bivariate polynomials of the same total degree, whose coefficients randomly chosen from $[-10, 10]$.

No.2) A pair of polynomials in x_1, x_2, x_3 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 polynomials in x_1, x_2, x_3 of the same total degree, whose coefficients randomly chosen from $[-10, 10]$.

No.3) A pair of bivariate polynomials of total degree randomly chosen from $[2, 12]$ and $[2, 11]$, having their GCD of total degree randomly chosen from $[1, 5]$, coefficients of their factors randomly chosen from $[-100, 100]$ and added noise bivariate polynomials of the same total degree, whose coefficients randomly chosen from $[-10, 10]$.

No.4) A pair of polynomials in x_1, x_2, x_3 of total degree randomly chosen from $[2, 8]$ and $[2, 7]$, having their GCD of total degree randomly chosen from $[1, 4]$, coefficients of their factors randomly chosen from $[-100, 100]$ and added noise polynomials in x_1, x_2, x_3 of the same total degree, whose coefficients randomly chosen from $[-10, 10]$.

	#success (avg. time) / search for $r = 0$	#success (avg. time) / search for all r
No.1	90/100 (2.06262 sec.)	95/100 (0.148304 sec.)
No.2	91/100 (1.73053 sec.)	96/100 (0.151634 sec.)
No.3	87/100 (4.26505 sec.)	94/100 (0.317935 sec.)
No.4	94/100 (30.8244 sec.)	99/100 (0.344426 sec.)

Table 2. Result of benchmark

Table 2 shows the results which suggest that we should search for all r. One may think that searching for all r (including $r = 0$) must be slower than searching for only $r = 0$ since the null space for $r = 0$ includes all the null vectors at one time. However, for large r (note that the algorithm starts with a large r), the size of $\mathcal{L}(f, g, r, c)$ is small and the number of irrelevant vectors which can not be detected by the early termination criteria is also small and the step 6–1 detects that the next steps can not find any approximate GCD and drastically decreases the number of $\mathcal{H}(f, g, r, c, t, s)$ which we should construct.

Moreover, we show the interesting example below. The underlined figure is the dominant part of the noise polynomials, showing that the tolerance of approximate GCD must be less than or equal to 10 in the max norm.

$$\begin{aligned} f(\vec{x}) &= (-11x_3^5 + 7x_1^2x_2^3 + 12)(17x_1^3x_3^2 - 9x_1x_2 + 7) + \underline{10}x_1x_2 - 7, \\ g(\vec{x}) &= (-11x_3^5 + 7x_1^2x_2^3 + 12)(21x_2^2x_3^2 + 7x_1^2x_2 - 8) - 9x_2^2x_3^3. \end{aligned}$$

Algorithm 2 computes the following approximate GCD and cofactors of $f(\vec{x})$ and $g(\vec{x})$.

$$\begin{gathered} appgcd(f, g) = h(\vec{x}) = -23x_3^5 + 15x_1^2x_2^3 + 25, \\ t(\vec{x}) = 8x_1^3x_3^2 - 4x_1x_2 + 3, \quad s(\vec{x}) = 10x_2^2x_3^2 + 3x_1^2x_2 - 4. \end{gathered}$$

In this case, we have $\varepsilon = 9$ while we expected $\varepsilon = 10$. Hence, $(-11x_3^5 + 7x_1^2x_2^3 + 12)$ is not the nearest approximate GCD in the max norm, and the above $h(\vec{x})$ computed by Algorithm 2 is closer to, or must be the nearest one.

4. GCD of Several Polynomials and Its Application

Our algorithm is applicable for simplifying mathematical expressions. We note that this idea comes from the discussion at the poster session of ISSAC 2008. The author wishes to thank several participants especially Adam Strzebonski, for their suggestions. For example, let $p(x, y, z)$ be the following polynomial which is primitive and irreducible over $\mathbb{Z}$.

$$\begin{aligned} p(x, y, z) &= \left(-2yx^2 + 2x^2 + y^2x - 2yx + 3x - y + 1\right) z^2 \\ &\quad + \left(-4yx^2 + 4x^2 + 2y^2x - 4yx + y - 1\right) z + y - 2x. \end{aligned}$$

By computing an approximate GCD over integers of coefficient polynomials f_0, f_1, f_2.

$$\begin{aligned} f_0(x, y) &= -2yx^2 + 2x^2 + y^2x - 2yx + 3x - y + 1, \\ f_1(x, y) &= -4yx^2 + 4x^2 + 2y^2x - 4yx + y - 1, \\ f_2(x, y) &= y - 2x, \\ appgcd(f_0, f_1, f_2) &= 2x - y + 1. \end{aligned}$$

We get

$$p(x, y, z) = (2x - y + 1)\left((-yx + x + 1)z^2 + (-2yx + 2x - 1)z - 1\right) + 1.$$

In this case, we have to compute an approximate GCD of three polynomials while Algorithm 2 is only for two polynomials. To extend our algorithm to several polynomials, we use the following generalized subresultant mapping $\mathcal{S}_r(f_0, \ldots, f_k)$ of polynomials $f_0(\vec{x}), \ldots, f_k(\vec{x})$ of order r (see also Rupprecht (1999)).

$$\mathcal{S}_r(f_0, \ldots, f_k) : \begin{cases} \prod_{i=0}^{k} \mathcal{P}_{n_i-r-1} \rightarrow \prod_{i=1}^{k} \mathcal{P}_{n_0-n_i-r-1} \\ \qquad \begin{pmatrix} u_0 \\ \vdots \\ u_k \end{pmatrix} \mapsto \begin{pmatrix} u_1 f_0 + u_0 f_1 \\ \vdots \\ u_k f_0 + u_0 f_k \end{pmatrix} \end{cases}$$

where $n_i = \mathrm{tdeg}(f_i)$ $(i = 0, \ldots, k)$. This mapping can be expressed by the following Sylvester subresultant matrix $Syl_r(f_0, \ldots, f_k)$ and we can compute candidate cofactors $u_0, \ldots, u_k$ and approximate GCDs by the LLL algorithm as in Algorithm 2.

$$Syl_r(f_0, \ldots, f_k) = \begin{pmatrix} C_{n_0-r}(f_1) & C_{n_1-r}(f_0) & \vec{0} & \cdots & \vec{0} \\ C_{n_0-r}(f_2) & \vec{0} & C_{n_2-r}(f_0) & \cdots & \vec{0} \\ \vdots & & & \ddots & \vdots \\ C_{n_0-r}(f_k) & \vec{0} & \cdots & \vec{0} & C_{n_k-r}(f_0) \end{pmatrix}.$$

Moreover, matrices $\mathcal{L}(f, g, r, c)$ and $\mathcal{H}(f, g, r, c, t, s)$ are also extended to several polynomials as follows.

$$\mathcal{L}(f_i s, r, c) = (E_{\beta_{n_0-1,r}+\cdots+\beta_{n_k-1,r}} \mid c \cdot Syl_r(f_0, \ldots, f_k)^t),$$

$$\mathcal{H}(f_i s, r, c, u_i s) = \left(E_{\beta_{r+1,0}+1} \left| \begin{matrix} c \cdot \overrightarrow{f_0}^t & c \cdot \overrightarrow{f_1}^t & \cdots & c \cdot \overrightarrow{f_k}^t \\ c \cdot C_{r+2}(-u_0)^t & c \cdot C_{r+2}(u_1)^t & \cdots & c \cdot C_{r+2}(u_k)^t \end{matrix} \right. \right).$$

The sizes of $\mathcal{L}(f_i s, r, c)$ and $\mathcal{H}(f_i s, r, c, u_i s)$ are $(\sum_{i=0,\ldots,k} \beta_{n_i-1,r}) \times (\sum_{i=0,\ldots,k} \beta_{n_i-1,r} + \sum_{i=1,\ldots,k} \beta_{n_0+n_i-1,r})$ and $(\beta_{r+1,0} + 1) \times (\sum_{i=0,\ldots,k} \beta_{n_i,0} + \beta_{r+1,0} + 1)$, respectively. We have almost same lemmas as Lemma 4 and Lemma 7 with $c_B = 2^{(\sum_{i=0,\ldots,k} \beta_{n_i-1,r}-1)/2} \sqrt{\sum_{i=0,\ldots,k} \beta_{n_i-1,r}} B$ and $c_H = 2^{\beta_{r+1,0}/2} \sqrt{\beta_{r+1,0} + 1} B + 1$. The early termination criteria 1, 2 and 3 are also easily extended to several polynomials. We just apply Criteria 1, 2 and 3 for each pair of all the combinations of $f_0(\vec{x})$ and other polynomials.

5. Various Remarks

To get a reduced lattice basis vectors, our implementation uses *Mathematica*'s built-in function "LatticeReduce" which is Storjohann's variant (Storjohann, 1996) of the LLL algorithm. Though this algorithm is faster than the original LLL algorithm, it is not fast since there are several variants using floating-point numbers. Hence, our algorithm becomes much faster if we use such a fast algorithm to get lattice basis vectors. For example, ntl-5.4.2 by Victor Shoup has a function LLL_FP which is a variant of Schnorr and Euchner's algorithm (Schnorr and Euchner, 1991).

One may be interested in the concrete condition (upper or lower bounds of tolerance) for approximate GCD over integers. However, our approach does not answer to this question since our algorithm only searches for approximate cofactors $s(\vec{x})$ and $t(\vec{x})$ satisfying $\|s(\vec{x})f(\vec{x})+t(\vec{x})g(\vec{x})\|\approx 0$ which is not a necessary condition of that $f(\vec{x})$ and $g(\vec{x})$ have an approximate GCD over integers. For the following (rare) counter condition:

$$\|s(\vec{x})f(\vec{x})+t(\vec{x})g(\vec{x})\| = \|s(\vec{x})\Delta_f(\vec{x})+t(\vec{x})\Delta_g(\vec{x})\| \gg 0,$$

it is difficult to find any appropriate GCD and the algorithm may find polynomial $h(\vec{x})$ such that $\|f(\vec{x})-t(\vec{x})h(\vec{x})\|\gg 0$ and $\|g(\vec{x})-s(\vec{x})h(\vec{x})\|\gg 0$. Of course, the lattice generated by row vectors of $\mathcal{L}(f,g,r,c)$ includes the vector corresponding to the approximate cofactors but this vector is not short and we can not find it by the LLL algorithm.

Conversely, our algorithm can find any approximate GCD if candidate cofactors $s(\vec{x})$ and $t(\vec{x})$ correspond to the short vector by the LLL algorithm. Moreover, according to our experiments, $s(\vec{x})f(\vec{x})+t(\vec{x})g(\vec{x})$ may have terms whose coefficients are very small even if $\|s(\vec{x})f(\vec{x})+t(\vec{x})g(\vec{x})\|\gg 0$. In such a situation, multiplying a column vector of $\mathcal{L}(f,g,r,c)$ by a larger integer gives an appropriate row vector and an approximate GCD. In fact, 5.9% of failure cases become to have approximate GCDs computed, though this technique is time-consuming even for univariate polynomials.

References

Backes, W., Wetzel, S., 2002. Heuristics on lattice basis reduction in practice. ACM J. Exp. Algorithmics 7, 21 pp. (electronic), fourth Workshop on Algorithm Engineering (Saarbrücken, 2000).

Christou, D., Mitrouli, M., 2005. 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.

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

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

Emiris, I. Z., Galligo, A., Lombardi, H., 1996. Numerical univariate polynomial GCD. In: The mathematics of numerical analysis (Park City, UT, 1995). Vol. 32 of Lectures in Appl. Math. Amer. Math. Soc., Providence, RI, pp. 323–343.

Emiris, I. Z., Galligo, A., Lombardi, H., 1997. Certified approximate univariate GCDs. J. Pure Appl. Algebra 117/118, 229–251, algorithms for algebra (Eindhoven, 1996).

Gao, S., Kaltofen, E., May, J., Yang, Z., Zhi, L., 2004. Approximate factorization of multivariate polynomials via differential equations. In: ISSAC 2004. ACM, New York, pp. 167–174.

Gel′fond, A. O., 1960. Transcendental and algebraic numbers. Translated from the first Russian edition by Leo F. Boron. Dover Publications Inc., New York.

Howgrave-Graham, N., 2001. Approximate integer common divisors. In: Cryptography and lattices (Providence, RI, 2001). Vol. 2146 of Lecture Notes in Comput. Sci. Springer, Berlin, pp. 51–66.

Kaltofen, E., Yang, Z., Zhi, L., 2006. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In: ISSAC 2006. ACM, pp. 169–176.

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

Karmarkar, N. K., Lakshman, Y. N., 1998. On approximate GCDs of univariate polynomials. J. Symbolic Comput. 26 (6), 653–666, symbolic numeric algebra for polynomials.

Laidacker, M. A., 1969. Another theorem relating Sylvester's matrix and the greatest common divisor. Math. Mag. 42, 126–128.

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

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

Mignotte, M., 1974. An inequality about factors of polynomials. Math. Comp. 28, 1153–1157.

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

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

Pan, V. Y., 1998. 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). ACM, New York, pp. 68–77.

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

Rössner, C., Seifert, J.-P., 1996. The complexity of approximate optima for greatest common divisor computations. In: Algorithmic number theory (Talence, 1996). Vol. 1122 of Lecture Notes in Comput. Sci. Springer, Berlin, pp. 307–322.

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

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

Schnorr, C. P., Euchner, M., 1991. Lattice basis reduction: improved algorithms and solving subset sum problems. In: Proceedings of Fundamentals of Computation Theory, FCT'91, Ed. L. Budach, Springer LNCS 529. pp. 68–85.

Schnorr, C.-P., Euchner, M., 1994. Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Programming 66 (2, Ser. A), 181–199.

Schönhage, A., 1985. Quasi-gcd computations. J. Complexity 1 (1), 118–137.

Storjohann, A., 1996. Faster Algorithms for Integer Lattice Basis Reduction. Technical Report 249. Zurich, Switzerland: Department Informatik, ETH.

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

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

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

Zeng, Z., Dayton, B. H., 2004. The approximate GCD of inexact polynomials. II. A multivariate algorithm. In: ISSAC 2004. ACM, New York, pp. 320–327.

Zhi, L., 2003. Displacement structure in computing approximate GCD of univariate polynomials. In: Computer mathematics. Vol. 10 of Lecture Notes Ser. Comput. World Sci. Publ., River Edge, NJ, pp. 288–298.

Zhi, L., Noda, M.-T., 2000a. Approximate GCD of multivariate polynomials. Sūrikaisekikenkyūsho Kōkyūroku (1138), 64–76, research on the theory and applications of computer algebra (Japanese) (Kyoto, 1999).

Zhi, L. H., Noda, M.-T., 2000b. Approximate GCD of multivariate polynomials. In: Computer mathematics (Chiang Mai, 2000). Vol. 8 of Lecture Notes Ser. Comput. World Sci. Publ., River Edge, NJ, pp. 9–18.

The preliminary implementation on *Mathematica* 6.0, of our algorithm introduced in this paper can be found at the following URL:
`http://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/issac08plus.nb`