Symbolic-Numeric Computation
D. Wang and L. Zhi, Eds.
Trends in Mathematics, 55–68

Computing Approximate GCD of Multivariate Polynomials

Masaru Sanuki

Abstract. A new algorithm for computing the approximate GCD of multivariate polynomials is proposed by modifying the PC-PRS algorithm for exact GCD. We have implemented the new algorithm and compared it by typical examples with (approximate) PRS, (approximate) EZ-GCD algorithms and two new algorithms based on SVD. The experiment shows a good performance of our algorithm.

Keywords. Approximate GCD, polynomial remainder sequence (PRS), power-series coefficient PRS (PC-PRS) algorithm, extended Zassenhaus GCD (EZ-GCD) algorithm, singular value decomposition (SVD).

1. Introduction

Let $\mathbb{C}[x, u_1, \ldots, u_\ell]$ be the ring of polynomials over the complex number field $\mathbb{C}$ in the main variable x and sub-variables $u_1, \ldots, u_\ell$. We abbreviate $\mathbb{C}[x, u_1, \ldots, u_\ell]$ to $\mathbb{C}[x, u]$. Let a given polynomial $F(x, u) \in \mathbb{C}[x, u]$ be expressed as $F(x, u) = f_m(u)x^m + f_{m-1}(u)x^{m-1} + \cdots + f_0(u),\ f_m \neq 0$. By $\deg(F), \mathrm{lc}(F)$ and $\mathrm{tdeg}_u(f_i)$, we denote the degree and the leading coefficient of F w.r.t. the main variable x, and the total-degree of f_i w.r.t. $u_1, \ldots, u_\ell$, respectively; if $T = cu_1^{e_1} \cdots u_\ell^{e_\ell}, c \in \mathbb{C}$, then $\mathrm{tdeg}_u(T) = e_1 + \cdots + e_\ell$. By $\mathrm{tdeg}_u(F)$ and $\mathrm{tdeg}(F)$, we denote the total-degree of F w.r.t. $u_1, \ldots, u_\ell$ and w.r.t. $x, u_1, \ldots, u_\ell$, respectively, i.e., $\mathrm{tdeg}_u(F) = \max\{\mathrm{tdeg}_u(f_m), \ldots, \mathrm{tdeg}_u(f_0)\}$. By $||F||$, we denote the norm of polynomial F; we use the infinity norm in this paper: $||F|| = \max\{||f_m||, \ldots, ||f_0||\}$. By Normalize$(F)$, we denote the normalization of the polynomial F, i.e. the scale transformation $F \to F' = \eta F$, $\eta \in \mathbb{C}$, so that we have $||F'|| = 1$. By appGCD(A, B) and $\gcd(A, B; \varepsilon)$, we denote the approximate greatest common divisor (appGCD) of A and B and of tolerance ε, respectively. By $\mathrm{cont}(F; \varepsilon)$ and $\mathrm{pp}(F; \varepsilon)$, we denote the content of F of tolerance ε and the primitive part of F of tolerance ε, respectively, w.r.t. x, i.e., $\mathrm{cont}(F; \varepsilon) = \gcd\big(f_m, \gcd(f_{m-1}, \ldots, f_0; \varepsilon); \varepsilon\big)$ and $\mathrm{pp}(F; \varepsilon) = F/\mathrm{cont}(F; \varepsilon)$.

Zhi and Noda [ZN00] compared three algorithms for the appGCD of multivariate polynomials $A(x,u)$ and $B(x,u)$ experimentally, the polynomial remainder sequence (PRS) algorithm by Ochi *et al.* [ONS91], the EZ-GCD algorithm by Moses and Yun [MY73] with enhancement by Wang [Wan80], and a modular algorithm by Corless *et al.* [CGTW95]. They concluded from several experiments as follows. (a) The PRS algorithm is useful for the appGCD of polynomials of small degrees w.r.t. each variable. It is, however, quite inefficient when A and B are of large degrees. Furthermore, the coefficient size grows exponentially w.r.t. the number of variables. (b) EZ-GCD algorithm is fast, but it often causes large cancellation errors. Furthermore, if initial factors $A^{(0)}(x)$ and $B^{(0)}(x)$ of the Hensel construction have a close root, then multivariate Hensel construction becomes unstable [SY98]. (c) The Modular algorithm is often unstable.

At ISSAC 2004, two new methods for computing appGCD of multivariate polynomials were proposed. Gao *et al.* [GKMYZ04] proposed their appGCD algorithm as one part of an approximate bivariate factorization algorithm using Gao's factorization algorithm. Zeng and Dayton [ZD04] gave a nearly identical GCD algorithm that has an additional Gauss-Newton refinement step. In both papers, the authors proposed generalized Sylvester matrices and determined appGCDs by computing nearest singular matrices of the Sylvester matrices.

As for exact GCD, Sasaki and Suzuki [SS92] presented the so-called Power-series Coefficient PRS (PC-PRS) algorithm. The PC-PRS algorithm is a modification of the PRS algorithm, using truncated power series for the coefficients, and it is as fast as the EZ-GCD algorithm. In this paper, we modify the PC-PRS algorithm for calculating the appGCD and test it by several examples; the results are very good. We also compare the PC-PRS algorithm with the PRS, EZ-GCD and SVD-based algorithms for computing appGCD in Maple, MATLAB and GAL (see Section 4). The comparison shows a good performance of the PC-PRS algorithm. However, the PC-PRS algorithm becomes unstable in some case, and we discuss the case.

2. Approximate PC-PRS Algorithm

Let P_1 and P_2 be primitive polynomials in $\mathbb{C}[x,u]$ and $(P_1, P_2, P_3, \ldots, P_k \neq 0, P_{k+1} = 0)$ be a PRS. Let $P_k = C(u)G(x,u)$ where $C(u)$ is the content of P_k; then $G(x,u)$ is the GCD of P_1 and P_2. Usually, $\mathrm{tdeg}_u(G)$ is not large, but $\mathrm{tdeg}_u(C)$ is very large. If P_k and C are given, we can compute G by dividing P_k by C. In this division, we need not treat all the terms of P_k and C. For example, suppose $\mathrm{tdeg}_u(P_k) = 100$ and $\mathrm{tdeg}_u(C) = 90$; then $\mathrm{tdeg}_u(G) = 10$. In this case, the computation of G requires only terms of highest 10 exponents or terms of lowest 10 exponents of P_k and C. In the PC-PRS algorithm, we use only terms of lowest some exponents, and cut-off all the other higher terms, which makes the computation quite fast.

Let e be an upper bound of $\mathrm{tdeg}_u(G)$; we discard all the terms of total-degrees greater than e. In order to cut-off terms simply, we introduce the total-degree variable t for the sub-variables $u_1, \ldots, u_\ell$ by transformation $u_i \mapsto tu_i$ $(i = 1, \ldots, \ell)$. Let $\tilde{P}_1 = P_1(x, tu), \tilde{P}_2 = P_2(x, tu) \in \mathbb{C}\{tu\}[x]$, where $\mathbb{C}\{tu\}$ is the power-series ring over $\mathbb{C}$. We compute a PRS with power-series coefficients, or PC-PRS, $(\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_k \not\equiv 0, \tilde{P}_{k+1} \equiv 0)$ by truncating terms of total-degree greater than e; hence $\tilde{P}_i \equiv P_i \pmod{t^{e+1}}$ $(i = 1, 2, \ldots)$. Actually, $\tilde{P}_i$ is generated as follows:

$$\begin{cases} \beta_i \tilde{P}_{i+1} \equiv (\alpha_i \tilde{P}_{i-1} - Q_i \tilde{P}_i) / \max\{||\alpha_i||, ||Q_i||\} \pmod{t^{e+1}}, \\ \alpha_i = \mathrm{lc}(\tilde{P}_i)^{d_i+1}, \quad d_i = \deg(\tilde{P}_{i-1}) - \deg(\tilde{P}_i), \end{cases}$$

where β_i is determined by the reduced-PRS or the subresultant PRS algorithm. By modifying the PC-PRS algorithm, we obtain the following approximate PC-PRS algorithm. (The algorithm does not consider the case of small leading coefficients. The treatment of PRS with small leading coefficients is now an open problem.)

Algorithm 1 (approximate PC-PRS algorithm).

Input:	Polynomials $P_1, P_2 \in \mathbb{C}[x, u]$ and a small number ε.
Output:	$G = \gcd(P_1, P_2; \varepsilon)$.
STEP 1:	$g := \gcd(\mathrm{lc}(P_1), \mathrm{lc}(P_2); \varepsilon)$.
	$E := \min_{i=1,2}\{\mathrm{tdeg}_u(P_i) + \mathrm{tdeg}_u(g) - \mathrm{tdeg}_u(\mathrm{lc}(P_i))\}$.
STEP 2:	Calculate PRS $(\tilde{P}_3, \ldots, \tilde{P}_k, \tilde{P}_{k+1}, ||\tilde{P}_{k+1}||/||\tilde{P}_k|| \le \varepsilon)$.
	/* cut-off higher degree E terms */
	if $\deg(\tilde{P}_k) = 0$ then return 1 else $\tilde{P}_k := \mathrm{Normalize}(\tilde{P}_k)$.
	$\tilde{P} := g\tilde{P}_k / \mathrm{lc}(\tilde{P}_k)$. /* power-series division */
	$G := \mathrm{pp}(\tilde{P}; \varepsilon)$.
	if $||\mathrm{rem}(P_1, G)|| \le \varepsilon$ and $||\mathrm{rem}(P_2, G)|| \le \varepsilon$
	then return G else return 1.
	end.

3. Other Algorithms

3.1. EZ-GCD Algorithm

The EZ-GCD algorithm may be described as follows.

1. Choose a lucky expansion point $s \in \mathbb{C}^\ell$ and put $I = (u_1 - s_1, \ldots, u_\ell - s_\ell)$.
2. Put $A^{(0)} \equiv A$, $B^{(0)} \equiv B \pmod{I}$, and compute $G^{(0)} = \gcd(A^{(0)}, B^{(0)})$.
3. Find a and $b \in \mathbb{C}$ such that $\gcd(G^{(0)}, H^{(0)}) = 1$ where $P^{(0)} = aA^{(0)} + bB^{(0)}$ and $H^{(0)} = P^{(0)}/G^{(0)}$, and put $P = aA + bB$.
4. Compute polynomials $U_i(x)$ and $V_i(x)$ $(i = 0, 1, \ldots, \deg(P^{(0)}))$ such that
$$U_i(x)G^{(0)} + V_i(x)H^{(0)} = x^i.$$
5. Perform the Hensel construction to obtain polynomials $G^{(k)}$ and $H^{(k)}$:
$$P(x, u) \equiv G^{(k)}(x, u)H^{(k)}(x, u) \pmod{I^{k+1}},$$

where ($n = \deg(P^{(0)})$ below)

$$G^{(k)} = G^{(k-1)} + \sum_{i=0}^{n} V_i f_i^{(k)}, \quad H^{(k)} = H^{(k-1)} + \sum_{i=0}^{n} U_i f_i^{(k)},$$

with

$$F^{(k)} \equiv P - G^{(k-1)}H^{(k-1)} = f_n^{(k)} x^n + \cdots + f_0^{(k)} \pmod{I^{k+1}}.$$

We compute $U_i(x)$ and $V_i(x)$ by the extended Euclidean algorithm.

6. If $G^{(k)}|A$ and $G^{(k)}|B$ then return $G^{(k)}$, else choose another expansion point $s' \in \mathbb{C}^\ell$ and go to step 1.

3.2. SVD-Based Algorithms

The algorithms of Gao *et al.* [GKMYZ04] and Zeng-Dayton [ZD04] for computing appGCD of multivariate polynomials are based on singular value decomposition (SVD) and are very similar to each other. Let $f, g \in \mathbb{C}[x_1, \ldots, x_\ell]$, $f_1 = f/\gcd(f, g)$ and $g_1 = g/\gcd(f, g)$. Then all the solutions $u, v \in \mathbb{C}[x_1, \ldots, x_\ell]$ of the equation

$$uf + vg = 0 \tag{3.1}$$

are of the form

$$u = g_1 q, \quad v = -f_1 q, \tag{3.2}$$

where $q \in \mathbb{C}[x_1, \ldots, x_\ell]$ (Lemma 2.1 in [GKMYZ04]). From the equation (3.1), we can derive a linear system for the coefficients of u and v, and we can obtain solutions $u = g_1$ and $v = -f_1$ by solving the linear system.

Let S_k and $S_{\mathbf{k}}$ be Gao *et al.*'s k-th generalized Sylvester matrix and Zeng-Dayton's $\mathbf{k}$-th generalized Sylvester matrix, respectively. Then $\mathrm{tdeg}(\gcd(f, g)) = k$ (or $\mathrm{ldeg}(\gcd(f, g)) = \mathbf{k}$ (for ldeg, see Sect. 3.2.2)) if and only if S_k (or $S_{\mathbf{k}}$) is rank-deficient by one with its nullspace being spanned by $[\mathbf{v}, -\mathbf{u}]^T$, where $\mathbf{u}$ and $\mathbf{v}$ are coefficient vectors corresponding to u and v, respectively. Then, the condition of degree of GCD tells us that $u = g_1$ and $v = -f_1$. Gao *et al.*'s and Zeng-Dayton's algorithms are summarized as follows.

1. Determine k (or $\mathbf{k}$), the total-degree (or the ℓ-degree) w.r.t. $x_1, \ldots, x_\ell$ of the appGCD(f, g) in one of the following two ways:
   a) Computing the degrees of the GCDs of several random univariate projections of f and g, and look for the numerical rank of the corresponding univariate Sylvester matrices.
   b) (Gao *et al.*'s algorithm only) From $S = S_1(f, g)$ where $f, g \in \mathbb{C}[x_1, \ldots, x_\ell]$ with $\mathrm{tdeg}(u) < \mathrm{tdeg}(g)$ and $\mathrm{tdeg}(v) < \mathrm{tdeg}(f)$, find the largest gap in the singular values of S and infer the degree from the numerical rank of S.
2. For the k-th (or $\mathbf{k}$-th) generalized Sylvester matrix S_k (or $S_{\mathbf{k}}$), compute a basis of the nullspace by computing the singular vector corresponding to the smallest singular value of S_k (or $S_{\mathbf{k}}$).

3. Find a $G = \gcd(f, g)$, the approximate quotient of f and v (or g and u); alternatively minimize $||g - Gu||_2^2 + ||f + Gv||_2^2$, by using a least-square algorithm.

We note that the generalized Sylvester matrices are different in Gao *et al.*'s and Zeng-Dayton's papers. The size of S_k is smaller than the size of $S_{\mathbf{k}}$ (see Example 5). Also, we note that Gao *et al.*'s algorithm does not require tolerance ε, but Zeng-Dayton's algorithm requires tolerance ε because of determining degree of univariate polynomial GCDs.

3.2.1. Gao et al.'s k-th Generalized Sylvester Matrix. Let $\mathrm{tdeg}(f) = m$, $\mathrm{tdeg}(g) = n$, and $\beta(k, \ell) = \binom{k+\ell}{\ell}$; number of all the different terms $x_1^{i_1} \cdots x_\ell^{i_\ell}$ with $i_1 + \cdots + i_\ell \leq k$. Then, Gao *et al.*'s k-th Sylvester matrix

$$S_k \in \mathbb{C}^{\beta(m+n-k,\ell)\times(\beta(m-k,\ell)+\beta(n-k,\ell))}. \tag{3.3}$$

3.2.2. Zeng-Dayton's k-th Generalized Sylvester Matrix. For $f \in \mathbb{C}[x_1, \ldots, x_\ell]$, let $m_i = \deg_{x_i}(f)$ $(i = 1, \ldots, \ell)$. By $\mathrm{ldeg}(f)$, or ℓ–degree of f, we denote the ℓ–tuple $\mathbf{m} = [m_1, \ldots, m_\ell]$. By $\nu(\mathbf{m})$, we denote the number of all the different terms $x_1^{i_1} \cdots x_\ell^{i_\ell}$ with $i_j \leq m_j$ $(1 \leq j \leq \ell)$ i.e., $\nu(\mathbf{m}) = (m_1 + 1)(m_2 + 1) \cdots (m_\ell + 1)$. In Zeng-Dayton's algorithm, every polynomial is transformed to a coefficient vector; if $f = \sum_{i_1+\cdots i_\ell \leq \nu(\mathbf{m}),\ i_j \leq m_j} f_s x_1^{i_1} \cdots x_\ell^{i_\ell}$ with $s = 1 + \sum_{k=1}^{\ell} \left[i_k \prod_{j=1}^{k-1} (m_j + 1) \right]$, then

$$\Psi_{\mathbf{m}} : f \mapsto \mathbf{f} = [f_1, f_2, \ldots, f_{\nu(\mathbf{m})}]^T \in \mathbb{C}^{\nu(m)}.$$

Let $\mathbf{n} = [n_1, \ldots, n_\ell]$. By $C_{\mathbf{n}}(f) \in \mathbb{C}^{\nu(\mathbf{m}+\mathbf{n})\times\nu(\mathbf{m})}$, we denote a convolution matrix of f; the s-th column of $C_{\mathbf{n}}(f)$ is generated by $\Psi_{\mathbf{m}+\mathbf{n}}(fq_s)$, where $q_s = x_1^{i_1} \cdots x_\ell^{i_\ell}$ with $s = 1 + \sum_{k=1}^{\ell} \left[i_k \prod_{j=1}^{k-1} (n_j + 1) \right]$ $(1 \leq s \leq \nu(\mathbf{m}))$. Let $\mathrm{ldeg}(g) = \mathbf{n}$; then Zeng-Dayton's $\mathbf{k}$-th Sylvester matrix

$$S_{\mathbf{k}}(f, g) = [C_{\mathbf{m}-\mathbf{k}}(g) | C_{\mathbf{n}-\mathbf{k}}(f)] \in \mathbb{C}^{\nu(\mathbf{m}+\mathbf{n}-\mathbf{k})\times(\nu(\mathbf{m}-\mathbf{k})+\nu(\mathbf{m}-\mathbf{k}))}. \tag{3.4}$$

4. Numerical Examples

We have implemented the Approximate PC-PRS algorithm in Maple and GAL, and Zeng-Dayton's algorithm in MATLAB. GAL is an algebra system constructed by Sasaki and Kako, being equipped with *effective floating-point number* system which allows us to detect the cancellation errors fairly well but not exactly; see [KS97] for details (since the relative errors are set to 10^{-15} initially, the predicted values are about 10 times larger than the actual values). We use GAL to estimate cancellation errors of PRS, EZ-GCD and PC-PRS, and we use Maple and MATLAB to compare average CPU times for the PC-PRS and SVD-based algorithms. We have used Gao's code "multigcd" is Maple for Gao *et al.*'s algorithm [Kal04, Zhi04]. We compare our algorithm with Gao *et al.*'s algorithm in Maple and Zeng-Dayton's algorithm in MATLAB. Since the size of generalized Sylvester matrices is very large in most cases, we have implemented `sparse` in

MATLAB [Matlab]. Computations reported in this paper were performed on an Xeon (and Ultra SPARC-IIi) running at 3.05GHz (and 440MHz) under Windows XP (and Solaris 8), using Maple 9.5, MATLAB 7.1 (and GAL).

An Execution of Approximate PC-PRS Algorithm

Example 1. Let multivariate polynomials $A(x, y, z)$ and $B(x, y, z)$ be

$$A(x, y, z) = (x^2 + y^2 + 0.3z^3 - 1)^2(xy - 0.25) - 10^{-5}xyz,$$
$$B(x, y, z) = (x^2 + y^2 + 0.3z^3 - 1)(x - y)^3 - 10^{-5}(x + 1 - z).$$

The approximate PC-PRS algorithm, with $\varepsilon = 0.001$, works as follows.

- $g = \gcd(\mathrm{lc}(P_1), \mathrm{lc}(P_2); \varepsilon) = \gcd(y, 1; \varepsilon) = 1$.
- $E = 6$.
- Calculate PC-PRS:
  $\deg(\tilde{P}_1), \ldots, \deg(\tilde{P}_6)$ are $5, 5, 4, 3, 2, 1$, respectively.
  $||\tilde{P}_1||, \ldots, ||\tilde{P}_5||$ are $O(1)$ and $||\tilde{P}_6|| \leq O(\varepsilon)$.
- Normalization of P_5:
  $\tilde{P}_5 = \mathrm{Normalize}(\mathrm{P}_5) = 0.8799\cdots x^2y^4 + 0.0959\cdots x^2y^2z^3 - 0.1199\cdots x^2y^2 - 0.0060\cdots x^2z^3 + 0.0066\cdots x^2 - y^4 - 0.1380\cdots y^2z^3 + 0.1266\cdots y^2 + 0.0080\cdots z^3 - 0.0066\cdots$.
- Power-series division:
  $\tilde{P} = g\tilde{P}_5/\mathrm{lc}(\tilde{P}_5) = x^2 + 0.9999\cdots y^2 + 0.2999\cdots z^3 - 1.0000\cdots$.
- $G = \mathrm{pp}(\tilde{P}; \varepsilon) = \tilde{P}$.
- Check $||\mathrm{rem}(P_1, G)|| \leq \varepsilon$ and $||\mathrm{rem}(P_2, G)|| \leq \varepsilon$.
  $||\mathrm{rem}(P_1, G)|| = 0.0001 < 0.001$, $||\mathrm{rem}(P_2, G)|| = 0.00019 < 0.001$.
- Return G.

Example 2 (Comparison of PRS algorithm with PC-PRS algorithm). We compare the PRS and PC-PRS algorithms on the following $A(x, y, z)$ and $B(x, y, z)$:

$$A(x, y, z) = (x^2 + y^{2m} + 0.3z^{3m} - 1)^2(xy - 0.25) - 10^{-5}xyz^n,$$
$$B(x, y, z) = (x^2 + y^{2m} + 0.3z^{3m} - 1)(x - y^n)^3 - 10^{-5}(x + 1 - z).$$

Cancellation Errors by PRS and PC-PRS

In Tables 1 and 2, "#term" and "ErrMax" show numbers of terms and maximum relative errors in the coefficients $P_i (i = 1, \ldots, 6)$, respectively. "–Error–" means that the computation stopped by failure (in this case, error happened in the approximate division $\tilde{P}/\mathrm{cont}(\tilde{P}; \varepsilon)$ with tolerance ε).

Comparison of Computing Times

- Table 3: Average CPU times for $m = 1$ and $n = 1, \ldots, 6$.
  The total-degree of appGCD is the same for different n.
- Table 4: Average CPU times for $n = 1$ and $m = 1, \ldots, 4$.
  The total-degree of appGCD increases as m increases.

TABLE 1. Approximate PRS Algorithm

	#term	ErrMax(P_i)
P_1	21	1.00e-15
P_2	17	1.00e-15
P_3	27	3.00e-15
P_4	58	2.70e-14
P_5	166	1.20e-7
P_6		$O(0.0001)$
approx. GCD		— Error —

TABLE 2. Approximate PC-PRS Algorithm

	#term	ErrMax(P_i)
P_1	21	1.00e-15
P_2	17	1.00e-15
P_3	15	2.00e-15
P_4	19	8.00e-15
P_5	22	1.62e-9
P_6		$O(0.0001)$
approx. GCD		2.53e-13

TABLE 3. The average CPU times for $m = 1$ and $n = 1, \ldots, 6$ (sec)

$m = 1$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$
Approx. PRS	0.202	0.828	0.424	1.408	1.380	1.714
Approx. PC-PRS	0.016	0.016	0.012	0.016	0.014	0.012

TABLE 4. The average CPU times for $n = 1$ and $m = 1, \ldots, 4$ (sec)

$n = 1$	$m = 1$	$m = 2$	$m = 3$	$m = 4$
Approx. PRS	0.202	0.370	0.694	0.744
Approx. PC-PRS	0.016	0.026	0.036	0.058

We see that the approximate PC-PRS algorithm shows a very nice efficiency. The average CPU time for approximate PRS algorithm depends on total-degrees of A and B largely. On the other hand, the average CPU time for the approximate PC-PRS algorithm depends only on the total-degree of appGCD, and they do not depend much on other quantities.

Comparison of PC-PRS Algorithm with EZ-GCD Algorithm

Example 3. We compare the PC-PRS and EZ-GCD algorithms on the following $A(x, u)$ and $B(x, u)$:

$$\begin{cases} A(x,u) = (x^3 + u^4 + u^3 - u^2 + u + 1 + 0.001)(x^2 + xu + u^2 + 1), \\ B(x,u) = (x^3 + u^4 + u^3 - u^2 + u + 1 + 0.001)(x^2 + xu + 1). \end{cases}$$

1. Approximate PC-PRS Algorithm
   We obtain an appGCD(A, B) as follows:

$$0.999000999001x^3 + 0.999000999001u^4 + 0.999000999001u^3 \\ -0.999000999001u^2 + 0.999000999001u + 1.$$

   The maximum relative error of this appGCD is 2.00×10^{-15}.

2. EZ-GCD Algorithm
We choose the expansion point at $s = 4/2005$ and put $a = 1$ and $b = 1$; then

$G^{(0)} = 0.99701787928153x^3 + 1,$
$H^{(0)} = 2.0059820807004x^2 + 0.0040019592632428x + 2.0059860726797.$

$G^{(0)}$ and $H^{(0)}$ have mutually close roots, and the maximum relative errors of $G^{(0)}$ and $H^{(0)}$ are 2.51×10^{-10} and 2.51×10^{-10}, respectively. We obtain the following appGCD(A, B):

$$\begin{aligned}\tilde{G} = {} & 0.9970\cdots x^3 + 0.0181\cdots x^2u^4 - 0.0026\cdots x^2u^3 \\ & -0.0015\cdots x^2u^2 + 0.0019\cdots x^2u + 0.0177\cdots xu^4 - 0.0105\cdots xu^3 \\ & +0.0044\cdots xu^2 - 0.0019\cdots xu - 0.0306\cdots u^4 + 0.9942\cdots u^3 \\ & -0.9935\cdots u^2 + 0.9950\cdots u + 1.\end{aligned}$$

The maximum relative error of this appGCD by the EZ-GCD algorithm is 6.33×10^{-6}. We find that remainders of A and B by $\tilde{G}$ are $O(1)$.

Since we have chosen the expansion point s to be near a singular point (in this case, the origin is a singular point), the EZ-GCD algorithm causes large cancellation errors. We see that the approximate PC-PRS algorithm does not cause such a large cancellation error; it is independent of the singular point.

Example 4. We compare the PC-PRS and EZ-GCD algorithms on the following $A(x, u)$, $B(x, u)$ and $\tilde{B}(x, u)$:

$$\begin{cases} A(x,u) = (x^2 + u^2 + 1)(x^2 - u - 0.5)(x^2 + u + 0.1), \\ B(x,u) = (x^2 + u^2 + 1)(x + u^3 + u - 0.4)(\underline{0.001}x^2 + u + 1), \\ \tilde{B}(x,u) = (x^2 + u^2 + 1)(x + u^3 + u - 0.4)(\underline{0.0001}x^2 + u + 1). \end{cases}$$

1. Approximate PC-PRS Algorithm
We obtain an appGCD(A, B) as follows

$$x^2 + u^2 + 1.$$

The maximum relative error of this appGCD is 1.13×10^{-11}. However, we cannot obtain any appGCD$(A, \tilde{B})$: we can calculate the PRS of A and B as $(P_1 = A, P_2 = B, \ldots, P_5, P_6,\ ||P_6|| = O(0.001))$, but the PRS of A and $\tilde{B}$ is $(P_1 = A, P_2 = B, \ldots, P_5,\ ||P_5|| = O(0.001))$. Since $||\mathrm{lc}(B)||$ is smaller than the norms of other coefficients of B, the approximate PC-PRS algorithm causes large cancellation errors.
2. EZ-GCD Algorithm
We choose the expansion point at $s = 0$ and put $a = 1$ and $b = 1$; then

$G^{(0)} = x^2 + 0.99999999999811,$
$H^{(0)} = x^4 + 0.001x^3 - 0.40039999999811x^2 + 1.0x - 0.45000000000264.$

We obtain appGCD(A, B) and appGCD$(A, \tilde{B})$ as follows

$$x^2 + u^2 + 1.$$

The maximum relative errors of appGCD(A, B) and appGCD$(A, \tilde{B})$ by EZ-GCD algorithm are 1.13×10^{-11} and 1.13×10^{-11}, respectively.

We see that the approximate PC-PRS algorithm shows a very good efficiency. We also see that, in some cases, the EZ-GCD algorithm gives better results than the approximate PC-PRS algorithm.

Example 5 (Comparison of PC-PRS with SVD-based algorithms). In Tables 5, 6 and 7, "*back_err*" denotes the backward error in Zeng-Dayton's algorithm [ZD04] defined as follows:

$$back\_err = \sqrt{\frac{||\tilde{c}\tilde{f}_1 - f||_2^2 + ||\tilde{c}\tilde{g}_1 - g||_2^2}{||f||_2^2 + ||g||_2^2}}\ , \tag{4.1}$$

where $\tilde{f}_1, \tilde{g}_1$ and $\tilde{c}$ are co-factors of f, g and an appGCD(f, g), see below.

We compare the PC-PRS and SVD-based algorithms on the following f and g.

Sample 1.

$$\begin{cases} f(x,u) = f_1(x,u)c(x,u), \\ g(x,u) = g_1(x,u)c(x,u), \end{cases} \text{ with } \begin{cases} c(x,u) = (x + u_1 + \cdots + u_r + 1)^2, \\ f_1(x,u) = (x^2 - u_1 - \cdots - u_r - 0.5)^2, \\ g_1(x,u) = (x^2 + u_1 + \cdots + u_r + 0.5)^2. \end{cases}$$

We set r as $r = 1, 2, \ldots, 5$. We show the comparison in Table 5, where "Ave. CPU", "ErrMax" and "*back_err*" are the average CPU time (sec), the maximum relative error and the backward error, respectively.

Table 5. Comparison of PC-PRS with SVD-based GCDs on Sample 1

	Approx. PC-PRS (Maple)		Gao *et al.*'s GCD (Maple)		Zeng-Dayton's GCD (MATLAB)	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r = 1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r = 2$	0.025	5.12e-13	10.286	1.16e-14	0.119	8.41e-14
$r = 3$	0.052	5.12e-13	96.206	1.36e-10	0.756	1.84e-13
$r = 4$	0.151	5.12e-13	about 1300	1.72e-10	1.519	1.16e-13
$r = 5$	0.220	5.12e-13	over 3000	- - - -	446.755	3.94e-13

Sample 2.

$$\begin{cases} f(x,u) = f_2(x,u)c(x,u), \\ g(x,u) = g_2(x,u)c(x,u), \end{cases} \text{ with } \begin{cases} c(x,u) = (x + u_1 + \cdots + u_r^r + 1)^2, \\ f_2(x,u) = (x^2 - u_1 - \cdots - u_r - 0.5)^2, \\ g_2(x,u) = (x^2 + u_1 + \cdots + u_r + 0.5)^2. \end{cases}$$

We set r as $r = 1, 2, \ldots, 5$.

TABLE 6. Comparison of PC-PRS with SVD-based GCDs on Sample 2

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r=1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r=2$	0.062	4.60e-12	11.114	1.30e-14	0.137	6.06e-14
$r=3$	0.130	8.19e-12	247.329	1.91e-10	1.988	7.94e-13
$r=4$	0.399	8.19e-12	over 3000	- - - -	338.754	7.88e-13
$r=5$	3.736	8.19e-12	over 3000	- - - -	over 3000	- - - -

Sample 3.

$$\begin{cases} f(x,u) = f_3(x,u)c(x,u), \\ g(x,u) = g_3(x,u)c(x,u), \end{cases} \text{ with } \begin{cases} c(x,u) = (x+u_1+\cdots+u_r+1)^2, \\ f_3(x,u) = (x^2-u_1-\cdots-u_r^r-0.5)^2, \\ g_3(x,u) = (x^2+u_1+\cdots+u_r+0.5)^2. \end{cases}$$

We set r as $r = 1, 2, \ldots, 5$.

TABLE 7. Comparison of PC-PRS with SVD-based GCDs on Sample 3

	Approx. PC-PRS		Gao *et al.*'s GCD		Zeng-Dayton's GCD	
	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
$r=1$	0.015	5.12e-13	2.647	5.62e-15	0.072	5.80e-15
$r=2$	0.082	5.12e-13	7.899	1.47e-14	0.163	4.87e-15
$r=3$	0.151	5.12e-13	426.549	1.32e-10	83.323	2.62e-13
$r=4$	0.438	5.12e-13	3000 over	- - - -	over 3000	- - - -
$r=5$	2.167	5.12e-13	3000 over	- - - -	over 3000	- - - -

We see that the approximate PC-PRS algorithm shows a very nice efficiency. The CPU time for Gao *et al.*'s algorithm depends on the total-degrees of f and g. Furthermore, Gao *et al.*'s algorithm is extremely slowed down if the number of variables is large. On the other hand, Zeng-Dayton's algorithm is not so slowed down even if the number of variables is increased (Example 1 in [ZD04]). This difference between Gao *et al.*'s algorithm and Zeng-Dayton's algorithm is due to computing environments: efficiency of Zeng-Dayton's algorithm is strongly due to very fast sparse matrix SVD procedures in MATLAB. If we implement Gao *et al.*'s algorithm in MATLAB, the algorithm will show a much better performance. Observing the maximum relative error and the backward error, the approximate PC-PRS algorithm is stable if the total-degree and the number of sub-variables are increased. On the other hand, SVD-based algorithms lose stability gradually.

Comparison of S_k with $S_{\mathbf{k}}$.

The size of the generalized Sylvester matrix increases very rapidly as the degree or the number of variables increases. The generalized Sylvester matrices S_k and

$S_{\mathbf{k}}$ are different. Table 8 shows the sizes of generalized Sylvester matrices S_k and $S_{\mathbf{k}}$ for Samples 1 and 3 above.

TABLE 8. Sizes of generalized Sylvester matrices

	Sample 1 (Table 5)		Sample 3 (Table 7)	
	S_k	$S_{\mathbf{k}}$	S_k	$S_{\mathbf{k}}$
$r=1$	66×30	77×30	66×30	77×30
$r=2$	286×70	539×90	286×70	819×120
$r=3$	1001×140	3773×270	1829×280	9009×660
$r=4$	3003×252	26411×810	20349×3129	99099×5130
$r=5$	8008×420	184877 ×2430	74613 ×8218	1756755 ×53190

We see that $\mathrm{size}(S_k) \le \mathrm{size}(S_{\mathbf{k}})$. By $\mathrm{row}(M)$ and $\mathrm{col}(M)$, we denote the numbers of rows and columns, respectively, of a matrix M. For Gao *et al.*'s Sylvester matrix S_k, $\mathrm{row}(S_k) = \beta(m+n-k,\ell) = \binom{m+n-k+\ell}{\ell}$ and $\mathrm{col}(S_k) = \beta(m-k,\ell) + \beta(n-k,\ell) = \binom{m-k+\ell}{\ell} + \binom{n-k+\ell}{\ell}$. For Zeng-Dayton's Sylvester matrix $S_{\mathbf{k}}$, $\mathrm{row}(S_{\mathbf{k}}) = \prod_{i=1}^{\ell}(m_i+n_i-k_i+1)$ and $\mathrm{col}(S_{\mathbf{k}}) = \prod_{i=1}^{\ell}(m_i-k_i+1)+\prod_{i=1}^{\ell}(n_i-k_i+1)$. We have

$$\begin{aligned}
\binom{m-k+\ell}{\ell} &= \frac{1}{\ell!}\prod_{i=1}^{\ell}(m-k+\ell-i+1) \\
&\le \frac{1}{\ell!}\Big\{(m-k+1)^{\ell} + \ell^2(m-k+1)^{\ell-1} \\
&\qquad +\ell^3(m-k+1)^{\ell-2} + \cdots + \ell^{\ell}\Big\} \\
&\le \frac{\ell+1}{\ell!} \times \Big(\max\{m-k+1,\ell\}\Big)^{\ell}, \qquad (4.2)
\end{aligned}$$

$$\prod_{i=1}^{\ell}(m_i-k_i+1) \ge \prod_{i=1}^{\ell}(\breve{m}-\breve{k}+1) = (\breve{m}-\breve{k}+1)^{\ell}, \qquad (4.3)$$

where $(\breve{m},\breve{k})$ is a pair which minimizes m_i-k_i $(i=1,\ldots,\ell)$. Since $m-k \ge \breve{m}-\breve{k}$, we can bound $\mathrm{row}(S_k)$ and $\mathrm{col}(S_k)$ as follows:

$$\begin{aligned}
\mathrm{row}(S_k) \le \frac{\ell+1}{\ell!}\Big(\max\{m+n-k+1,\ell\}\Big)^{\ell} \le (\breve{m}+\breve{n}-\breve{k}+1)^{\ell} \le \mathrm{row}(S_{\mathbf{k}}), \\
\mathrm{col}(S_k) \le \frac{\ell+1}{\ell!}\Big(\max\{m-k+1,\ell\}+\max\{n-k+1,\ell\}\Big)^{\ell} \\
\le (\breve{m}-\breve{k}+1)+(\breve{n}-\breve{k}+1) \le \mathrm{col}(S_{\mathbf{k}}).
\end{aligned}$$

Therefore, $\mathrm{size}(S_{\mathbf{k}})$ is larger than $\mathrm{size}(S_k)$. In most cases, $\max\{m+n-k+1,\ell\} \approx \breve{m}+\breve{n}-\breve{k}+1$; hence we have

$$\mathrm{row}(S_k) \lesssim \Big(\frac{\ell+1}{\ell!}\Big)\mathrm{row}(S_{\mathbf{k}}). \qquad (4.4)$$

Example 6. We compare the performance of approximate PC-PRS and Gao *et al.*'s GCD by multivariate polynomials generated randomly in Maple. We generate ten polynomial pairs (A_i, B_i) $(i = 1, \ldots, 10)$ as follows:

$$\begin{cases} A_i = a_i c_i + 10^{-2} d_i \\ B_i = b_i c_i + 10^{-2} e_i \end{cases} \quad (i = 1, \ldots, 5),$$

$$\begin{cases} A_{i+5} = a_i c_i + 10^{-5} d_i \\ B_{i+5} = b_i c_i + 10^{-5} e_i \end{cases} \quad (i = 1, \ldots, 5),$$

where $a_i, b_i, c_i, d_i, e_i \in \mathbb{C}[x, y]$ $(i = 1, 2, 3)$ and $a_i, b_i, c_i, d_i, e_i \in \mathbb{C}[x, y, z]$ $(i = 4, 5)$, with $||a_i|| = ||b_i|| = ||c_i|| = ||d_i|| = ||e_i|| = 1$ $(i = 1, \ldots, 5)$, and we generate them randomly in Maple. Table 9 shows the comparison.

TABLE 9. Comparison of PC-PRS with SVD-Based GCDs (sec)

	Approx. PC-PRS (Maple)		Gao *et al.*'s GCD (Maple)		Zeng-Dayton's GCD (MATLAB)	
Ex.	Ave. CPU	ErrMax	Ave. CPU	*back_err*	Ave. CPU	*back_err*
1	0.025	7.27e-12	0.871	9.65e-3	0.094	3.52e-1
2	0.044	3.00e-14	1.825	1.66e-2	0.094	4.83e-1
3	0.029	1.37e-13	3.670	1.25e-2	0.090	5.13e-1
4	0.110	3.47e-7	4.892	1.66e-2	0.253	3.32e-0
5	— Error —		4.119	1.18e-2	0.294	2.44e-0
6	0.037	4.68e-11	2.170	9.65e-6	0.098	2.08e-5
7	0.040	1.39e-13	1.121	1.66e-5	0.087	1.80e-5
8	0.039	1.66e-9	0.341	1.25e-5	0.082	1.95e-5
9	0.049	3.22e-6	2.503	1.66e-5	0.102	2.24e-5
10	0.471	1.90e-8	2.563	1.18e-5	0.100	1.76e-5

We see from Table 9 that Gao *et al.*'s algorithm is unstable if the perturbation is not small; it is difficult to determine the total-degree of appGCD by only the SVDs of univariate polynomials. For example, in Ex. 3, we obtain an appGCD of the total-degree 1, but $\mathrm{tdeg}(\gcd(A_3, B_3)) = 3$. In Gao *et al.*'s algorithm, we determine the total-degree by rank deficiency by observing the largest gap of singular values, and hence the determination is unstable if perturbation is not small. The situation is the same in Zeng-Dayton's algorithm. Furthermore, since Zeng-Dayton's algorithm requires degrees of ℓ univariate polynomial GCDs for ℓ variables, it is more unstable if the number of variables is large. In fact, Zeng-Dayton's algorithm does not give correct appGCD for Ex. 1–5. In Ex. 4 and 8, $||\mathrm{lc}(B_4)||$ and $||\mathrm{lc}(B_9)||$ are small compared with the norms of other coefficients. Therefore, the approximate PC-PRS algorithm causes large cancellation errors. On the other hand, Gao *et al.*'s algorithm and Zeng-Dayton's algorithm are not affected by norms of coefficients. In Ex. 5, "Error" means that the computation

stopped by failure (in this case, an error happened in computing an appGCD of $\mathrm{lc}(A_5)$ and $\mathrm{lc}(B_5)$; $\gcd(\mathrm{lc}(A_5), \mathrm{lc}(B_5); 0.01) = \gcd(0, \mathrm{lc}(B_5); 0.01) = 0$. The reason is the small leading coefficient: $||\mathrm{lc}(A_5)|| < 0.01$). The approximate PC-PRS algorithm is not complete yet. Improvement of the algorithm is our future work.

5. Conclusion

In this paper, we proposed an Approximate PC-PRS algorithm and briefly discuss five algorithms for computing appGCD of multivariate polynomials. Our algorithm cut-offs unnecessary higher degree terms in the computation of PRS, making the computation quite efficient. We showed a good performance of the approximate PC-PRS algorithm. This algorithm is very nice except in the case of a small leading coefficient (Example 4 and Ex. 5 in Example 6). SVD-based algorithms can compute appGCD fast, only when the degree and number of variable are small. The size of the generalized Sylvester matrix increases very rapidly as the degree and the number of variables are increased; hence SVD-based algorithms require very fast SVD routines such as those in MATLAB. SVD-based algorithms are unstable when the perturbation is not small; we cannot determine the degree of appGCD. On the other hand, SVD-based algorithms are not affected by norms of coefficients.

Acknowledgments. The author expresses his sincere thank to Prof. Tateaki Sasaki, his master thesis supervisor, who gave him a theme of modifying the PC-PRS GCD algorithm to an approximate one and helped him variously. He also thanks Prof. Zhonggang Zeng for presenting him the multivariate GCD package MVGCD in MATLAB code [ZD04].

References

[CGTW95] R. Corless, P. Gianni, B. Trager and S. Watt, *The Singular Value Decomposition for Polynomial Systems*, Proc. of ISSAC '95, ACM, 1995, 195–207.

[EGL97] I.Z. Emiris, A. Galligo and H. Lombarbi, *Certified Approximate Univariate GCDs*, J. Pure and Appl. Alge., **117**&**118** (1997), 229–251.

[GKMYZ04] S. Gao, E. Kaltofen, J.P. May, Z. Yang and L. Zhi, *Approximate Factorization of Multivariate Polynomials via Differential Equations*, Proc. of ISSAC '04, ACM, 2004, 167–174.

[GL89] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, Maryland, 1989.

[Kal04] E. Kaltofen, `http://www4.ncsu.edu/~kaltofen/software/appfac/issac04_mws/`, 2004.

[KS97] F. Kako and T. Sasaki, *Proposal of "Effective Floating-point Number" for Approximate Algebraic Computation*, Preprint of Tsukuba Univ., 1997.

[LZN00] K. Li, L. Zhi and M-T. Noda, *Solving Approximate GCD of Multivariate Polynomial by Maple/Matlab/C Combination*, Proc. of ATCM 2000, World Scientific, 2000, 492–499.

[Matlab] MATLAB Application Program Interface Guide, The Math Works, Inc.

[MY73] J. Moses and D.Y.Y. Yun, *The EZGCD Algorithm*, Proc. of ACM National Conference, ACM, 1973, 159–166.

[ONS91] M. Ochi, M-T. Noda and T. Sasaki, *Approximate Greatest Common Divisor of Multivariate Polynomials and Its Application to Ill-Conditioned Systems of Algebraic Equations*, J. Inform. Proces., **14** (1991), 292–300.

[SN89] T. Sasaki and M-T. Noda, *Approximate Square-free Decomposition and Root-finding of Ill-conditioned Algebraic Equations*, J. Inform. Proces., **12** (1989), 159–168.

[SS92] T. Sasaki and M. Suzuki, *Three New Algrithms for Multivariate Polynomial GCD*, J. Symb. Comput., **13** (1992), 395–411.

[SY98] T. Sasaki and S. Yamaguchi, *An Analysis of Cancellation Error in Multivariate Hensel Construction with Floating-point Number Arithmetic*, Proc. of ISSAC '98, ACM, 1998, 1–8.

[Wan80] P.S. Wang, *The EEZ-GCD Algorithm*, SIGSAM Bulletin **14** (1980), 50–60.

[YZ05] T.Y. Li and Z. Zeng, *A Rank-Revealing Method with Updating, Downdating, and Applications*, SIAM J. Matrix Anal. Appl., **26** (2005), no. 4, 918–946.

[ZD04] Z. Zeng and B.H. Dayton, *The Approximate GCD of Inexact Polynomials Part I I: A Multivariate Algorithm*, Proc. of ISSAC '04, ACM, 2004, 320–327.

[Zen04] Z. Zeng, MVGCD – `http://www.neiu.edu/~zzeng/polynmat.htm`, 2004.

[Zhi04] L. Zhi, `http://www.mmrc.iss.ac.cn/~lzhi/Research/issac04_mws.html`, 2004.

[ZN00] L. Zhi and M-T. Noda, *Approximate GCD of Multivariate Polynomials*, Proc. of ASCM 2000, World Scientific, 2000, 9–18.

[ZY04] L. Zhi and Z. Yang, *Computing Approximate GCD of Multivariate Polynomials by Structure Total Least Norm*, MM Research Preprint, MMRC, AMSS, Academia, Sinica, 2004, No. 23, 388–401.

Masaru Sanuki
Graduate School of Pure and Applied Sciences
University of Tsukuba
Tsukuba-Shi, Ibaraki 305-8571, Japan
e-mail: `sanuki@math.tsukuba.ac.jp`