The approximate GCD of inexact polynomials Part II: a multivariate algorithm [*]

Zhonggang Zeng
Department of Mathematics
Northeastern Illinois University
Chicago, IL 60625
zzeng@neiu.edu

Barry H. Dayton
Department of Mathematics
Northeastern Illinois University
Chicago, IL 60625
bhdayton@neiu.edu

ABSTRACT

This paper presents an algorithm and its implementation for computing the approximate GCD (greatest common divisor) of multivariate polynomials whose coefficients may be inexact. The method and the companion software appears to be the first practical package with such capabilities. The most significant features of the algorithm are its robustness and accuracy as demonstrated in the results of computational experiment. In addition, two variations of a squarefree factorization algorithm for multivariate polynomials are proposed as an application of the GCD algorithm.

Categories and Subject Descriptors

G.1.5 [**Mathematics of Computing**]: Roots of Nonlinear Equations – Iterative methods; methods for polynomials; G.4 [**Mathematics of Computing**]: Mathematical Software – Algorithm design and analysis; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

polynomial, greatest common divisor, squarefree, factorization

1. INTRODUCTION

In this paper we present an algorithm and its companion software package that compute the approximate GCD (greatest common divisor) of multivariate polynomials whose coefficients may be perturbed. As an application, we also introduces two variations of a squarefree factorization method for inexact polynomials.

As surveyed in [13], GCD-finding is one of the basic operations in algebraic computation with a wide range of applications. The multivariate GCD in particular, can be applied to engineering problems such as image restoration [9, 11] where the given polynomials contain noises. Same as in the univariate case, the existing symbolic GCD-finders may not be suitable for inexact polynomials since GCD computation is infinitely sensitive to perturbations. Therefore, robust algorithms capable of computing the *approximate* GCD of inexact polynomials is needed for practical problems.

GCD-finding also appears frequently as a component of other algebraic computations. For example, the irreducible factorization of an inexact multivariate polynomial is an open challenge of symbolic computation [5]. In several approaches for solving this problem, the polynomial is required to be squarefree, while the approximate GCD computation may be necessary to extract squarefree components [3, 4] if the given polynomial has repeated factors.

Compared with univariate case, the difficulty level of multivariate GCD computation appears much higher. While previous methods may not be convincing enough in robustness for univariate GCD, the endeavors in the multivariate case are at even earlier stages of development. Nevertheless, the approximate GCD computation for multivariate polynomials has been studied in many reports since 1970's [1]. A possible extension of a univariate GCD is briefly outlined in [2] by Corless, Gianni, Trager and Watt. Other methods include polynomial remainder sequence [10] by Sasaki and Sasaki, generalized subresultant method [8] by Ochi, Noda and Sasaki, as well as Hensel lifting strategy [16, 17] by Zhi, Li and Noda. A Sylvester-like resultant method is proposed by Phillai and Liang in the context of image restoration [9]. All those approaches are promising. However, we have not seen completed implementations either as stand-alone software or as build-in functions in computer algebra packages.

We introduce a blackbox-type algorithm along with the release of an implementation MVGCD, which appears to be the first practical software for computing the approximate GCD of multivariate polynomials. The contribution of this paper includes an original design of a three stage algorithm described in §5. At Stage I, the degree of the GCD in each variable is determined using the robust univariate GCD algorithm that is presented in Part I [13] of this series. Once the degree structure is identified, the GCD factors are obtained at Stage II in an iterative null vector computation and a least squares division. To achieve the optimal robust-

[*]This is the second part of the series for computing the approximate GCD. Copies of Part I [13] along with software packages UVGCD and MVGCD are available from the authors upon request.

ISSAC'04, July 4–7, 2004, Santander, Spain

ness and accuracy, the GCD factors are refined and certified at Stage III as a least squares solution to an overdetermined quadratic system. In theoretical aspects, we provide a rigorous formulation of the approximate GCD that removes the ill-posedness of exact GCD finding. Moreover, we introduce an approximate GCD condition number and error bound, along with other analytical issues in §4. Extensive computing results are shown in §7. In §6, we present two versions of a squarefree factorization method.

2. A BRIEF INTRODUCTION TO THE UNIVARIATE ALGORITHM

The univariate GCD algorithm and its implementation presented in Part I [13] is an indispensable component for the multivariate GCD computation. The algorithm is mainly based on numerical matrix computation. For a given pair of univariate polynomials p and q, we seek a GCD triplet (u, v, w) such that, with boldface letters representing corresponding coefficient vectors,

$$u\,v = f, \quad u\,w = g, \quad \mathbf{r}^H \mathbf{u} = 1. \tag{1}$$

The first stage of the computation is to find the degrees of u. For that purpose, a sequence of Sylvester subresultant matrices $S_0(p,q)$, $S_1(p,q)$, $\cdots$ is constructed one-by-one, using the coefficients of p and q, until the first rank-deficient matrix $S_{n-k}(p,q)$ emerges. Here n is the higher degree of p and q, while matrix rank is in approximate sense [6]. Then the degree of u is k, while the coefficient vectors $\mathbf{v}$ and $\mathbf{w}$ are extracted from the null vector of $S_{n-k}(p,q)$.

To avoid the unstable synthetic division, $\mathbf{u}$ is obtained via solving a linear least squares problem. The obtained coefficient vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ are an initial approximation to the GCD triplet (u, v, w).

With the degrees of u, v, and w, the system (1) can be formulated as an overdetermined quadratic system

$$F(\mathbf{u}, \mathbf{v}, \mathbf{w}) = 0 \tag{2}$$

Solving this system is proved to be a regular problem in contrast to an ill-posed one in exact GCD-finding. Since a good approximation has been obtained as the initial iterate, the Gauss-Newton iteration applied on the system (2) converges. The outcome of the Gauss-Newton iteration serves two objectives. For one, it provides a refinement for the GCD triplet and reaches the highest possible accuracy permissible by the GCD condition number. At the same time the residual of (2) provides a measurement for the backward error, which is the distance of (p,q) to a polynomial pair with exact GCD in u.

The univariate GCD algorithm is implemented as a package uvGCD in both Maple and Matlab. The numerical testing demonstrates its tremendous robustness and high accuracy compared with existing software. In addition to its essential role in the multivariate algorithm here, it has also been integrated in a new root-finding package MultRoot [15] that is capable of calculating multiple roots with high accurately without extending machine precision even if the polynomial is inexact.

3. NOTATION

Unless otherwise specified, polynomials are ℓ-variate in variables x_1, $\cdots$, x_ℓ with complex coefficients. The complex vector space of dimension k is denoted by $\mathbb{C}^k$. For every vector or matrix $(\cdot)$, the notation $(\cdot)^\top$ represents the transpose of $(\cdot)$, and $(\cdot)^H$ the Hermitian adjoint (i.e. conjugate transpose). For any full (column) rank matrix A of $m \times n$ with $m \geq n$, $A^+ = (A^H A)^{-1} A^H$ is the pseudo-inverse of A. We say a polynomial p has an *ℓ-degree* $\mathbf{n} = [n_1, \cdots, n_\ell]$, denoted by $deg(p) = \mathbf{n}$, if the univariate degree of p in x_j is n_j for $j = 1, \cdots, \ell$. For example, the polynomial

$$p(x_1, x_2, x_3) = x_1^3 x_2 + 3x_1 x_3^5 - 5x_2^2 x_3^4 - 8 \tag{3}$$

has a 3-degree $\mathbf{n} = [3, 2, 5]$. We say an ℓ-degree $\mathbf{m} = [m_1, \cdots, m_\ell]$ is less than or equal to $\mathbf{n} = [n_1, \cdots, n_\ell]$, or $\mathbf{m} \leq \mathbf{n}$, if $m_j \leq n_j$, $j = 1, \cdots, \ell$. Every ℓ-degree is denoted by a boldface letters, say $\mathbf{n}$, with the same letter in plain upper case, say N, denoting the dimension of the vector space $\mathbb{P}_\mathbf{n}$ consisting of all polynomials with ℓ-degree less than or equal to $\mathbf{n}$. It is easy to see that

$$N = \nu(\mathbf{n}) \overset{\text{def}}{=} (n_1 + 1)(n_2 + 1) \cdots (n_\ell + 1). \tag{4}$$

Degree addition $\mathbf{m} + \mathbf{n}$ is defined as vector addition.

3.1 The coefficient vector

For an ℓ-degree $\mathbf{n} = [n_1, \cdots, n_\ell]$, let $\mathbb{P}_\mathbf{n}$ be the complex vector space of all polynomials with ℓ-degree no larger than $\mathbf{n}$. Naturally, there is a monomial basis

$$\left\{ x_1^{i_1} x_2^{i_2} \cdots x_\ell^{i_\ell} \;\middle|\; [i_1, \cdots, i_\ell] \leq \mathbf{n} \right\} \tag{5}$$

for $\mathbb{P}_\mathbf{n}$. In practice a convenient ordering of monomials needs to be specified. Within $\mathbb{P}_\mathbf{n}$ we use the pure lexicographical order (plex) with $x_1 < x_2 < \cdots < x_\ell$. When $[i_1, \cdots, i_\ell] \leq \mathbf{n}$ the monomial $x_1^{i_1} \cdots x_\ell^{i_\ell}$ will be said to be the j-th monomial in $\mathbb{P}_\mathbf{n}$ if it is the j-th smallest monomial in this ordering. The index j can be calculated by

$$j = 1 + \sum_{k=1}^{\ell} \left[i_k \prod_{l=1}^{k-1} (n_l + 1) \right], \quad \text{with } \prod_{l=1}^{0} (n_l + 1) \equiv 1. \tag{6}$$

We therefore have an ordered basis $\{p_1, \cdots, p_N\}$ for $\mathbb{P}_\mathbf{n}$, where p_j is the j-th monomial in $\mathbb{P}_\mathbf{n}$ for $j = 1, \cdots, N$. Every polynomial $p \in \mathbb{P}_\mathbf{n}$, as a linear combination $p = \sum_{j=1}^N \alpha_j p_j$ of p_j's, corresponds to a unique coefficient vector

$$\Psi_\mathbf{n}(p) = \mathbf{p} = (\alpha_1, \cdots, \alpha_N)^\top \in \mathbb{C}^N.$$

For example, the polynomial p in (3) corresponds to the coefficient vector $\mathbf{p} = (\alpha_1, \cdots, \alpha_{72})^\top$ with α_j's being zero except $\alpha_1 = -8$, $\alpha_7 = 1$, $\alpha_{57} = -5$, $\alpha_{62} = 3$. Throughout this paper, a polynomial is denoted by a lower case letter, say p, with the same letter in boldface, say $\mathbf{p}$ denote the coefficient vector, while $\Psi_\mathbf{n} : \mathbb{P}_\mathbf{n} \longrightarrow \mathbb{C}^N$ denotes the mapping that converts a polynomial $p \in \mathbb{P}_\mathbf{n}$ to $\mathbf{p} \in \mathbb{C}^N$. Notice that a polynomial $p \in \mathbb{P}_\mathbf{n}$ can also be considered an element in $\mathbb{P}_\mathbf{k}$ for every $\mathbf{k} > \mathbf{n}$, while $\Psi_\mathbf{n}$ and $\Psi_\mathbf{k}$ maps same p to different coefficient vectors in $\mathbb{C}^N$ and $\mathbb{C}^K$ respectively. The meaning of $\mathbf{p}$ should be clear from context.

3.2 The convolution matrix

For a fixed $f \in \mathbb{P}_\mathbf{n}$ and any $g \in \mathbb{P}_\mathbf{m}$, the polynomial multiplication $f\,g$ is regarded as a linear transformation

$$F_\mathbf{m} : \mathbb{P}_\mathbf{m} \longrightarrow \mathbb{P}_{\mathbf{m}+\mathbf{n}} \quad \text{with} \quad F_\mathbf{m}(g) = f\,g.$$

With the ordered monomial bases for $\mathbb{P}_\mathbf{n}$, $\mathbb{P}_\mathbf{m}$ and $\mathbb{P}_{\mathbf{n}+\mathbf{m}}$ defined in §3.1, there is a matrix $C_\mathbf{m}(f)$ of dimension $\nu(\mathbf{m}+$

$\mathbf{n}) \times \nu(\mathbf{m})$ associated with the linear transformation $F_{\mathbf{m}}$ such that $C_{\mathbf{m}}(f)$ maps every coefficient vector in $\mathbb{C}^M$ into a coefficient vector in $\mathbb{C}^{\nu(\mathbf{m}+\mathbf{n})}$ in the manner of

$$C_{\mathbf{m}}(f)\Psi_{\mathbf{m}}(g) = \Psi_{\mathbf{m}+\mathbf{n}}(f\,g), \quad \text{for every} \quad g \in \mathbb{P}_{\mathbf{m}}.$$

This matrix in univariate case is called a convolution matrix with a Toeplitz structure [13]. For convenience we also call $C_{\mathbf{m}}(f)$ the *convolution matrix* associated with $\mathbf{m}$ and f.

We can construct the convolution matrix $C_{\mathbf{m}}(f)$ in a streamlined process as follows. Let $\{q_1, \cdots, q_M\}$ be the ordered monomial basis for $\mathbb{P}_{\mathbf{m}}$ and $g = \sum_{j=1}^{M} \gamma_j q_j$. Then $\Psi_{\mathbf{m}}(g) = \mathbf{g} = (\gamma_1, \cdots, \gamma_M)^\top$ and

$$\begin{aligned} C_{\mathbf{m}}(f)\mathbf{g} &= \Psi_{\mathbf{n}+\mathbf{m}}\left(f \sum_{j=1}^{M} \gamma_j q_j\right) = \sum_{j=1}^{M} \gamma_j \Psi_{\mathbf{n}+\mathbf{m}}(f\,q_j) \\ &= \Big[\Psi_{\mathbf{n}+\mathbf{m}}(fq_1), \cdots, \Psi_{\mathbf{n}+\mathbf{m}}(fq_M)\Big] \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_M \end{bmatrix}. \end{aligned}$$

Namely,

$$C_{\mathbf{m}}(f) = \Big[\Psi_{\mathbf{n}+\mathbf{m}}(fq_1), \cdots, \Psi_{\mathbf{n}+\mathbf{m}}(fq_M)\Big] \qquad (7)$$

More specifically, let $q_j = x_1^{i_1} \cdots x_\ell^{i_\ell}$ be the j-th monomial in $\mathbb{P}_{\mathbf{m}}$. Then fq_j is obtained by changing every term in $x_1^{k_1} \cdots x_\ell^{k_\ell}$ of f to a term in $x_1^{k_1+i_1} \cdots x_\ell^{k_\ell+i_l}$ with the same coefficient. From this observation, the following procedure is designed to construct the convolution matrix $C_{\mathbf{m}}(f)$.

```
• Input: $\mathbf{n} = [n_1, \cdots, n_\ell]$,  $\mathbf{m} = [m_1, \cdots, m_\ell]$,  $f \in \mathbb{P}_{\mathbf{n}}$.
• Set $A$ as a zero matrix of dimension $\nu(\mathbf{n}+\mathbf{m}) \times \nu(\mathbf{m})$.
• Replace the first column of $A$ with $\Psi_{\mathbf{n}+\mathbf{m}}(f)$.
• For  $j = 2, \cdots, M$  do
    – Find the $j$-th monomial $x_1^{i_1} \cdots x_\ell^{i_\ell}$ in $\mathbb{P}_{\mathbf{m}}$
    – Set
          $\alpha_j = \sum_{\sigma=1}^{\ell} \left[ i_\sigma \prod_{\eta=1}^{\sigma-1} (n_\eta + m_\eta + 1) \right]$      (8)
    – Form a vector $\mathbf{b}_j \in \mathbb{C}^{\nu(\mathbf{n}+\mathbf{m})}$ by moving every
      nonzero entry of $\Psi_{\mathbf{n}+\mathbf{m}}(f)$ down $\alpha_j$ positions.
    – Replace the $j$-th column of $A$ with $\mathbf{b}_j$.
  End do
• Output $A$ as $C_{\mathbf{m}}(f)$.
```

For example, let p be given in (3), the structure of $C_{[2,3,4]}(f)$ is shown in Figure 1. This convolution matrix is of dimension 360×60 where the same four nonzero entries $-8, 1, -5, 3$ ordered from top down appear in every column. Convolution matrices for multivariate polynomials can be quite large. When a given polynomial is sparse like p in (3), the convolution matrix is also sparse. As shown in Figure 1, there are only 240 nonzero entries (about 1% of all entries) in $C_{[2,3,4]}(p)$.

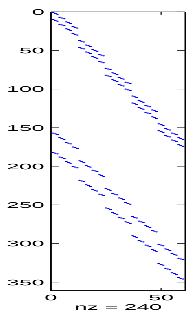

Figure 1: A convolution matrix

The structure of convolution matrices is also convenient for storage. The minimum storage requirement consists of only the nonzero entries in the first column as well as the α_j's in (8). The other columns can be generated on demand.

4. THEORETICAL BASIS FOR THE ALGORITHM

For simplicity, we use EGCD for "exact GCD" and AGCD for "approximate GCD". We shall define AGCD later in this section. Two polynomials f and g are called *co-prime* if there is no non-constant polynomial divides both f and g. For any polynomial pair (p, q), if there is a triplet (u, v, w) of polynomials such that

$$p = u\,v, \;\; q = u\,w, \;\; v \text{ and } w \text{ are co-prime} \qquad (9)$$

then u is called an EGCD of (p, q), denoted by $u = GCD(p, q)$, with co-factors v and w. We also call (u, v, w) an EGCD triplet of p and q. The EGCD triplet is unique up to constant multiples. We use the Euclidean norm on the coefficient difference as the distance between polynomials.

4.1 The AGCD and its regularity

Let the ℓ-degrees of p and q be $\mathbf{m}$ and $\mathbf{n}$ respectively. The objective is to find an ℓ-degree $\mathbf{k}$ and to solve a quadratic system (9) in the form of

$$F(\mathbf{z}) = \mathbf{b} \qquad (10)$$

for $\mathbf{z}$ where

$$F(\mathbf{z}) = \begin{bmatrix} \mathbf{r}^H\mathbf{u} - 1 \\ C_{\mathbf{m}-\mathbf{k}}(v)\,\mathbf{u} \\ C_{\mathbf{n}-\mathbf{k}}(w)\,\mathbf{u} \end{bmatrix}, \;\; \mathbf{z} = \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{bmatrix}, \;\; \mathbf{b} = \begin{bmatrix} 0 \\ \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \quad (11)$$

$$\mathbf{r}, \mathbf{u} \in \mathbb{C}^K, \; \mathbf{v} \in \mathbb{C}^{\nu(\mathbf{m}-\mathbf{k})}, \; \mathbf{w} \in \mathbb{C}^{\nu(\mathbf{n}-\mathbf{k})}, \; \mathbf{p} \in \mathbb{C}^M, \; \mathbf{q} \in \mathbb{C}^N.$$

The vector $\mathbf{r}$ in (11) is a scaling vector that will be specified in §5.3. The system (10) is generally overdetermined. When p and q are inexact, there are no conventional solutions to (10). Instead, we seek a *least squares solution* to (10) that minimizes the nearness $\|\, F(\mathbf{z}) - \mathbf{b} \,\|_2$. Same as in [13, Lemma 1], the minimum occurs at a point $\mathbf{z}$ where

$$J(\mathbf{z})^H [\, F(\mathbf{z}) - \mathbf{b} \,] = 0.$$

Here,

$$J(\mathbf{z}) = \begin{bmatrix} \mathbf{r}^H & & \\ C_{\mathbf{k}}(v) & C_{\mathbf{m}-\mathbf{k}}(u) & \\ C_{\mathbf{k}}(w) & & C_{\mathbf{n}-\mathbf{k}}(u) \end{bmatrix} \qquad (12)$$

is the Jacobian of $F(\mathbf{z})$. Similar to the univariate case [13, Definition 1], we define the AGCD as follows.

Definition 1. *Let p and q be ℓ-variate polynomials of ℓ-degrees $\mathbf{m}$ and $\mathbf{n}$ respectively. A polynomial u is called an AGCD of p and q within tolerance $\varepsilon > 0$ if u is of the highest ℓ-degree $\mathbf{k}$ along with co-factors $v \in \mathbb{P}_{\mathbf{m}-\mathbf{k}}$ and $w \in \mathbb{P}_{\mathbf{n}-\mathbf{k}}$ that form $\mathbf{z}$ in (11) satisfying $\|\, F(\mathbf{z}) - \mathbf{b} \,\|_2 \le \varepsilon$ and $J(\mathbf{z})^H [\, F(\mathbf{z}) - \mathbf{b} \,] = 0$, where $J(\mathbf{z})$ given in (12) is the Jacobian of $F(\mathbf{z})$. For simplicity, we also call (u, v, w) an AGCD triplet for p and q.*

The following proposition is fundamental in establishing the regularity of AGCD finding.

Proposition 1. *Let polynomials u, v and w be of ℓ-degrees $\mathbf{k}$, $\mathbf{m} - \mathbf{k}$ and $\mathbf{n} - \mathbf{k}$ respectively. Assuming the scaling vector $\mathbf{r}$ satisfies $\mathbf{r}^H\mathbf{u} \neq 0$ while v and w are co-prime, then the Jacobian $J(\mathbf{z})$ in (12) is of full (column) rank.*

4.2 The sensitivity and error bound

Let (u,v,w) be an AGCD triplet for p and q. Since the Jacobian of the system (10) is of full rank, its smallest singular value is strictly positive. Using the same argument in [13, §4], we thereby obtain a sensitivity measurement and an asymptotic error bound.

Definition 2. *Let (u,v,w) be an AGCD triplet of certain polynomial pair (p,q) with $J(\mathbf{z})$, $\mathbf{z}$ and $\mathbf{r}$ being given in (11) and (12). Let σ_{min} be the smallest singular value of $J(\mathbf{z})$. Then $\kappa = \frac{1}{\sigma_{min}}$ is called the AGCD condition number of (p,q) at the triplet (u,v,w) associated with $\mathbf{r}$.*

Proposition 2. *Let (p,q) and $(\hat{p},\hat{q})$ be two pairs of ℓ-variate polynomials with AGCD triplets (u,v,w) and $(\hat{u},\hat{v},\hat{w})$ respectively, associated with scaling vector $\mathbf{r}$ within a sufficiently small tolerance ε. Assume $(\hat{p},\hat{q})$ are sufficiently near (p,q). Then*

$$\left\| \begin{bmatrix} \hat{\mathbf{u}}-\mathbf{u} \\ \hat{\mathbf{v}}-\mathbf{v} \\ \hat{\mathbf{w}}-\mathbf{w} \end{bmatrix} \right\|_2 \leq \kappa \left(2\varepsilon + \left\| \begin{bmatrix} \hat{\mathbf{p}}-\mathbf{p} \\ \hat{\mathbf{q}}-\mathbf{q} \end{bmatrix} \right\|_2 \right) + h.o.t. \qquad (13)$$

where h.o.t represents higher order terms of the ε.

By Proposition 1 and Proposition 2, computing the approximate GCD is a regular problem and no longer ill-posed. Under our formulation of AGCD-finding, a tiny perturbation does not necessarily alter the structure of the AGCD outcome. The sensitivity of the AGCD under perturbation can be conveniently measured while the error on the computed AGCD triplet is bounded.

4.3 The AGCD degree

The ℓ-degree of the AGCD can be determined using the univariate algorithm in [13]. Consider a given pair (p,q) of polynomials. For a fixed $\mathbf{s} = (s_1,\cdots,s_\ell)^\top \in \mathbb{C}^\ell$, let

$$\begin{array}{rcl} p_j(t) & = & p(s_1,\cdots,s_{j-1},t,s_{j+1},\cdots,s_\ell) \\ q_j(t) & = & q(s_1,\cdots,s_{j-1},t,s_{j+1},\cdots,s_\ell) \\ u_j & = & GCD(p_j,q_j), \;\; v_j = \frac{p_j}{u_j}, \;\; w_j = \frac{q_j}{u_j} \\ & & j = 1,\cdots,\ell. \end{array} \qquad (14)$$

We obtain ℓ pairs of univariate polynomials. If (p,q) has an EGCD in u with ℓ-degree $\mathbf{k} = [k_1,\cdots,k_\ell]$, then each pair (p_j,q_j) has a univariate EGCD u_j of a degree may or may not equal to k_j. When $\mathbf{s}$ is chosen randomly, however, there is a probability one to get u_j with exact degree k_j.

Proposition 3. *Let (p,q) be a pair of polynomials with an EGCD in u of ℓ-degree $\mathbf{k} = [k_1,\cdots,k_\ell]$. Then for almost all $\mathbf{s} \in \mathbb{C}^\ell$, the univariate polynomial pair (p_j,q_j) defined in (14) has an EGCD $u_j = GCD(p_j,q_j)$ of degree k_j for $j = 1,\cdots,\ell$.*

For an AGCD triplet (u,v,w) of (p,q), (p,q) is near $(\hat{p},\hat{q})$ that has EGCD triplet (u,v,w). Therefore (p_j,q_j) is near (u_jv_j, u_jw_j). By calculating the univariate AGCD of each pair (p_j,q_j) we can find the AGCD ℓ-degree $\mathbf{k}$.

4.4 The AGCD factors

When the ℓ-degree of the AGCD is known for a given pair (p,q), the following proposition provides a mechanism for calculating the co-factors v and w in the AGCD triplet (u,v,w).

Proposition 4. *Let p and q be polynomials of ℓ-degrees $\mathbf{m}$ and $\mathbf{n}$ respectively. Then $u = GCD(p,q)$ is of ℓ-degree $\mathbf{k}$ if and only if the matrix*

$$S_{\mathbf{k}}(p,q) = \left[\, C_{\mathbf{m}-\mathbf{k}}(q) \; \middle| \; C_{\mathbf{n}-\mathbf{k}}(p) \, \right]$$

is rank-deficient by one. In that case the null space of $S_{\mathbf{k}}(p,q)$ is spanned by $\begin{bmatrix} -\mathbf{v} \\ \mathbf{w} \end{bmatrix}$, where $\mathbf{v} \in \mathbb{C}^{\nu(\mathbf{m}-\mathbf{k})}$ and $\mathbf{w} \in \mathbb{C}^{\nu(\mathbf{n}-\mathbf{k})}$ are coefficient vectors of the co-factors v and w respectively.

In AGCD computation, $S_{\mathbf{k}}(p,q)$ can not be expected to be exactly rank-deficient. Nonetheless $S_{\mathbf{k}}(p,q)$ has its smallest singular value being tiny, while the corresponding right singular vector is formed by scaled $\mathbf{v}$ and $\mathbf{w}$. After extracting the co-factors v and w from that right singular vector, we can solve the linear system

$$C_{\mathbf{k}}(v)\,\mathbf{u} = \mathbf{p} \qquad (15)$$

for its (approximate) least squares solution $\mathbf{u}$.

4.5 Iterative refinement

If the ℓ-degree of the AGCD is known for a given polynomial pair (p,q), the quadratic system (10) can be set up. Since the Jacobian $J(\mathbf{z})$ is of full rank at the AGCD triplet (u,v,w) forming $\mathbf{z}$, the Gauss-Newton iteration

$$\mathbf{z}_{j+1} = \mathbf{z}_j - J(\mathbf{z}_j)^+ \left[F(\mathbf{z}_j) - \mathbf{b} \right], \quad j = 0,1,\cdots \qquad (16)$$

is well defined and locally convergent to the AGCD triplet. Here in (16), $\mathbf{z}$, $F(\mathbf{z})$ and $\mathbf{b}$ are defined in (11) while $J(\mathbf{z})$ is the Jacobian of $F(\mathbf{z})$ given in (12).

Proposition 5. *Let p and q be polynomials of ℓ-degrees $\mathbf{m}$ and $\mathbf{n}$ respectively with an AGCD triplet (u,v,w) within tolerance ε. Let F, $\mathbf{z}$ and $\mathbf{b}$ be defined in (11). Then there are constants $\delta > 0$ and $\tau > 0$ depending on (u,v,w) and ε such that if $\|F(\mathbf{z}) - \mathbf{b}\|_2 < \delta$ and $\|\mathbf{z}_0 - \mathbf{z}\|_2 < \tau$, then the Gauss-Newton iteration (16) converges to $\mathbf{z}$ with a linear rate. In addition to above assumptions, if $\|F(\mathbf{z}) - \mathbf{b}\|_2 = 0$, then the convergence is quadratic.*

The AGCD factors calculated by the method in §4.4 serve as initial iterate for the iterative refinement (16). As a by-product, the nearness $\|F(\mathbf{z}) - \mathbf{b}\|_2$ is also obtained at the end of the iteration (16). This nearness certifies the final AGCD triplet (u,v,w).

5. THE ALGORITHM

The algorithm consists of three stages:

Stage I: Calculating the AGCD ℓ-degree.
Stage II: Calculating the AGCD factors.
Stage III: Iterative refinement if necessary.

We describe the details for each stage in this section.

5.1 Stage I

Let (p,q) be a given pair polynomials along with a tolerance $\varepsilon > 0$. As shown in §4.3, the components of the AGCD ℓ-degree are the degrees of ℓ univariate AGCD's. An accurate and robust univariate AGCD-finder is crucial for the multivariate case. The univariate AGCD algorithm in [13] and the companion software uvGCD is suitable for our purpose. The procedure of Stage I can be summarized below.

```
• Input: polynomials $p$ and $q$, residual tolerance $\varepsilon$.
• Generate a random vector $\mathbf{s} = (s_1, \cdots, s_\ell)^\top$.
• For  $j = 1, \cdots, \ell$  do
    – Form a univariate polynomial pair
          $p_j(t) = p(s_1, \cdots, s_{j-1}, t, s_{j+1}, \cdots, s_\ell)$
          $q_j(t) = q(s_1, \cdots, s_{j-1}, t, s_{j+1}, \cdots, s_\ell)$
    – Apply uvGCD to attain the AGCD of $(p_j, q_j)$ as
      $u_j$ within tolerance $\varepsilon$.
    – Set $k_j = deg(u_j)$.
  End do
• Output $\mathbf{k} = [k_1, \cdots, k_\ell]$.
```

5.2 Stage II

This stage mainly involves matrix computation. Once the ℓ-degree of the AGCD is identified at Stage I, a Sylvester-like matrix can be constructed:

$$S_{\mathbf{k}}(p,q) = \left[\, C_{\mathbf{m}-\mathbf{k}}(q) \; \middle| \; C_{\mathbf{n}-\mathbf{k}}(p) \, \right], \qquad (17)$$

where the two blocks $C_{\mathbf{m}-\mathbf{k}}(q)$ and $C_{\mathbf{n}-\mathbf{k}}(p)$ are convolution matrices described in §3.2 with $\mathbf{m} = deg(p)$ and $\mathbf{n} = deg(q)$. By Proposition 4, this matrix is rank-deficient by one in an approximate sense [6]. In other words, the smallest singular value ϱ of $S_{\mathbf{k}}(p,q)$ is expected to be tiny while the second smallest one ρ is not. However, computing the full singular value decomposition (SVD) is unnecessarily costly since only ϱ and the associated right singular vector $\mathbf{y}$ are needed. At the same time the full SVD may not be able to take advantage of the sparse structure of $S_{\mathbf{k}}(p,q)$.

A method for partial SVD developed in [6] is designed for calculating the singular pair $(\varrho, \mathbf{y})$. Let $QR = S_{\mathbf{k}}(p,q)$ be the QR decomposition with unitary matrix Q and upper triangular matrix R. From a random initial iterate $\mathbf{y}_0$

$$\left\{ \begin{array}{l} \mathbf{y}_{i+1} = \mathbf{y}_i - \begin{pmatrix} 2\tau\mathbf{y}_i^H \\ R \end{pmatrix}^{+} \begin{pmatrix} \tau\mathbf{y}_i^H\mathbf{y}_i - \tau \\ R\mathbf{y}_i \end{pmatrix} \\ \\ \text{normalize } \; \mathbf{y}_i, \text{ and set } \varrho_i = \| R\mathbf{y}_i \|_2, \\ \qquad i = 0, 1, \cdots, \quad \tau = \|R\|_\infty. \end{array} \right. \qquad (18)$$

This iteration produces two sequences

$$\varrho_1, \varrho_2, \cdots \longrightarrow \varrho, \quad \text{and} \quad \mathbf{y}_1, \mathbf{y}_2, \cdots \longrightarrow \mathbf{y}.$$

The convergence rates are linear for both sequences and depend on the squared ratio between the smallest singular value ϱ and the second smallest one ρ [6, Corollary 1]:

$$\left| \varrho_l - \varrho \right| \le \left(\frac{\varrho^2}{\rho^2} \right)^l \eta, \quad \left\| \mathbf{y}_l - \mathbf{y} \right\|_2 \le \left(\frac{\varrho^2}{\rho^2} \right)^l \mu. \qquad (19)$$

where η and μ are constants depending on the initial iterate $\mathbf{y}_0$. Notice that $S_{\mathbf{k}}(p,q)$ is rank-deficient by one in approximate sense. Namely the ratio ϱ^2/ρ^2 is expected to be very small. Therefore three to five iterations are usually enough to obtain an accurate singular vector $\mathbf{y}$.

The singular vector $\mathbf{y}$ is of dimension $\nu(\mathbf{m}-\mathbf{k}) + \nu(\mathbf{n}-\mathbf{k})$, where the function ν is defined in (4). Once $\mathbf{y}$ is calculated we extract the AGCD co-factors v and w as coefficient vectors. The first $\nu(\mathbf{m}-\mathbf{k})$ components of $\mathbf{y}$ form $\mathbf{v}$ and the remaining components form $-\mathbf{w}$. Then we solve the linear system (15) for its least squares solution $\mathbf{u}$.

In summary, Stage II procedure is as follows.

```
• Input: $p$, $q$, $\mathbf{k}$
• Form the matrix $S_{\mathbf{k}}(p,q)$.
• Get the QR decomposition $QR = S_{\mathbf{k}}(p,q)$.
• For  $l = 0, 1, \cdots$  do
    – Calculate $\varrho_l$ and $\mathbf{y}_l$ by (18).
    – If $\|\mathbf{y}_l - \mathbf{y}_{l-1}\|$ stops decreasing then
          Set $\mathbf{y} = \mathbf{y}_l$, break the do loop
      End if.
  End do
• Extract $\mathbf{v}$ and $\mathbf{w}$ from $\mathbf{y}$
• Solve (15) for $\mathbf{u}$
• Calculate the nearness
      $\theta = \sqrt{\| C_{\mathbf{k}}(v)\mathbf{u} - \mathbf{p} \|_2^2 + \| C_{\mathbf{k}}(w)\mathbf{u} - \mathbf{q} \|_2^2}$
• Output the triplet $(u, v, w)$ and $\theta$
```

In many applications the nearness $\theta \le \varepsilon$ can be enough to accept (u, v, w) as the AGCD triplet and the refinement stage may not be necessary. Nevertheless Stage III below provides an option to ensure the optimal robustness and accuracy in the computed AGCD.

5.3 Stage III

The AGCD triplet (u, v, w) obtained at the end of Stage II is used as the initial iterate (u_0, v_0, w_0) for the Gauss-Newton iteration (16), where F, $\mathbf{z}$ and $\mathbf{b}$ are defined in (11) while the Jacobian $J(\mathbf{z})$ is defined in (12). There is a scaling vector $\mathbf{r}$ that is not yet specified in those definitions. Generally, $\mathbf{r}$ can be any vector that is not perpendicular to the AGCD coefficient vector $\mathbf{u}$. Since u_0 is already an approximation to u, we use $\mathbf{r} = \mathbf{u}_0$ as the scaling vector. With those preparations, we provide the detailed procedure for Stage III.

```
• Input: $\mathbf{p}$, $\mathbf{q}$, $(u_0, v_0, w_0)$, $\mathbf{r}$.
• Form $\mathbf{z}_0$ with $(u_0, v_0, w_0)$ as in (11) along with $\mathbf{b}$.
• For  $j = 0, 1, \cdots$  do
    – Solve  $J(\mathbf{z}_j)\mathbf{d}_j = F(\mathbf{z}_j) - \mathbf{b}$  for $\mathbf{d}_j$ as a least
      squares solution.
    – Set  $\mathbf{z}_{j+1} = \mathbf{z}_j - \mathbf{d}_j$.
    – Calculate the nearness  $\vartheta_{j+1} = \| F(\mathbf{z}_{j+1}) - \mathbf{b} \|_2$.
    – If the nearness  $\vartheta_{j+1}$ stops decreasing, then
          Set  $\mathbf{z} = \mathbf{z}_j$, $\vartheta = \vartheta_j$, break do loop.
      End if
  End do
• Extract $(u, v, w)$ from $\mathbf{z}$.
• Output $(u, v, w)$ and nearness $\vartheta$.
```

In solving $J(\mathbf{z}_j)\mathbf{d}_j = F(\mathbf{z}_j) - \mathbf{b}$ for its least squares solution, the QR decomposition of $J(\mathbf{z}_j)$ is computed and becomes available. At the end of Stage III there is an option to calculate the AGCD condition number with negligible cost. Using the upper triangular matrix R in the QR decomposition of the last $J(\mathbf{z}_j)$. Putting this R into the iteration (18) with a random initial iterate, the sequence ϱ_i converges to the smallest singular value σ_{min} of $J(\mathbf{z}_j)$, while $\kappa = \frac{1}{\sigma_{min}}$ is the AGCD condition number.

5.4 Remarks on sparse computation

The dimension of the vector space $\mathbb{P}_{\mathbf{n}}$ can become huge, particularly when the number of variables increases. For example, the polynomial pair (f, g) in §7 Example 1, case $k = 10$ has an 11-degree $[4, \cdots, 4]$ where the GCD has an 11-degree $[2, \cdots, 2]$. A coefficient vector has a dimension as large as 5^{11}, a personal computer may not be able to store a single convolution matrix. On the other hand, polynomials in question are often very sparse. The same polynomial pair in that example has a manageable total of 2500 nonzero coefficients.

When the given polynomial pair (p, q) is sparse, it is natural to expect the same for its AGCD triplet. Whenever a zero coefficient in the triplet is identified, the corresponding column in matrices (12) and (17) can be deleted. After deleting those columns, a large portion of rows in those matrices can be entirely zero, resulting in matrices of manageable sizes.

There are many possible ways to identify zero coefficients. At the current stage of development, we use a *total degree criterion* in our software package. If $u\, v = p$, then the total degree of u is less than or equal to the total degree of p minus the largest entry in $deg(v)$. Therefore, any monomial in u with an impossible total degree must be associated with a zero coefficient. Same criterion also eliminates coefficients in co-factors v and w. As a result, the matrix $S_{\mathbf{k}}(p, q)$ in the same example shrinks to the size 12376×156. This matrix may still be considered large. Nonetheless it can easily be handled by personal computers.

6. SQUAREFREE FACTORIZATIONS

A GCD-finder naturally leads to methods for squarefree factorizations of polynomials if it is robust enough to handle recursive GCD computation. On the other hand, squarefree factorizations provide a stern test of robustness for any GCD-finder. Here, $p = p_1^{m_1} \cdots p_k^{m_k}$ is called a squarefree factorization of p if none of the p_j's have repeated nonconstant factors. An *approximate squarefree factorization* (ASFF) is, roughly speaking, an exact squarefree factorization of a nearby polynomial.

As a closely related problem, the approximate irreducible factorization is proposed as Open Problem 1 in *Challenges of symbolic computation* [5]. By simplifying this open problem tremendously, an ASFF becomes a prerequisite for approaches such as [3, 4] in finding an irreducible one.

For an ℓ-variate polynomial p and $j \in \{1, \cdots, \ell\}$, let $\partial_j p$ denote the partial derivative of p with respect to the j-th variable. For any vector $\mathbf{a} = (\alpha_1, \cdots, \alpha_\ell)^\top \in \mathbb{C}^\ell$, we define $\partial_{\mathbf{a}} p = (\alpha_1 \partial_1 + \cdots + \alpha_\ell \partial_\ell)\, p$. The following proposition establishes a squarefree test criterion.

Proposition 6. *Let p be an ℓ-variate polynomial. If there is an $\mathbf{a} \in \mathbb{C}^\ell$ such that $GCD(p, \partial_{\mathbf{a}} p) = 1$, then p is squarefree. Conversely, if p is squarefree then $GCD(p, \partial_{\mathbf{a}} p) = 1$ holds for almost all $\mathbf{a} \in \mathbb{C}^\ell$.*

We propose two algorithms for calculating ASFF.

ASFF Algorithm I.

```
• Input: polynomial $p$, tolerance $\varepsilon$.
• Set $u_0 = p$, $n = 0$, and a random vector $\mathbf{a} \in \mathbb{C}^\ell$.
• For $k = 1, 2 \cdots$ do
    – Find AGCD triplet $(u_k, v_k, w_k)$ of $u_{k-1}$ and $\partial_{\mathbf{a}} u_{k-1}$ within $\varepsilon$.
    – If $\partial_{\mathbf{a}} u_k = 0$, break do loop, end if.
  End do.
• Output: ASFF $v_1 v_2 \cdots v_k$.
```

ASFF Algorithm II.

```
• Input: polynomial $p$, tolerance $\varepsilon$.
• Set a random vector $\mathbf{a} \in \mathbb{C}^\ell$.
• Calculate an AGCD triplet $(u_0, v_0, w_0)$ for $p$ and $\partial_{\mathbf{a}} p$.
• For $k = 1, 2 \cdots$ do
    – Find AGCD triplet $(h_k, v_k, w_k)$ of $v_{k-1}$ and $w_{k-1} - \partial_{\mathbf{a}} v_{k-1}$ within $\varepsilon$.
    – If $\partial_{\mathbf{a}} w_k = 0$, break do loop, end if.
  End do.
• Output: ASFF $h_1^1 h_2^2 \cdots h_k^k$.
```

Two algorithms produce conjugate ASFF's. For example, if the given polynomial has an irreducible factorization $p = p_1^5 p_2^3 p_3^3 p_4$, Algorithm I and II seek the squarefree factorizations

$$
\begin{aligned}
p &= (p_1 p_2 p_3 p_4)(p_1 p_2 p_3)(p_1 p_2 p_3)(p_1)(p_1) \qquad (20)\\
&= (p_4)^1 (1)^2 (p_2 p_3)^3 (1)^4 (p_1)^5 \qquad (21)
\end{aligned}
$$

respectively. If necessary, ASFF (20) can be converted to ASFF (21) by least square divisions, while a rearrangement of ASFF (21) leads to ASFF (20).

Both algorithms are tested in §7, Example 6.

Remark 1. If there is only $\ell = 1$ variable, ASFF Algorithm I is identical to the recursive factorization process in [14], while Algorithm II reduces to an algorithm that is equivalent to a process proposed in [12].

Remark 2. We propose the use of a random vector $\mathbf{a}$ in both algorithms to ensure squarefreeness. Some previous works seek squarefreeness via $GCD(p, \partial_1 p) = 1$. Methods can be designed for ASFF via computing the AGCD of $(p, \partial_1 p)$ instead of $(p, \partial_{\mathbf{a}} p)$. We use the latter for its certainty that the co-factor v consists of all irreducible factors of p. If Algorithm II uses a particular ∂_j, say ∂_1 instead of $\partial_{\mathbf{a}}$, it would fail on polynomials like $p = (x_2 + 2)^2 (x_1 + x_2)$.

7. COMPUTATIONAL EXPERIMENT

The algorithm presented in this paper has been implemented in both Maple and Matlab as a blackbox package mvGCD along with the univariate package uvGCD [13]. The only required input items are the polynomial pair (f, g) and the tolerance ε. The priority at current stage of development is to ensure the highest possible robustness and accuracy rather than fast computation. We test the package for the following capabilities:

1. Finding AGCD's of exact polynomials.
2. Finding AGCD's of high sensitivity.
3. Finding AGCD's of many variables.
4. Finding AGCD's of high degrees.
5. Finding AGCD's within different tolerances.
6. Finding AGCD's with various magnitudes of noises.

7. Finding AGCD's in recursive computation.

We have been collecting polynomial pairs to establish a test suite. Our current package and test suite are electronically available from the first author upon request.

We present sample results in this section. Those tests are conducted using Matlab 6.1 on a desktop personal computer with Intel Pentium 4 CPU (1.8 GHz) and 512 Mb memory. For a given pair (p, q) of ℓ-degree $\mathbf{m}$ and $\mathbf{n}$ respectively with computed AGCD triplet (u, v, w), the backward error is measured by

$$b\_err = \sqrt{\frac{\| \Psi_{\mathbf{m}}(u\,v - p) \|_2^2 + \| \Psi_{\mathbf{n}}(u\,w - q) \|_2^2}{\| \Psi_{\mathbf{m}}(p) \|_2^2 + \| \Psi_{\mathbf{n}}(q) \|_2^2}}$$

If the exact GCD triplet $(\hat{u}, \hat{v}, \hat{w})$ is known, then the forward error is calculated as

$$f\_err = \sqrt{\frac{\| \mathbf{u} - \hat{\mathbf{u}} \|_2^2 + \| \mathbf{v} - \hat{\mathbf{v}} \|_2^2 + \| \mathbf{w} - \hat{\mathbf{w}} \|_2^2}{\| \hat{\mathbf{u}} \|_2^2 + \| \hat{\mathbf{v}} \|_2^2 + \| \hat{\mathbf{w}} \|_2^2}}$$

All timing measures are in seconds.

Example 1 [7, §7, Case 2]. For $k = 1, 2 \cdots, 10$, let

$$\begin{cases} u_k = (x_0 + x_1 + \cdots + x_k + 1)^2 \\ v_k = (x_0 - x_1 - \cdots - x_k - 2)^2 \\ w_k = (x_0 + x_1 + \cdots + x_k + 2)^2 \end{cases}, \quad \begin{cases} p_k = u_k\, v_k \\ q_k = u_k\, w_k \end{cases}.$$

The objective is to test the capability of the algorithm in handling increasing number of variables and finding the AGCD of the exact polynomial pairs (p_k, q_k). The accuracy and timing are listed in Table 1.

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
time (seconds)	0.062	0.141	0.359	0.766	1.734
backward error	4e-16	6e-16	7e-16	8e-16	1e-15
forward error	8e-16	2e-15	2e-15	2e-15	2e-15
	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
time (seconds)	4.047	9.860	25.20	64.86	179.2
backward error	2e-15	2e-15	4e-15	3e-15	5e-15
forward error	8e-15	1e-14	2e-14	2e-14	3e-14

Table 1: Results for Example 1.

Example 2 [7, §7, Case 7]. For integer pairs j, k, let

$$p_{jk} = f^j\, g^k, \quad q_{jk} = f^k\, g^j, \quad \text{where} \quad f = x - yz + 1, \quad g = x - y + 3z.$$

The objective is to test the algorithm for its capability in handling increasing degrees while the AGCD is not co-prime with its co-factors. AGCD methods involving Hensel-lifting [16, 17] may have difficulty in this case. Table 2 lists the results in finding AGCD of various pairs (p_{jk}, q_{jk}).

(j,k)	(1,3)	(1,4)	(2,4)	(2,5)	(3,4)
time (seconds)	0.296	0.703	0.640	1.594	0.781
backward error	3e-16	4e-16	3e-16	6e-16	5e-16
forward error	4e-16	6e-16	3e-16	2e-15	3e-16
(j,k)	(3,5)	(4,6)	(5,8)	(5,10)	(6,12)
time (seconds)	1.485	4.187	19.72	57.78	183.3
backward error	5e-16	8e-16	9e-16	3e-15	3e-14
forward error	7e-16	1e-15	2e-15	4e-14	2e-13

Table 2: Results for Example 2.

Example 3. For $n = 10, 20, 30, 40$, let u_n be a dense bivariate polynomial of bidegree $[n, n]$ with random rational coefficients, We test the function GCD in Maple 9 and our MVGCD on

$$p_n = u_n\, v; \quad q_n = u_n\, w, \quad \text{where} \quad v = 1 + x + y + xy, \quad w = 1 - x + y - xy + y^2$$

The bivariate polynomial pairs (p_n, q_n) have increasing degrees in EGCD. This is one of the cases where Maple's symbolic GCD-finder is not as efficient as MVGCD. Maple GCD are tested on the pairs with exact rational coefficients while MVGCD is tested on approximate data. Table 3 lists the execution time and accuracy of both Maple GCD and our MVGCD. Notice that Maple's symbolic GCD reported error messages for cases $n = 30, 40$ without correct GCD after relatively long running time.

	Maple GCD			MVGCD		
	time	b_err	f_err	time	b_err	f_err
$n=10$	2.8	0	0	0.406	2e-15	1e-15
$n=20$	69.3	0	0	2.53	2e-15	1e-15
$n=30$	254	fail		12.89	1e-14	3e-15
$n=40$	691	fail		48.70	1e-14	1e-15

Table 3: Results for Example 3.

Example 4. For $k = 3, 4, \cdots, 9$, let

$$p_k = (xy + y^2 + 2)(xy + y + 2 + 10^{-k}x)(1 + 2x^3 - 2xy + 3y^2)$$
$$q_k = (xy + y^2 + 2)(xy + y + 2 - 10^{-k}x)(1 - 2x^3 + 2xy - 3y^2).$$

For each k, there is an AGCD $\check{u} = xy + y + 2$ within a "tight" tolerance $\check{\varepsilon} < 10^{-k}$ and an AGCD $\hat{u}$ near $\check{u}^2$ within a "loose" tolerance $\hat{\varepsilon} > 10^{-k}$. Also, within the tight tolerance, the AGCD condition number $\check{\kappa}$ increases proportionally to 10^k, causing the forward accuracy deteriorates. On the other hand, the AGCD within the loose tolerance has a healthy conditioned number $\hat{\kappa}$ that is nearly a constant for all k's.

Table 4 shows the results for increasing k on both types of tolerances. Notice that the forward errors are calculated on different sets of AGCD triplets. Our MVGCD shows considerable robustness in handling sensitive AGCD's and accurately attains the AGCD triplet for k up to nine. Also notice that, when sensitivity increases, the forward error deteriorates but the tiny backward error remains unaffected.

	within tight $\check{\varepsilon} < 10^{-k}$			within loose $\hat{\varepsilon} > 10^{-k}$		
	$\check{\kappa}$	b_err	f_err	$\hat{\kappa}$	b_err	f_err
$k=3$	1e04	5e-16	2e-14	9.528	7e-03	3e-04
$k=4$	1e05	4e-16	8e-14	9.528	8e-04	3e-05
$k=5$	1e06	3e-16	3e-13	9.528	8e-05	6e-07
$k=6$	1e07	5e-16	3e-11	9.528	8e-06	6e-08
$k=7$	1e08	4e-16	2e-10	9.528	8e-07	6e-09
$k=8$	4e08	5e-16	3e-09	9.528	8e-08	6e-10
$k=9$	4e08	5e-16	4e-09	9.528	8e-09	6e-11

Table 4: Results for Example 4.

Example 5. Let (p_3, q_3) be the 4-variate polynomial pairs defined in in Example 1. A 5-term "noise" polynomial pair $(\tilde{f}_j, \tilde{g}_j)$ is generated with random coefficients in the interval $[-\beta_j, \beta_j]$ where $\beta_j = -9 \times 10^{j-10}$. Then the test polynomial pairs (f_j, g_j) are defined as

$$f_j = p_3 + \tilde{f}_j, \quad g_j = q_3 + \tilde{g}_j, \quad j = 1, 2, \cdots, 7.$$

These series of polynomial pairs test the robustness of our AGCD finder when the data is under perturbation of increasing magnitudes. The AGCD's are insensitive with condition numbers around 14. Table 5 shows that not only MVGCD roughly identifies the noise magnitude in backward error, but it also consistently calculates the AGCD's that are at least as accurate as the given data for j up to seven. The code fails for $j \geq 8$ when the data has no more than two correct digits in each coefficient.

	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
b_err	1e-7	2e-6	1e-5	1e-4	1e-03	2e-02
f_err	3e-9	5e-8	3e-8	1e-6	4e-05	1e-04

Table 5: **Results for Example 5.**

Example 6. Let $p = p_1^4 p_2^3 p_3 + 10^{-8}\tilde{p}$ where

$$p_1 = x^2 + xy - y^2 + 1,$$
$$p_2 = (x^3+1)(y^2-2) = x^3y^2 + y^2 - 2x^3 - 2,$$
$$p_3 = x^3 - y^3 - 3xy^2 + 2,$$

and $\tilde{p}$ is a 10-term random polynomial with coefficients in $[-5, 5]$. The experiment tests the approximate squarefree factorization (ASFF) Algorithm I and II under perturbation. Also, this experiment tests the robustness of MVGCD under recursive application.

The polynomial p has 271 terms with coefficient magnitudes ranging from 0 to 5648. ASFF Algorithm I is expected to output v_1, v_2, v_3 and v_4 that approximate $p_1p_2p_3$, p_1p_2, p_1p_2 and p_1 respectively, while ASFF Algorithm II should produces $h_1 h_2^2 h_3^3 h_4^4$ where h_1, h_2, h_3 and h_4 are expected to approximate p_3, 1, p_2 and p_1 respectively. Table 6 lists the forward error measures on the results of both algorithms.

ASFF Algorithm I forward error at				ASFF Algorithm II forward error at			
v_1	v_2	v_3	v_4	h_1	h_2	h_3	h_4
3.8e-8	2.9e-7	3.1e-7	1.1e-7	3.4e-8	0	5.0e-8	1.1e-7

Table 6: **Results for Example 6.**

The actual polynomial factors from ASFF Algorithm II are

$$\begin{aligned} h_1 &= x^3 - 0.999999967y^3 - 2.99999969xy^2 + 1.99999975 \\ h_2 &= 1 \\ h_3 &= x^3y^2 + 0.999999912y^2 - 2.000000057x^3 - 2.00000016 \\ h_4 &= x^2 + 0.99999976xy - 1.000000055y^2 + 0.999999935 \end{aligned}$$

with terms of tiny coefficients omitted. The ASFF has at least 7 digits correct in every coefficient. The outcome of ASFF Algorithm I shows similar accuracy.

8. REFERENCES

[1] W. S. Brown. On Euclid algorithm and the computation of polynomial greatest common divisors. *J. Assoc. Comput. Mach.*, 18:478–504, 1971.

[2] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. Proc. ISSAC '95, ACM Press, pp 195-207, 1995.

[3] R. M. Corless, M. Giesbrecht, M. van Hoeij, I. Kotsireas, and S. Watt. Towards factoring bivariate approximate polynomials. Proc. of ISSAC'01, ACM Press, pages 85-92, 2001.

[4] S. Gao. Factoring multivariate polynomials via partial differential equations. *Math. Comp.*, 72:801–822, 2003.

[5] E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symb. Comput.*, 29:161–168, 2000.

[6] T. Y. Li and Z. Zeng. A rank-revealing method and its applications. Preprint, 2003.

[7] H.-C. Liao and R. J. Fateman. Evaluation of the heuristic polynomial GCD. Proc. of ISSAC'95, ACM, 1995.

[8] M.-A. Ochi, M. Noda, and T. Sasaki. Approximate greatest common divisor of multivariate polynomials and its application to ill-conditioned system of algebraic equations. *J. Inf. Proces*, 12:292–300, 1991.

[9] S. Pillai and B. Liang. Blind image deconvolution using GCD approach. *IEEE Trans. Image Processing*, 8:202–219, 1999.

[10] T. Sasaki and M. Sasaki. Polynomial remainder sequence and approximate GCD. *ACM SIGSAM Bulletin*, 31:4–10, 1997.

[11] F. Šroubek and J. Flusser. Multichannel blind iterative image restoration. *IEEE Trans. Image Processing*, 9:1094–1106, 2003.

[12] D. Y. Y. Yun. On square-free decomposition algorithms. Proc. of ISSAC '76, ACM Press, pp 26-35, 1976.

[13] Z. Zeng. The approximate GCD of inexact polynomials. Part I: a univariate algorithm. Manuscript, 2004.

[14] Z. Zeng. A method computing multiple roots of inexact polynomials. Proc. of ISSAC '03, ACM Press, pp 266-272, 2003 full Journal version to appear in *Math. Comp.*

[15] Z. Zeng. Multroot – a Matlab package for computing polynomial roots and multiplicities. ACM Trans. Math. Software, to appear.

[16] L. Zhi, K. Li, and M.-T. Noda. Approximate GCD of multivariate polynomials using Hensel lifting. MM Research Preprint 241-248, MMRC, AMSS, Academia, Sinica, Beijing, China, No. 21, 2001, 2001.

[17] L. Zhi and M.-T. Noda. Approximate GCD of multivariate polynomials. Proc. ASCM 2000, World Scientific Press, pages 9-18, 2000.