An Iterative Method for Calculating Approximate GCD of Univariate Polynomials

Akira Terui
Graduate School of Pure and Applied Sciences
University of Tsukuba
Tsukuba, 305-8571 Japan
terui@math.tsukuba.ac.jp

ABSTRACT

We present an iterative algorithm for calculating approximate greatest common divisor (GCD) of univariate polynomials with the real coefficients. For a given pair of polynomials and a degree, our algorithm finds a pair of polynomials which has a GCD of the given degree and whose coefficients are perturbed from those in the original inputs, making the perturbations as small as possible, along with the GCD. The problem of approximate GCD is transfered to a constrained minimization problem, then solved with a so-called modified Newton method, which is a generalization of the gradient-projection method, by searching the solution iteratively. We demonstrate that our algorithm calculates approximate GCD with perturbations as small as those calculated by a method based on the structured total least norm (STLN) method, while our method runs significantly faster than theirs by approximately up to 30 times, compared with their implementation. We also show that our algorithm properly handles some ill-conditioned problems with GCD containing small or large leading coefficient.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Symbolic and Algebraic Manipulation—*Algorithms*; G.1.2 [**Mathematics of Computing**]: Numerical Analysis—*Approximation*

General Terms

Algorithms, experimentation

Keywords

Approximate polynomial GCD, gradient-projection method, ill-conditioned problem, optimization

1. INTRODUCTION

For algebraic computations on polynomials and matrices, approximate algebraic algorithms are attracting broad range of attentions recently. These algorithms take inputs with some "noise" such as polynomials with floating-point number coefficients with rounding errors, or more practical errors such as measurement errors, then, with minimal changes on the inputs, seek a meaningful answer that reflect desired property of the input, such as a common factor of a given degree. By this characteristic, approximate algebraic algorithms are expected to be applicable to more wide range of problems, especially those to which exact algebraic algorithms were not applicable.

As an approximate algebraic algorithm, we consider calculating the approximate greatest common divisor (GCD) of univariate polynomials with the real coefficients, such that, for a given pair of polynomials and a degree d, finding a pair of polynomials which has a GCD of degree d and whose coefficients are perturbations from those in the original inputs, with making the perturbations as small as possible, along with the GCD. This problem has been extensively studied with various approaches including the Euclidean method on the polynomial remainder sequence (PRS) ([1], [16], [17]), the singular value decomposition (SVD) of the Sylvester matrix ([3], [6]), the QR factorization of the Sylvester matrix or its displacements ([4], [20], [22]), Padé approximation [13], optimization strategies ([2], [8], [9], [10], [21]). Furthermore, stable methods for ill-conditioned problems have been discussed ([4], [12], [15]).

Among methods in the above, we focus our attention on optimization strategy in this paper, especially iterative method for approaching an optimal solution, after transferring the approximate GCD problem into a constrained minimization problem. Already proposed algorithms utilize iterative methods including the Levenberg-Marquardt method [2], the Gauss-Newton method [21] and the structured total least norm (STLN) method ([8], [9]). Among them, STLN-based methods have shown good performance calculating approximate GCD with sufficiently small perturbations efficiently.

Here, we utilize a so-called modified Newton method [18], which is a generalization of the gradient-projection method [14], for solving the constrained minimization problem. This method has interesting features such that it combines the *projection* and the *restoration* steps in the original gradient-projection method, which reduces the number of solving a linear system. We demonstrate that our algorithm calculates approximate GCD with perturbations as small as those calculated by the STLN-based methods, while our method show significantly better performance over them in its speed compared with their implementation, by approximately up

http://doi.acm.org/10.1145/1576702.1576750

to 30 times. Furthermore, we also show that our algorithm can properly handle some ill-conditioned problems such as those with GCD containing small or large leading coefficient.

The rest part of the paper is organized as follows. In Section 2, we transform the approximate GCD problem into a constrained minimization problem. In Section 3, we review the framework of the gradient-projection method and a modified Newton method. In Section 4, we show an algorithm for calculating the approximate GCD, with discussing issues in the application of the gradient-projection method or a modified Newton method. In Section 5, we demonstrate performance of our algorithm with experiments.

2. FORMULATION OF THE APPROXIMATE GCD PROBLEM

Let $F(x)$ and $G(x)$ be univariate polynomials with real coefficients, given as

$$\begin{aligned} F(x) &= f_m x^m + f_{m-1} x^{m-1} + \cdots + f_0, \\ G(x) &= g_n x^n + g_{n-1} x^{n-1} + \cdots + g_0, \end{aligned} \tag{1}$$

with $m \geq n > 0$ and relatively prime in general. For a given degree d satisfying $n \geq d > 0$, let us calculate a deformation of $F(x)$ and $G(x)$ in the form of

$$\begin{aligned} \tilde{F}(x) &= F(x) + \Delta F(x) = H(x) \cdot \bar{F}(x), \\ \tilde{G}(x) &= G(x) + \Delta G(x) = H(x) \cdot \bar{G}(x), \end{aligned} \tag{2}$$

where $\Delta F(x)$, $\Delta G(x)$ are polynomials whose degrees do not exceed those of $F(x)$ and $G(x)$, respectively, $H(x)$ is a polynomial of degree d, and $\bar{F}(x)$ and $\bar{G}(x)$ are pairwise relatively prime. If we find $\tilde{F}$, $\tilde{G}$, $\bar{F}$, $\bar{G}$ and H satisfying (2), then we call *H an approximate GCD of F and G*. For a given degree d, we tackle the problem of finding an approximate GCD H with minimizing the norm of the deformations $\|\Delta F(x)\|_2^2 + \|\Delta G(x)\|_2^2$. In this paper, we search for all the polynomials with the real coefficients.

In the case $\tilde{F}(x)$ and $\tilde{G}(x)$ have a GCD of degree d, then the theory of subresultants tells us that the subresultant of $\tilde{F}$ and $\tilde{G}$ of degree $d-1$ becomes zero, namely we have

$$\mathrm{S}_{d-1}(\tilde{F}, \tilde{G}) = 0,$$

where $\mathrm{S}_k(\tilde{F}, \tilde{G})$ denotes the subresultant of $\tilde{F}$ and $\tilde{G}$ of degree k. Then, the $(d-1)$-th subresultant matrix

$$N_{d-1}(\tilde{F}, \tilde{G}) = \begin{pmatrix} \tilde{f}_m & & & \tilde{g}_n & & \\ \vdots & \ddots & & \vdots & \ddots & \\ \tilde{f}_0 & & \tilde{f}_m & \tilde{g}_0 & & \tilde{g}_n \\ & \ddots & \vdots & & \ddots & \vdots \\ & & \tilde{f}_0 & & & \tilde{g}_0 \end{pmatrix}, \tag{3}$$

(with $n-d+1$ columns for $\tilde{f}$ and $m-d+1$ columns for $\tilde{g}$)

where the k-th subresultant matrix $N_k(\tilde{F}, \tilde{G})$ is a submatrix of the Sylvester matrix $N(\tilde{F}, \tilde{G})$ by taking the left $n-k$ columns of coefficients of $\tilde{F}$ and the left $m-k$ columns of coefficients of $\tilde{G}$, has a kernel of dimension equal to 1. Thus, there exist polynomials $A(x), B(x) \in \mathbf{R}[x]$ satisfying

$$A\tilde{F} + B\tilde{G} = 0, \tag{4}$$

with $\deg(A) = n-d$ and $\deg(B) = m-d$ and $A(x)$ and $B(x)$ are relatively prime. Therefore, for the given $F(x)$, $G(x)$ and d, our problem is to find $\Delta F(x)$, $\Delta G(x)$, $A(x)$ and $B(x)$ satisfying Eq. (4) with making $\|\Delta F\|_2^2 + \|\Delta G\|_2^2$ as small as possible.

By representing $\tilde{F}(x)$, $\tilde{G}(x)$, $A(x)$ and $B(x)$ as

$$\begin{aligned} \tilde{F}(x) &= \tilde{f}_m x^m + \cdots + \tilde{f}_0 x^0, \\ \tilde{G}(x) &= \tilde{g}_n x^n + \cdots + \tilde{g}_0 x^0, \\ A(x) &= a_{n-d} x^{n-d} + \cdots + a_0 x^0, \\ B(x) &= b_{m-d} x^{m-d} + \cdots + b_0 x^0, \end{aligned} \tag{5}$$

We express Eq. (4) and $\|\Delta F\|_2^2 + \|\Delta G\|_2^2$ as

$$N_{d-1}(\tilde{F}, \tilde{G}) \cdot {}^t(a_{n-d}, \ldots, a_0, b_{m-d}, \ldots, b_0) = \mathbf{0}, \tag{6}$$

$$\begin{aligned} \|\Delta F\|_2^2 + \|\Delta G\|_2^2 = (\tilde{f}_m - f_m)^2 + \cdots + (\tilde{f}_0 - f_0)^2 \\ + (\tilde{g}_n - g_n)^2 + \cdots + (\tilde{g}_0 - g_0)^2, \end{aligned} \tag{7}$$

respectively. Then, Eq. (6) is regarded as a system of $m+n-d+1$ equations in $\tilde{f}_m, \ldots, \tilde{f}_0, \tilde{g}_n, \ldots, \tilde{g}_0, a_{n-d}, \ldots, a_0, b_{m-d}, \ldots, b_0$, as

$$\begin{aligned} g_1 &= \tilde{f}_m a_{n-d} + \tilde{g}_n b_{m-d} = 0, \\ &\vdots \\ g_{m+n-d+1} &= \tilde{f}_0 a_0 + \tilde{g}_0 b_0 = 0, \end{aligned} \tag{8}$$

by putting g_j as the j-th row. Furthermore, for solving the problem below stably, we add another constraint enforcing the coefficients of $A(x)$ and $B(x)$ such that $\|A(x)\|_2^2 + \|B(x)\|_2^2 = 1$; thus we add

$$g_0 = a_{n-d}^2 + \cdots + a_0^2 + b_{m-d}^2 + \cdots + b_0^2 - 1 = 0 \tag{9}$$

into Eq. (8).

Now, we substitute the variables

$$(\tilde{f}_m, \ldots, \tilde{f}_0, \tilde{g}_n, \ldots, \tilde{g}_0, a_{n-d}, \ldots, a_0, b_{m-d}, \ldots, b_0) \tag{10}$$

as $\boldsymbol{x} = (x_1, \ldots, x_{2(m+n-d+2)})$, thus Eq. (7) and (8) with (9) become

$$\begin{aligned} f(\boldsymbol{x}) = (x_1 - f_m)^2 + \cdots + (x_{m+1} - f_0)^2 \\ + (x_{m+2} - g_n)^2 + \cdots + (x_{m+n+2} - g_0)^2, \end{aligned} \tag{11}$$

$$\boldsymbol{g}(\boldsymbol{x}) = {}^t(g_0(\boldsymbol{x}), g_1(\boldsymbol{x}), \ldots, g_{m+n-d+1}(\boldsymbol{x})) = \mathbf{0}, \tag{12}$$

respectively. Therefore, the problem of finding an approximate GCD can be formulated as a constrained minimization problem of finding a minimizer of the objective function $f(\boldsymbol{x})$ in (11), subject to $\boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}$ in Eq. (12).

3. THE GRADIENT-PROJECTION METHOD AND A MODIFIED NEWTON METHOD

In this section, we consider the problem of minimizing an objective function $f(\boldsymbol{x}) : \mathbf{R}^n \to \mathbf{R}$, subject to the constraints $\boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}$ for $\boldsymbol{g}(\boldsymbol{x}) = {}^t(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x}))$, with $m \leq n$, where $g_j(\boldsymbol{x})$ is a function of $\mathbf{R}^n \to \mathbf{R}$, and $f(\boldsymbol{x})$ and $g_j(\boldsymbol{x})$ are twice continuously differentiable (here, we refer presentations of the problem to Tanabe [18] and the references therein).

If we assume that the Jacobian matrix

$$J_{\boldsymbol{g}}(\boldsymbol{x}) = \left(\frac{\partial g_i}{\partial x_j} \right)$$

is of full rank, or

$$\mathrm{rank}(J_{\boldsymbol{g}}(\boldsymbol{x})) = m, \tag{13}$$

on the feasible region $V_{\boldsymbol{g}}$ defined by $V_{\boldsymbol{g}} = \{\boldsymbol{x} \in \mathbf{R}^n \mid \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}\}$, then the feasible region $V_{\boldsymbol{g}}$ is an $(n-m)$-dimensional differential manifold in $\mathbf{R}^n$ and f is differentiable function on the manifold $V_{\boldsymbol{g}}$. Thus, our problem is to find a point in $V_{\boldsymbol{g}}$, which will be a candidate of a local minimizer, satisfying the well-known "first-order necessary conditions" (for the proof, refer to the literature on optimization [11]).

THEOREM 1 (FIRST-ORDER NECESSARY CONDITIONS). *Suppose that $\boldsymbol{x}^* \in V_{\boldsymbol{g}}$ is a local solution of the problem in the above, that the functions $f(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ are continuously differentiable at $\boldsymbol{x}^*$, and that we have* (13) *at $\boldsymbol{x}^*$. Then, there exist a* Lagrange multiplier vector $\boldsymbol{\lambda}^* \in \boldsymbol{R}^m$ *satisfying*

$$\nabla f(\boldsymbol{x}^*) - {}^t(J_{\boldsymbol{g}}(\boldsymbol{x}^*))\boldsymbol{\lambda}^* = \boldsymbol{0}, \quad \boldsymbol{g}(\boldsymbol{x}^*) = \boldsymbol{0}. \quad \square$$

3.1 The Gradient-Projection Method

Let $\boldsymbol{x}_k \in \mathbf{R}^n$ be a feasible point, or a point satisfying $\boldsymbol{x}_k \in V_{\boldsymbol{g}}$. Rosen's gradient projection method [14] is based on projecting the steepest descent direction onto the tangent space of the manifold $V_{\boldsymbol{g}}$ at $\boldsymbol{x}_k$, which is denoted to $T_{\boldsymbol{x}_k}$ and represented by the kernel of the Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x}_k)$ as

$$T_{\boldsymbol{x}_k} = \ker(J_{\boldsymbol{g}}(\boldsymbol{x}_k)) = \{\boldsymbol{z} \in \mathbf{R}^n \mid J_{\boldsymbol{g}}(\boldsymbol{x}_k)\boldsymbol{z} = \boldsymbol{0} \in \mathbf{R}^m\}. \tag{14}$$

We have steepest descent direction of the objective function f at $\boldsymbol{x}_k$ as

$$-\nabla f(\boldsymbol{x}_k) = -{}^t\left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right). \tag{15}$$

Then, the search direction $\boldsymbol{d}_k$ is defined by the projection of the steepest descent direction of f in (15) onto $T_{\boldsymbol{x}_k}$ in (14) as

$$\boldsymbol{d}_k = -P(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k). \tag{16}$$

Here, $P(\boldsymbol{x}_k)$ is the orthogonal projection operator on $T_{\boldsymbol{x}_k}$ defined as $P(\boldsymbol{x}_k) = I - (J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+(J_{\boldsymbol{g}}(\boldsymbol{x}_k))$, where I is the identity matrix and $(J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+$ is the Moore-Penrose inverse of $(J_{\boldsymbol{g}}(\boldsymbol{x}_k))$. Under the assumption (13), we have $(J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+ = {}^t(J_{\boldsymbol{g}}(\boldsymbol{x}_k)) \cdot ({}^t(J_{\boldsymbol{g}}(\boldsymbol{x}_k)))^{-1}$.

With an appropriate step width α_k (in this paper, we omit how to calculate α_k in detail) satisfying $0 < \alpha_k \leq 1$, let $\boldsymbol{y}_k = \boldsymbol{x}_k + \alpha_k \cdot \boldsymbol{d}_k$. Since $V_{\boldsymbol{g}}$ is nonlinear in general, $\boldsymbol{y}_k$ may not in $V_{\boldsymbol{g}}$: in such a case, we take a *restoration* move to bring $\boldsymbol{y}_k$ back to $V_{\boldsymbol{g}}$, as follows. Let $\boldsymbol{x} \in \mathbf{R}^n$ be an arbitrary point. Then, at $\boldsymbol{y}_k$, the constraint $\boldsymbol{g}(\boldsymbol{x})$ can be linearly approximated as $\boldsymbol{g}(\boldsymbol{y}_k + \boldsymbol{x}) \simeq \boldsymbol{g}(\boldsymbol{y}_k) + J_{\boldsymbol{g}}(\boldsymbol{y}_k)\boldsymbol{x}$. Assuming $\boldsymbol{y}_k + \boldsymbol{x} \in V_{\boldsymbol{g}}$, we have $\boldsymbol{g}(\boldsymbol{y}_k + \boldsymbol{x}) = \boldsymbol{0}$ thus the approximation of $\boldsymbol{x}$ can be calculated as

$$\boldsymbol{x} = -(J_{\boldsymbol{g}}(\boldsymbol{y}_k))^+\boldsymbol{g}(\boldsymbol{y}_k). \tag{17}$$

If $\boldsymbol{y}_k$ is sufficiently close to $V_{\boldsymbol{g}}$, then we can restore $\boldsymbol{y}_k$ back onto $V_{\boldsymbol{g}}$ by applying (17) iteratively for several times. Note that the restoration move can also be used in the case the initial point of the minimization process is away from the feasible region $V_{\boldsymbol{g}}$.

Summarizing the above, we obtain an algorithm for the gradient projection as follows.

ALGORITHM 1. (THE GRADIENT-PROJECTION METHOD [14])

Step 1 **[Restoration]** If the given point $\boldsymbol{x}_0$ does not satisfy $\boldsymbol{x}_0 \in V_{\boldsymbol{g}}$, first move $\boldsymbol{x}_0$ onto $V_{\boldsymbol{g}}$ by the iteration of Eq. (17), then let $\boldsymbol{x}_0$ be the restored point on $V_{\boldsymbol{g}}$. Let $k = 0$.

Step 2 **[Projection]** For $\boldsymbol{x}_k$, calculate $\boldsymbol{d}_k = -P(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)$ by (16). If $\|\boldsymbol{d}_k\|$ is sufficiently small for an appropriate norm, go to Step 4. Otherwise, calculate the step width α_k by an appropriate line search method (we omit its detail here) then let $\boldsymbol{y}_{k,0} = \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k$.

Step 3 **[Restoration]** If $\boldsymbol{g}(\boldsymbol{y}_{k,0}) \neq \boldsymbol{0}$, move $\boldsymbol{y}_{k,0}$ back onto $V_{\boldsymbol{g}}$ iteratively by (17). Let $\boldsymbol{y}_{k,l+1} = -(J_{\boldsymbol{g}}(\boldsymbol{y}_{k,l}))^+\boldsymbol{g}(\boldsymbol{y}_{k,l})$ for $l = 0, 1, 2, \ldots$. When $\boldsymbol{y}_{k,l}$ satisfies $\boldsymbol{g}(\boldsymbol{y}_{k,l})$, then let $\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k,l}$ and go to Step 2.

Step 4 **[Checking the first-order necessary conditions]** If $\boldsymbol{x}_k$ satisfies Theorem 1, then return $\boldsymbol{x}_k$.

3.2 A Modified Newton Method

A modified Newton method by Tanabe [18] is a generalization of the Newton's method, which derives several different methods, by modifying the Hessian of the Lagrange function. A generalization of the gradient-projection method combines the *restoration step* and the *projection step* in Algorithm 1. For $\boldsymbol{x}_k \in V_{\boldsymbol{g}}$, we calculate the search direction $\boldsymbol{d}_k$, along with the associated Lagrange multipliers $\boldsymbol{\lambda}_{k+1}$, by solving a linear system

$$\begin{pmatrix} I & -{}^t(J_{\boldsymbol{g}}(\boldsymbol{x}_k)) \\ J_{\boldsymbol{g}}(\boldsymbol{x}_k) & \boldsymbol{O} \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_k \\ \boldsymbol{\lambda}_{k+1} \end{pmatrix} = -\begin{pmatrix} \nabla f(\boldsymbol{x}_k) \\ \boldsymbol{g}(\boldsymbol{x}_k) \end{pmatrix}, \tag{18}$$

then put $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \cdot \boldsymbol{d}_k$ with an appropriate step width α_k. Solving Eq. (18) under assumption (13), we have

$$\begin{aligned} \boldsymbol{d}_k &= -P(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k) - (J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+\boldsymbol{g}(\boldsymbol{x}_k), \\ \boldsymbol{\lambda}_{k+1} &= {}^t((J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+)\nabla f(\boldsymbol{x}_k) \\ &\quad - (J(\boldsymbol{x}_k) \cdot {}^t(J(\boldsymbol{x}_k)))^{-1}\boldsymbol{g}(\boldsymbol{x}_k). \end{aligned} \tag{19}$$

Note that, in $\boldsymbol{d}_k$ in (19), the term $-P(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)$ comes from the projection (16), while another term $-(J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+\boldsymbol{g}(\boldsymbol{x}_k)$ comes from the restoration (17). If we have $\boldsymbol{x}_k \in V_{\boldsymbol{g}}$, the iteration formula (18) is equivalent to the projection (16). After an iteration, the new estimate $\boldsymbol{x}_{k+1}$ may not satisfy $\boldsymbol{x}_{k+1} \in V_{\boldsymbol{g}}$: in such a case, in the next iteration, the point will be pulled back onto $V_{\boldsymbol{g}}$ by the $-(J_{\boldsymbol{g}}(\boldsymbol{x}_k))^+\boldsymbol{g}(\boldsymbol{x}_k)$ term. Therefore, by solving Eq. (18) iteratively, we expect that the approximations $\boldsymbol{x}_k$ moves toward descending direction of f along with tracing the feasible set $V_{\boldsymbol{g}}$.

Summarizing the above, we obtain an algorithm as follows.

ALGORITHM 2. (A MODIFIED NEWTON METHOD [18])

Step 1 **[Finding a search direction]** For $\boldsymbol{x}_k$, calculate $\boldsymbol{d}_k$ by solving the linear system (18). If $\|\boldsymbol{d}_k\|$ is sufficiently small, go to Step 2. Otherwise, calculate the step width α_k by an appropriate line search method (we omit its detail here), let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k$, then go to Step 1.

Step 2 **[Checking the first-order necessary conditions]** If $\boldsymbol{x}_k$ satisfies Theorem 1, then return $\boldsymbol{x}_k$.

4. THE ALGORITHM FOR APPROXIMATE GCD

In applying the gradient-projection method or a modified Newton method to the approximate GCD problem, we discuss issues in the construction of the algorithm in detail, such as

- Representation of the Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x})$ and certifying that $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank (Section 4.1),
- Setting the initial values (Section 4.2),
- Regarding the minimization problem as the minimum distance problem (Section 4.3),
- Calculating the actual GCD and correcting the coefficients of $\tilde{F}$ and $\tilde{G}$ (Section 4.4),

as follows. After presenting the algorithm, we end this section with examples.

4.1 Representation and the rank of the Jacobian Matrix

By the definition of our problem (12), the Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x})$ is represented (with the original notation of variables (10) for $\boldsymbol{x}$) as shown in Fig. 1, which can be easily constructed in every iteration in Algorithms 1 and 2.

In executing Algorithm 1 or 2, we need to keep that $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank: otherwise, we cannot correctly calculate $(J_{\boldsymbol{g}}(\boldsymbol{x}))^+$ (in Algorithm 1) or the matrix in (18) becomes singular (in Algorithm 2) thus we are unable to decide proper search direction. For this requirement, we have the following observations.

PROPOSITION 1. *Let $\boldsymbol{x}^* \in V_{\boldsymbol{g}}$ be any feasible point satisfying Eq.* (12)*. Then, if the corresponding polynomials do not have a GCD whose degree exceeds d, then $J_{\boldsymbol{g}}(\boldsymbol{x}^*)$ has full rank.*

PROOF. Let $\boldsymbol{x}^* = (\tilde{f}_m, \ldots, \tilde{f}_0,\ \tilde{g}_n, \ldots, \tilde{g}_0,\ a_{n-d} \ldots, a_0,\ b_{m-d}, \ldots, b_0)$ with its polynomial representation expressed as in (5) (note that this assumption permits the polynomials $\tilde{F}(x)$ and $\tilde{G}(x)$ to be relatively prime in general). To verify our claim, we show that we have $\operatorname{rank}(J_{\boldsymbol{g}}(\boldsymbol{x}^*)) = m+n-d+2$ as in (13). Let us express $J_{\boldsymbol{g}}(\boldsymbol{x}^*) = \left(J_{\mathrm{L}} \mid J_{\mathrm{R}}\right)$, where J_{L} is a column block consisting of left $m+n+2$ columns of a_js and b_js and J_{R} is the column block consisting of the rest of columns. Then, we have the following lemma.

LEMMA 1. *We have* $\operatorname{rank}(J_{\mathrm{L}}) = m+n-d+1$.

PROOF. Let us express $J_{\mathrm{L}} = \left(J_{\mathrm{LL}} \mid J_{\mathrm{LR}}\right)$, where J_{LL} is the column block consisting of left $m+1$ columns of a_js and J_{LR} is the column block consisting of right $n+1$ columns of b_js, and let $\bar{J}_{\mathrm{L}}$ be a submatrix of J_{L} by taking the right $m-d$ columns of J_{LL} and the right $n-d$ columns of J_{LR}. Then, we see that the bottom $m+n-2d$ rows of $\bar{J}_{\mathrm{L}}$ is equal to $N(A,B)$, the Sylvester matrix of $A(x)$ and $B(x)$. By the assumption, polynomials $A(x)$ and $B(x)$ are relatively prime, and there exist no nonzero elements in $\bar{J}_{\mathrm{L}}$ except for the bottom $m+n-2d$ rows, we have $\operatorname{rank}(\bar{J}_{\mathrm{L}}) = m+n-2d$.

By the above structure of $\bar{J}_{\mathrm{L}}$ and the lower triangular structure of J_{LL} and J_{LR}, we can take the left $d+1$ columns of J_{LL} or J_{LR} satisfying linear independence along with the $m+n-2d$ columns in $\bar{J}_{\mathrm{L}}$. Therefore, these $m+n-d+1$ columns generate a $(m+n-d+1)$-dimensional subspace in $\mathbf{R}^{m+n-d+2}$ satisfying

$$\{{}^t(x_1, \ldots, x_{m+n-d+2}) \in \mathbf{R}^{m+n-d+2} \mid x_1 = 0\}, \tag{20}$$

and we see that none of the columns in J_{L} have nonzero element in the top coordinate. This proves the lemma. □

PROOF OF PROPOSITION 1 (CONTINUED). By the assumptions, we have at least one column vector in J_{R} with nonzero coordinate on the top row. By adding such a column vector to the basis of the subspace (20) that are generated as in Lemma 1, we have a basis of $\mathbf{R}^{m+n-d+2}$. This implies $\operatorname{rank}(J_{\boldsymbol{g}}(\boldsymbol{x})) = m+n-d+2$, which proves the proposition. □

Proposition 1 says that, so long as the search direction in the minimization problem satisfies that corresponding polynomials have a GCD of degree not exceeding d, then $J_{\boldsymbol{g}}(\boldsymbol{x})$ has full rank, thus we can safely calculate the next search direction for approximate GCD.

4.2 Setting the Initial Values

At the beginning of iterations, we give the initial value $\boldsymbol{x}_0$ by using the singular value decomposition (SVD) [5] of the $(d-1)$-th subresultant matrix $N_{d-1}(F,G) : \mathbf{R}^{m+n-2d} \to \mathbf{R}^{m+n-d}$ in (3). Let $N_{d-1}(F,G) = U\,\Sigma\,{}^tV$ be the SVD of $N_{d-1}(F,G)$, where

$$\begin{gathered} U = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{m+n-2d}), \quad V = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{m+n-2d}), \\ \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_{m+n-2d}), \end{gathered} \tag{21}$$

with $\boldsymbol{u}_j \in \mathbf{R}^{m+n-d}$, $\boldsymbol{v}_j \in \mathbf{R}^{m+n-2d}$, and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_{m+n-2d})$ denotes the diagonal matrix whose the j-th diagonal element is σ_j. Note that U and V are orthogonal matrices. Then, by a property of the SVD [5, Theorem 3.3], the smallest singular value σ_{m+n-2d} gives the minimum distance of the image of the unit sphere $\mathrm{S}^{m+n-2d-1} = \{\boldsymbol{x} \in \mathbf{R}^{m+n-2d} \mid \|\boldsymbol{x}\|_2 = 1\}$ by N_{d-1}, represented as $N_{d-1} \cdot \mathrm{S}^{m+n-d-1} = \{N_{d-1}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbf{R}^{m+n-2d}, \|\boldsymbol{x}\|_2 = 1\}$, from the origin, along with $\sigma_{m+n-2d}\boldsymbol{u}_{m+n-2d}$ as its coordinates. By (21), we have $N_{d-1} \cdot \boldsymbol{v}_{m+n-2d} = \sigma_{m+n-2d}\boldsymbol{u}_{m+n-2d}$, thus $\boldsymbol{v}_{m+n-2d}$ represents the coefficients of $A(x)$ and $B(x)$: let $\boldsymbol{v}_{m+n-2d} = {}^t(\bar{a}_{n-d}, \ldots, \bar{a}_0, \bar{b}_{n-d}, \ldots, \bar{b}_0)$, $\bar{A}(x) = \bar{a}_{n-d}x^{n-d} + \cdots + \bar{a}_0x^0$ and $\bar{B}(x) = \bar{b}_{m-d}x^{m-d} + \cdots + \bar{b}_0x^0$. Then, $\bar{A}(x)$ and $\bar{B}(x)$ give the least norm of $AF+BG$ satisfying $\|A\|_2^2 + \|B\|_2^2 = 1$ by putting $A(x) = \bar{A}(x)$ and $B(x) = \bar{B}(x)$.

Therefore, we admit the coefficients of F, G, $\bar{A}$ and $\bar{B}$ as the initial values of the iterations as

$$\begin{aligned} \boldsymbol{x}_0 = (&f_m, \ldots, f_0, g_n, \ldots, g_0, \\ &\bar{a}_{n-d}, \ldots, \bar{a}_0, \bar{b}_{m-d}, \ldots, \bar{b}_0). \end{aligned} \tag{22}$$

4.3 Regarding the Minimization Problem as the Minimum Distance (Least Squares) Problem

Since we have the object function f as in (11), we have

$$\begin{aligned} \nabla f(\boldsymbol{x}) = 2 \cdot {}^t(&x_1 - f_m, \ldots, x_{m+1} - f_0, \\ &x_{m+2} - g_n, \ldots, x_{m+n+2} - g_0, 0, \ldots, 0). \end{aligned} \tag{23}$$

However, we can regard our problem as finding a point $\boldsymbol{x} \in V_{\boldsymbol{g}}$ which has the minimum distance to the initial point $\boldsymbol{x}_0$ with respect to the $(x_1, \ldots, x_{m+n+2})$-coordinates which correspond to the coefficients in $F(x)$ and $G(x)$. Therefore,

$$J_{\boldsymbol{g}}(\boldsymbol{x}) = \begin{pmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 & 2a_{n-d} & \cdots & 2a_0 & 2b_{m-d} & \cdots & 2b_0 \\ a_{n-d} & & & b_{m-d} & & & \tilde{f}_m & & & \tilde{g}_n & & \\ \vdots & \ddots & & \vdots & \ddots & & \vdots & \ddots & & \vdots & \ddots & \\ a_0 & & a_{n-d} & b_0 & & b_{m-d} & \tilde{f}_0 & & \tilde{f}_m & \tilde{g}_0 & & \tilde{g}_n \\ & \ddots & \vdots & & \ddots & \vdots & & \ddots & \vdots & & \ddots & \vdots \\ & & a_0 & & & b_0 & & & \tilde{f}_0 & & & \tilde{g}_0 \\ \underbrace{\qquad}_{m+1} & & & \underbrace{\qquad}_{n+1} & & & \underbrace{\qquad}_{n-d+1} & & & \underbrace{\qquad}_{m-d+1} & & \end{pmatrix}.$$

Figure 1: The Jacobian matrix $J_{\boldsymbol{g}}(\boldsymbol{x})$. See Section 4.1 for details.

in the gradient projection method at $\boldsymbol{x} \in V_{\boldsymbol{g}}$, the projection of $-\nabla f(\boldsymbol{x})$ in (16) should be the projection of

$$^t(x_1 - f_m, \ldots, x_{m+1} - f_0, x_{m+2} - g_n, \ldots, x_{m+n+2} - g_0, 0, \ldots, 0), \quad (24)$$

onto the $T_{\boldsymbol{x}}$. This change is equivalent to changing the objective function as $\bar{f}(\boldsymbol{x}) = \frac{1}{2} f(\boldsymbol{x})$ then solving the minimization problem of $\bar{f}(\boldsymbol{x})$, subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$.

4.4 Calculating the Actual GCD and Correcting the Deformed Polynomials

After successful end of the iterations in Algorithms 1 or 2, we obtain the coefficients of $\tilde{F}(x)$, $\tilde{G}(x)$, $A(x)$ and $B(x)$ satisfying (4) with $A(x)$ and $B(x)$ are relatively prime. Then, we need to compute the actual GCD $H(x)$ of $\tilde{F}(x)$ and $\tilde{G}(x)$. Although H can be calculated as the quotient of $\tilde{F}$ divided by B or $\tilde{G}$ divided by A, naive polynomial division may cause numerical errors in the coefficient. Thus, we calculate the coefficients of H by the so-called least squares division [21], followed by correcting the coefficients in $\tilde{F}$ and $\tilde{G}$ by using the calculated H, as follows.

For a polynomial $P(x) \in \mathbf{R}[x]$ represented as $P(x) = p_n x^n + \cdots + p_0 x^0$, let $C_k(P)$ be a real $(n+k, k+1)$ matrix defined as

$$C_k(P) = \begin{pmatrix} p_n & & \\ \vdots & \ddots & \\ p_0 & & p_n \\ & \ddots & \vdots \\ & & p_0 \\ \underbrace{\qquad}_{k+1} & & \end{pmatrix}.$$

Then, for polynomials $\tilde{F}$, $\tilde{G}$, A and B represented as in (5) and H represented as $H(x) = h_d x^d + \cdots + h_0 x^0$, solve the equations $HB = \tilde{F}$ and $HA = \tilde{G}$ with respect to H as solving the least squares problems of linear systems

$$C_d(A)\,^t(h_d, \ldots, h_0) = {}^t(\tilde{g}_n, \ldots, \tilde{g}_0), \quad (25)$$

$$C_d(B)\,^t(h_d, \ldots, h_0) = {}^t(\tilde{f}_m, \ldots, \tilde{f}_0), \quad (26)$$

respectively. Let $H_1(x), H_2(x) \in \mathbf{R}[x]$ be the candidates for the GCD whose coefficients are calculated as the least squares solutions of (25) and (26), respectively. Then, for $i = 1, 2$, calculate the norms of the residues as $r_i = \|\tilde{F} - H_i B\|_2^2 + \|\tilde{G} - H_i A\|_2^2$, respectively, and set the GCD $H(x)$ be $H_i(x)$ giving the minimum value of r_i.

Finally, for the chosen $H(x)$, correct the coefficients of $\tilde{F}(x)$ and $\tilde{G}(x)$ as $\tilde{F}(x) = H(x) \cdot B(x)$, $\tilde{G}(x) = H(x) \cdot A(x)$, respectively.

4.5 The Algorithm

Summarizing the above, the algorithm for calculating approximate GCD becomes as follows.

Algorithm 3. (GPGCD: Approximate GCD by the Gradient-Projection Method)

- Inputs:
  - $F(x), G(x) \in \mathbf{R}[x]$ with $\deg(F) \geq \deg(G) > 0$,
  - $d \in \mathbf{N}$: the degree of approximate GCD with $d \leq \deg(G)$,
  - $\varepsilon > 0$: a threshold for terminating iteration in the gradient-projection method,
  - $u \in \mathbf{N}$: an upper bound for the number of iterations permitted in the gradient-projection method.
- Outputs: $\tilde{F}(x), \tilde{G}(x), H(x) \in \mathbf{R}[x]$ such that $\tilde{F}$ and $\tilde{G}$ are deformations of F and G, respectively, whose GCD is equal to H with $\deg(H) = d$.

Step 1 [Setting the initial values] With the discussions in Section 4.2, set the initial values $\boldsymbol{x}_0$ as in (22).

Step 2 [Iteration] With the discussions in Section 4.3, solve the minimization problem of $\bar{f}(\boldsymbol{x}) = \frac{1}{2} f(\boldsymbol{x})$, subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$, with $f(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ as in (11) and (12), respectively. Apply Algorithm 1 or 2 for the minimization: repeat iterations until the search direction $\boldsymbol{d}_k$ (in (16) in the gradient-projection method or in (19) in a modified Newton method, respectively) satisfies $\|\boldsymbol{d}_k\|_2 < \varepsilon$, or the number of iteration reaches its upper bound u.

Step 3 [Construction of $\tilde{F}$, $\tilde{G}$ and H] With the discussions in Section 4.4, construct the GCD $H(x)$ and correct the coefficients of $\tilde{F}(x)$ and $\tilde{G}(x)$. Then, return $\tilde{F}(x)$, $\tilde{G}(x)$ and $H(x)$. If Step 2 did not end with the number of iterations less than u, report it to the user.

4.6 Preserving Monicity

While Algorithm 3 permits changing the leading coefficients for calculating $\tilde{F}(x)$ and $\tilde{G}(x)$, we can also give an algorithm restricting inputs $F(x)$ and $G(x)$ and outputs $\tilde{F}(x)$ and $\tilde{G}(x)$ to be monic. However, because of limitations of space, we omit it here and refer to the full-length version of this paper [19].

4.7 Examples

Now we show examples of Algorithm 3 (more comprehensive experiments are presented in the next section).

Note that, for the minimization method we have employed a modified Newton method (Algorithm 2). Computations in Example 1 have been executed on Mathematica 6 with hardware floating-point arithmetic, while those in Example 2 and 3 have been executed on Maple 12 with `Digits=10`.

Example 1. This example is given by Karmarker and Lakshman [10], followed by Kaltofen *et al* [9]. Let $F(x), G(x) \in \mathbf{R}[x]$ be

$$F(x) = x^2 - 6x + 5 = (x-1)(x-5),$$
$$G(x) = x^2 - 6.3x + 5.72 = (x-1.1)(x-5.2),$$

and find $\tilde{F}(x), \tilde{G}(x) \in \mathbf{R}[x]$ which have the GCD of degree 1, namely $\tilde{F}(x)$ and $\tilde{G}(x)$ have one common zero.

Case 1: The leading coefficient can be perturbed. Applying Algorithm 3 to F and G, with $d = 1$ and $\varepsilon = 1.0 \times 10^{-8}$, after 7 iterations, we obtain the polynomials $\tilde{F}$ and $\tilde{G}$ as

$$\tilde{F}(x) = 0.985006x^2 - 6.00294x + 4.99942,$$
$$\tilde{G}(x) = 1.01495x^2 - 6.29707x + 5.72058,$$

with perturbations as $\|\tilde{F}-F\|_2^2+\|\tilde{G}-G\|_2^2 = 0.0004663065027$ and the common zero of $\tilde{F}(x)$ and $\tilde{G}(x)$ as $x = 5.09890419203$. In Kaltofen *et al* [9], the calculated perturbations obtained is 0.0004663 with the common zero as $x = 5.09890429$. Karmarker and Lakshman [10] only give an example without perturbations on the leading coefficients.

Case 2: The leading coefficient cannot be perturbed. Applying Algorithm 3 (preserving monicity) with the same arguments as in Case 1, after 7 iterations, we obtain the polynomials $\tilde{F}$ and $\tilde{G}$ as

$$\tilde{F}(x) = x^2 - 6.07504x + 4.98528,$$
$$\tilde{G}(x) = x^2 - 6.22218x + 5.73527,$$

with perturbations as $\|\tilde{F}-F\|_2^2+\|\tilde{G}-G\|_2^2 = 0.01213604416$ and the common zero of $\tilde{F}(x)$ and $\tilde{G}(x)$ as $x = 5.0969464650$. In Kaltofen *et al* [9], the calculated perturbations obtained is 0.01213604583 with the common zero as $x = 5.0969478$. In Karmarker and Lakshman [10], the calculated perturbations obtained is 0.01213605293 with the common zero as $x = 5.096939087$.

The next examples, originally by Sanuki and Sasaki [15], are ill-conditioned ones with the small or large leading coefficient GCD.

Example 2. (A small leading coefficient problem [15, Example 4].) Let $F(x)$ and $G(x)$ be

$$F(x) = (x^4 + x^2 + x + 1)(0.001x^2 + x + 1),$$
$$G(x) = (x^3 + x^2 + x + 1)(0.001x^2 + x + 1).$$

Applying Algorithm 3 to F and G, with $d = 2$ and $\varepsilon = 1.0 \times 10^{-8}$, after 1 iteration, we obtain the polynomials $\tilde{F}$, $\tilde{G}$ and H as $\tilde{F}(x) \simeq F(x)$, $\tilde{G}(x) \simeq G(x)$, $H(x) = 0.001x^2 + 0.9999999936x + 0.9999999936$, with $\|\tilde{F}-F\|_2^2+\|\tilde{G}-G\|_2^2 = 7.2 \times 10^{-23}$.

Example 3. (A big leading coefficient problem [15, Example 5].) Let $F(x)$ and $G(x)$ be

$$F(x) = (x^6 - 0.00001(0.8x^5 + 3x^4 - 4x^3 - 4x^2 - 5x + 1)) \times C(x),$$
$$G(x) = (x^5 + x^4 + x^3 - 0.1x^2 + 1) \cdot C(x),$$

with $C(x) = x^2 + 0.001$. Applying Algorithm 3 to F and G, with $d = 2$ and $\varepsilon = 1.0 \times 10^{-8}$, after 1 iteration, we obtain the polynomials $\tilde{F}$, $\tilde{G}$ and H as $\tilde{F}(x) \simeq F(x)$, $\tilde{G}(x) \simeq G(x)$, $H(x) = x^2 + 1.548794164 \times 10^{-16}x + 0.001$, with $\|\tilde{F}-F\|_2^2 + \|\tilde{G}-G\|_2^2 = 3.01 \times 10^{-28}$.

5. EXPERIMENTS

We have implemented our GPGCD method (Algorithm 3) on Mathematica and Maple, and carried out the following tests:

- Comparison of performance of the gradient-projection method (Algorithm 1) and a modified Newton method (Algorithm 2),
- Comparison of performance of the GPGCD method with a method based on the structured total least norm (STLN) method [8],

on randomly generated polynomials with approximate GCD.

In the tests, we have generated random polynomials with GCD then added noise, as follows. First, we have generated a pair of monic polynomials $F_0(x)$ and $G_0(x)$ of degrees m and n, respectively, with the GCD of degree d. The GCD and the prime parts of degrees $m-d$ and $n-d$ are generated as monic polynomials and with random coefficients $c \in [-10, 10]$ of floating-point numbers. For noise, we have generated a pair of polynomials $F_{\mathrm{N}}(x)$ and $G_{\mathrm{N}}(x)$ of degrees $m-1$ and $n-1$, respectively, with random coefficients as the same as for $F_0(x)$ and $G_0(x)$. Then, we have defined a pair of test polynomials $F(x)$ and $G(x)$ as

$$F(x) = F_0(x) + \frac{e_F}{\|F_{\mathrm{N}}(x)\|_2} F_{\mathrm{N}}(x),$$
$$G(x) = G_0(x) + \frac{e_G}{\|G_{\mathrm{N}}(x)\|_2} G_{\mathrm{N}}(x),$$

respectively, scaling the noise such that the 2-norm of the noise for F and G is equal to e_F and e_G, respectively. In the present test, we set $e_F = e_G = 0.1$.

The tests have been carried out on Intel Core2 Duo Mobile Processor T7400 (in Apple MacBook "Mid-2007" model) at 2.16 GHz with RAM 2GB, under MacOS X 10.5.

5.1 Comparison of the Gradient-projection Method and a Modified Newton Method

In this test, we have used an implementation on Mathematica and compared performance of the gradient-projection method (Algorithm 1) and a modified Newton method (Algorithm 2). For every example, we have generated one random test polynomial as in the above, and we have applied Algorithm 3 (preserving monicity) with $u = 100$ and $\varepsilon = 1.0 \times 10^{-8}$.

Table 1 shows the result of the test: m and n denotes the degree of a tested pair F and G, respectively, and d denotes the degree of approximate GCD; "Error" is the perturbation

$$\|\tilde{F} - F\|_2^2 + \|\tilde{G} - G\|_2^2, \tag{27}$$

Ex.	m, n	d	Error	#Iterations		Time (sec.)	
				Alg. 1	Alg. 2	Alg. 1	Alg. 2
1	10, 10	5	$7.65e{-3}$	3	4	0.08	0.05
2	30, 30	10	$3.10e{-3}$	3	4	2.05	0.80
3	40, 40	20	$3.60e{-3}$	3	4	3.37	1.33
4	60, 60	30	$7.27e{-3}$	3	4	10.14	4.41
5	80, 80	40	$5.24e{-3}$	3	4	22.61	10.39
6	100, 100	50	$4.92e{-3}$	3	4	42.88	20.34

Table 1: Test results comparing the gradient-projection method and a modified Newton method. See Section 5.1 for details.

where "$ae-b$" denotes $a \times 10^{-b}$; "#Iterations" is the number of iterations; "Time" is computing time in seconds. The columns with "Alg. 1" and "Alg. 2" are the data for Algorithm 1 (the gradient-projection method) and Algorithm 2 (a modified Newton method), respectively. Note that, the "Error" is a single column since both algorithms give almost the same values in each examples.

We see that, in all the test cases, the number of iterations of the gradient-projection method (Algorithm 1) is equal to 3, which is smaller than that of a modified Newton method (Algorithm 2) which is equal to 4. However, an iteration in Algorithm 1 includes solving a linear system at least twice: once in the *projection step* (Step 2) and at least once in the *restoration step* (Step 3); whereas an iteration in Algorithm 2 includes that only once. Thus, total number of solving a linear system in Algorithm 2 is about a half of that in Algorithm 1. Furthermore, computing time shows that, although both implementations are rather inefficient because of elementary implementations, a modified Newton method runs approximately twice as fast as the gradient projection method. Therefore, we adopt Algorithm 2 as the method of minimization in the GPGCD method (Algorithm 3).

5.2 Tests on Large Sets of Randomly-generated Polynomials

In this test, we have used our implementation on Maple and compared its performance with a method based on the structured total least norm (STLN) method [8], using their implementation (see Acknowledgments). In our implementation of Algorithm 3, we have chosen a modified Newton method (Algorithm 2) for minimization, while, in the STLN-based method, we have used the procedure `R_con_mulpoly` which calculates the approximate GCD of several polynomials in $\mathbf{R}[x]$. The tests have been carried out on Maple 12 with `Digits=15` executing hardware floating-point arithmetic.

For every example, we have generated 100 random test polynomials as in the above. In executing Algorithm 3, we set $u = 200$ and $\varepsilon = 1.0 \times 10^{-8}$; in `R_con_mulpoly`, we set the tolerance $e = 1.0 \times 10^{-8}$.

Table 2 shows the results of the test: m and n denotes the degree of a pair F and G, respectively, and d denotes the degree of approximate GCD. The columns with "STLN" are the data for the STLN-based method, while "GPGCD" are the data for the GPGCD method (Algorithm 3). "#Fail" is the number of "failed" cases such as: in the STLN-based method, the number of iterations exceeds 50 times (which is the built-in threshold in the program), while, in the GPGCD method, the perturbation (27) exceeds 1 (note that, in the GPGCD method, all the iterations have converged within far less than 200 times). All the other data are the average over results for the "not failed" cases: "Error", "#Iterations" and "Time" are the same as those in Table 1, respectively.

We see that, in the most of tests, both methods calculate approximate GCD with almost the same amount of perturbations, while GPGCD method runs much faster than STLN-based method by approximately from 10 to 30 times. On the other hand, in some cases, the GPGCD method did not give an answer with sufficiently small amount of perturbations.

Remark 1. In this experiment, we compare our implementation designed for problems of two univariate polynomials against theirs designed for multivariate multi-polynomial problems with additional linear coefficient constraints.

Kaltofen [7] has reported that they have tested their implementation for univariate polynomials [9] on an example similar to ours with degree 100 and GCD degree 50, and it took (on a ThinkPad of 1.8 GHz with RAM 1GB) 2 iterations and 9 seconds. This result will give the reader some idea on efficiency of our method. □

6. CONCLUDING REMARKS

We have proposed an iterative method, based on a modified Newton method which is a generalization of the gradient-projection method, for calculating approximate GCD of univariate polynomials.

Our experiments have shown that our algorithm calculates approximate GCD with perturbations as small as those calculated by methods based on the structured total least norm (STLN) method, while our method has shown significantly better performance over the STLN-based methods in its speed, by approximately up to 30 times, which seems to be sufficiently practical for inputs of low or moderate degrees. Furthermore, by other examples, we have shown that our algorithm can properly handle some ill-conditioned problems such as those with GCD containing small or large leading coefficient.

However, our experiments have also shown that there are some cases in which the GPGCD method did not give an answer with sufficiently small amount of perturbations. Factors leading to such phenomena is under investigation.

For the future research, the followings will be of interest: time complexity of our method depends on the minimization, or solving a linear system in each iteration. Thus, analyzing the structure of matrices might improve the efficiency. Furthermore, generalizing our method to polynomials with the complex coefficients and/or several polynomials will be among our next problems.

Ex.	m, n	d	#Fail		Error		#Iterations		Time (sec.)	
			STLN	GPGCD	STLN	GPGCD	STLN	GPGCD	STLN	GPGCD
1	10, 10	5	0	2	$3.63e{-3}$	$3.67e{-3}$	4.65	4.99	0.43	0.05
2	20, 20	10	0	4	$4.37e{-3}$	$4.28e{-3}$	4.97	4.78	1.33	0.09
3	30, 30	15	2	1	$4.65e{-3}$	$4.64e{-3}$	4.34	5.28	2.54	0.16
4	40, 40	20	0	0	$4.73e{-3}$	$4.73e{-3}$	4.28	4.54	4.41	0.23
5	50, 50	25	0	0	$4.79e{-3}$	$4.79e{-3}$	4.32	4.51	6.96	0.33
6	60, 60	30	0	0	$4.82e{-3}$	$4.54e{-3}$	4.27	4.45	10.44	0.45
7	70, 70	35	1	1	$4.71e{-3}$	$4.71e{-3}$	3.97	4.27	13.28	0.58
8	80, 80	40	0	2	$4.77e{-3}$	$4.77e{-3}$	4.06	4.34	17.96	0.78
9	90, 90	45	0	1	$5.10e{-3}$	$4.94e{-3}$	4.18	4.29	23.61	0.97
10	100, 100	50	1	0	$4.82e{-3}$	$4.81e{-3}$	4.11	4.56	29.87	1.28

Table 2: Test results for large sets of polynomials with approximate GCD. See Section 5.2 for details.

Acknowledgments

We thank Professor Erich Kaltofen for making their implementation for computing approximate GCD available on the Internet and providing experimental results. We also thank the anonymous reviewers for their helpful comments.

This research was supported in part by the Ministry of Education, Culture, Sports, Science and Technology of Japan, under Grant-in-Aid for Scientific Research (KAKENHI) 19700004.

7. REFERENCES

[1] B. Beckermann and G. Labahn. A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. *J. Symbolic Comput.*, 26(6):691–714, 1998.

[2] P. Chin, R. M. Corless, and G. F. Corliss. Optimization strategies for the approximate GCD problem. In *Proc. ISSAC'98*, pages 228–235. ACM, 1998.

[3] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *Proc. ISSAC'95*, pages 195–207. ACM, 1995.

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

[5] J. W. Demmel. *Applied numerical linear algebra*. SIAM, 1997.

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

[7] E. Kaltofen. Private communication. 2009.

[8] E. Kaltofen, Z. Yang, and L. Zhi. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In *Proc. ISSAC'06*, pages 169–176. ACM, 2006.

[9] E. Kaltofen, Z. Yang, and L. Zhi. Structured low rank approximation of a Sylvester matrix. In D. Wang and L. Zhi, editors, *Symbolic-Numeric Computation*, pages 69–83. Birkhäuser, 2007.

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

[11] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, New York, second edition, 2006.

[12] N. Ohsako, H. Sugiura, and T. Torii. A stable extended algorithm for generating polynomial remainder sequence (in Japanese). *Trans. Japan Soc. Indus. Appl. Math*, 7(3):227–255, 1997.

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

[14] J. B. Rosen. The gradient projection method for nonlinear programming. II. Nonlinear constraints. *J. Soc. Indust. Appl. Math.*, 9:514–532, 1961.

[15] M. Sanuki and T. Sasaki. Computing approximate GCDs in ill-conditioned cases. In *Proc. SNC'07*, pages 170–179. ACM, 2007.

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

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

[18] K. Tanabe. A geometric method in nonlinear programming. *J. Optim. Theory Appl.*, 30(2):181–210, 1980.

[19] A. Terui. An iterative method for computing approximate GCD of univariate polynomials. preprint. 9 pages. 2009.

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

[21] Z. Zeng. The approximate GCD of inexact polynomials, Part I: a univariate algorithm (extended abstract). preprint. 8 pages.

[22] L. Zhi. Displacement structure in computing approximate GCD of univariate polynomials. In *Proc. the Sixth Asian Symposium on Computer Mathematics (ASCM 2003)*, pages 288–298. World Scientific, 2003.