Approximate square-free part and decomposition

Nagasaka, Kosaku

Approximate Square-free Part and Decomposition

Kosaku Nagasaka

Kobe University, Japan

Abstract

Square-free decomposition is one of fundamental computations for polynomials. However, any conventional algorithm may not work for polynomials with a priori errors on their coefficients. There are mainly two approaches to overcome this empirical situation: approximate polynomial GCD (greatest common divisor) and the nearest singular polynomial. In this paper, we show that these known approaches are not enough for detecting the nearest square-free part (which has no multiple roots) within the given upper bound of perturbations (a priori errors), and we propose a new definition and a new method to detect a square-free part and its decomposition numerically by following a recent framework of approximate polynomial GCD.

Keywords: Symbolic-Numeric Algorithm for Polynomials, Approximate Polynomial GCD

1. Introduction

Let K be a field of characteristic zero (e.g. the real number field $\mathbb{R}$) and $K[x]$ be the polynomial ring over K. The square-free decomposition of polynomial $f(x) \in K[x]$ is defined as

$$f(x) = f_1(x)^1 f_2(x)^2 \cdots f_r(x)^r$$

where each $f_i(x)$ is square-free (no multiple roots (i.e. $\gcd(f_i, f_i') = 1$)) and are pairwise coprime. This decomposition is one of fundamental computations for polynomials, and in general calculated by computing polynomial GCDs successively (for example, see Yun (1976)). Similarly the square-free part of $f(x)$ above is defined as

$$f(x)/\gcd(f, f') = f_1(x)f_2(x)\cdots f_r(x)$$

where $f'(x)$ is the first derivative of $f(x)$. Computing the square-free part is useful for factoring $f(x)$, calculating numerical roots of $f(x)$ and so on, as one of preprocessings.

However, any conventional algorithm may not work for polynomials with a priori errors on their coefficients since in general polynomials become square-free easily by perturbing their coefficients (i.e. any multiple root becomes a cluster of roots). In other words, we need to perturb the input polynomial being not square-free within the estimated upper bound of a priori errors. To overcome this empirical situation, there are mainly two approaches: approximate polynomial GCD and the nearest singular polynomial.

Email address: nagasaka@main.h.kobe-u.ac.jp (Kosaku Nagasaka)
URL: https://wwwmain.h.kobe-u.ac.jp/~nagasaka/ (Kosaku Nagasaka)

As for the former approach, though there are several different definitions (for example, see Boito (2011)), in this paper the approximate polynomial GCD of $f(x)$ and $g(x)$ of tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$ is defined as $d(x) \in K[x]$ of maximum degree, satisfying

$$f(x) + \Delta_f(x) = f_1(x)d(x),\ g(x) + \Delta_g(x) = g_1(x)d(x)$$

for some polynomials $\Delta_f(x), \Delta_g(x), f_1(x), g_1(x) \in K[x]$ such that

$$\deg(\Delta_f) \leq \deg(f),\ \deg(\Delta_g) \leq \deg(g),\ \|\Delta_f\| \leq \varepsilon\ \|f\|,\ \|\Delta_g\| \leq \varepsilon\ \|g\|$$

where $\|\cdot\|$ denotes a polynomial norm (e.g. the Euclidean norm of its coefficient vector). There are many studies on approximate polynomial GCD (see Boito (2011) for example). However, there are only a few known studies for approximate square-free decomposition.

Dunaway (1974) uses numerical computations (floating-point arithmetics) to calculate the square-free decomposition for computing numerical roots. The concept of approximate square-free decomposition is first appeared in Sasaki and Noda (1989) though its definition has some ambiguities on the upper bound of allowing perturbations (a priori errors). Their method is based on approximate polynomial GCD by the Euclidean algorithm, and nowadays it is not robust against to other approximate polynomial GCD algorithms based on matrix factorizations. In Diaz-Toca and Gonzalez-Vega (2006), we can find a method based on the Bezout matrix. However, the method does not control the perturbations of the input polynomial directly. It computes approximate polynomial GCDs successively and independently, and only tests if their results meet the upper bound of allowing perturbations of approximate square-free decomposition. Hence in general it may not compute a decomposition with a nearly minimum perturbation.

As for the latter approach, the nearest singular polynomial is a polynomial that has a multiple root and is nearest to the input polynomial by some measure (e.g. the Euclidean norm). This can be considered as one of non square-free polynomials of the input polynomial. In Karmarkar and Lakshman (1996), their method seeks a nearest singular polynomial that has a double root by solving a parametric Lagrange multiplier problem of a minimization problem of the quadratic form. By following this method, there are relevant studies in Lihong and Wenda (1998); Zhi et al. (2004); Li and Zhi (2013). They give an explicit recursive formula to attain the minimum value of the quadratic form. Especially in Li and Zhi (2013), their method can find a nearest singular polynomial with the specified multiplicity structures of roots (e.g. one triple root and two double roots). Therefore, if we know the multiplicity structures of the input polynomial that may have been hidden by a priori errors, we can compute an approximate square-free part with the smallest perturbation. However, as shown in the next section, this is not enough to get a square-free part in general. Moreover, in Kaltofen et al. (2006) we can find a method to compute a nearest singular polynomial that has a root of multiplicity k by approximate polynomial GCD. This method has some similarity to our method but they do not give a method for approximate square-free part and its decomposition.

The main contribution of this paper is formed by two parts: 1) we propose a new explicit definition for approximate square-free part and its decomposition, and 2) we propose a new method to find an approximate square-free part and its decomposition within the given upper bound of perturbations, by following a recent framework of approximate polynomial GCD. First of all, in section 2, we show that the known approaches are not enough for detecting the nearest square-free part and its decomposition within the given upper bound of perturbations. Our methods is given in section 3. In section 4, we show some numerical examples of our method and comparisons against known methods, and give some remarks in section 5.

2. Approximate square-free part and its decomposition, and known methods

First of all, we give an explicit definition of approximate square-free part and its decomposition, and note that this kind of formulation was not appeared in the literatures.

Definition 1 (approximate square-free part and its decomposition)**.**
For the given polynomial $f(x) \in K[x]$, we call the polynomial $d(x) \in K[x]$ "approximate square-free part" of tolerance ε, that has minimum degree and satisfies

$$f(x) + \Delta_f(x) = d(x) \times \gcd\left(f(x) + \Delta_f(x),\ \frac{\partial}{\partial x}(f(x) + \Delta_f(x))\right) \tag{2.1}$$

for some polynomial $\Delta_f(x) \in K[x]$ with minimum $\varepsilon \in \mathbb{R}_{\geq 0}$ such that $\deg(\Delta_f) \leq \deg(f)$ *and* $\|\Delta_f\| \leq \varepsilon \|f\|$ *where $\|\cdot\|$ denotes a polynomial norm (e.g. the Euclidean norm). Moreover, we call the square-free decomposition of the resulting $f(x) + \Delta_f(x)$ "approximate square-free decomposition" of tolerance ε.*

One may think that this definition is very similar to approximate polynomial GCD. This is true and in fact we can compute an approximate square-free part of $f(x)$ by computing an approximate polynomial GCD of $f(x)$ and $f'(x)$ as in the next section, though this formulation with much consideration has not been appeared in the literatures. Again, the "minimum degree" condition is most essential part of this definition, and this means that the number of distinct roots is to be minimized. In other words, in our definition, the optimization target is explicitly given and to be minimized within the specified tolerance, while any known method just finds some square-free factors independently that may not satisfy the specified tolerance.

As similar to the problem of finding approximate polynomial GCD, we have the following two fundamental problems for approximate square-free part and its decomposition. In this paper, we solve the problem 2 by solving the corresponding problems 1 successively for possible degrees k, and we use the Euclidean norm (2-norm).

Problem 1 (degree given problem)**.**
For the given polynomial $f(x) \in K[x]$ and degree $k \in \mathbb{N}$, find an approximate square-free part of degree k and its decomposition of $f(x)$.

Problem 2 (tolerance given problem)**.**
For the given polynomial $f(x) \in K[x]$ and tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$, find an approximate square-free part and its decomposition of $f(x)$, whose tolerance is not larger than ε.

2.1. Finding approximate square-free part by approximate polynomial GCD

There are known studies (Sasaki and Noda, 1989; Diaz-Toca and Gonzalez-Vega, 2006; Kaltofen et al., 2006) for approximate square-free decomposition. However, their methods can not guarantee to attain the minimum degree of square-free part, since their methods do not control the resulting tolerance (size of perturbations) directly. They are just trying to compute a numerical or approximate variations of square-free decomposition by some conventional algorithm with approximate polynomial GCD and divisions (getting quotients) instead of exact ones.

For example, the methods in Diaz-Toca and Gonzalez-Vega (2006); Kaltofen et al. (2006) are based on the following well-known fact.

structure	$(3,1,0,0,0)$	$(1,2,0,0,0)$	$(2,0,1,0,0)$	$(0,1,1,0,0)$	$(1,0,0,1,0)$
$\|\|\Delta_f\|\|_2$	0.182707e-3	0.402327e-3	0.593679e-3	0.398911e-2	0.390635e-0
ε	0.408648e-4	0.899856e-4	0.132784e-3	0.892217e-3	0.873704e-1
$\deg(d(x))$	4	3	3	2	2

Table 1: multiplicity structures, their tolerances and degrees of square-free parts

Lemma 1. *The following two conditions are equivalent for $f(x) \in K[x]$ with $2 \le k \le \deg(f)$.*

- *There exists a polynomial $g(x) \in K[x]$ of positive degree and $g(x)^k$ divides $f(x)$.*
- $\gcd(f, f^{(1)}, \ldots, f^{(k-1)})$ *has a positive degree where $f^{(i)}$ denotes the i-th derivative.*

By this lemma, they can compute the following $g_k(x)$ for each $k = 2, \ldots, \deg(f)$ independently.

$$f(x) + \tilde{\Delta}_k(x) = g_k(x)^k \tilde{g}_k(x), \quad \|\tilde{\Delta}_k(x)\| \le \varepsilon_k \|f(x)\|$$

for some $\tilde{g}_k(x) \in K[x]$ and $\varepsilon_k \in \mathbb{R}_{\ge 0}$. However, in general, all the $\tilde{\Delta}_k(x)$ are different and heavily depending on the order of k (e.g. descending, ascending and so on). Therefore they can only minimize the tolerance for each k (i.e. $g_k(x)$) and not for the whole decomposition (i.e. $f_1(x) \cdots f_r(x)$ and $f_1(x) f_2(x)^2 \cdots f_r(x)^r$).

Their theoretical difficulty is caused by the order of computations. They compute all the possible square-free factors successively and independently at first, and they have to compute the whole decomposition based on the computed independent factors. To control the tolerance for $f(x)$ or the whole decomposition, we must compute the whole square-free part or decomposition at first, and compute each square-free factor, as in the next section.

2.2. Finding approximate square-free part by nearest singular polynomial

The problem of nearest singular polynomial is very similar to the problem 1 but with the specified multiplicity structure of roots instead of the degree of square-free part. To see their similarity, let $f(x)$ be the following square-free polynomial of Example 2 in Li and Zhi (2013).

$$x^5 - 3.0x^4 + 2.998997x^3 - 0.997991998x^2 - 0.001007004x + 0.000402002.$$

The possible multiplicity structures of roots of $f(x)$ are $(5,0,0,0,0)$, $(3,1,0,0,0)$, $(1,2,0,0,0)$, $(2,0,1,0,0)$, $(0,1,1,0,0)$, $(1,0,0,1,0)$ and $(0,0,0,0,1)$ where $(e_1, \ldots, e_5)$ denotes that the number of roots of multiplicity k is e_k. Please note that $\sum e_k$ is the number of distinct roots and the degree of approximate square-free part. We cite the results for some of these structures from Li and Zhi (2013), and the table 1 shows them that are modified for our notations and definitions where $(fraction)$e−$(integer)$ denotes $(fraction) \times 10^{-(integer)}$. For example, this result shows that the nearest singular polynomial of multiplicity structure $(0,1,1,0,0)$ is the approximate square-free part of tolerance 0.1e−1, and $(1,2,0,0,0)$ is corresponding to 0.1e−3.

As shown in this example, computing an approximate square-free part (in the both problems 1 and 2) can be done by computing all the nearest singular polynomials for the possible multiplicity structures and picking out the polynomial that meets our minimization condition. The problem here is that the number of possible multiplicity structures may be large. For example, the possible numbers for polynomials of degrees 5, 15 and 25 are 7, 176 and 1958, respectively. In fact, the number of possible multiplicity structures of roots of $f(x)$ is the partition function $p(n)$ where n denotes the degree of $f(x)$. Since a lower bound of $p(n)$ is $e^{2\sqrt{n}}/14$ (Maróti, 2003), this means that any approximate square-free part algorithm based on the nearest singular polynomial has the sub-exponential time complexity at least. Therefore, any other approach is preferable.

3. Our method for approximate square-free part and its decomposition

First we consider the problem 1 that finds an approximate square-free part of degree k, of the given polynomial $f(x)$. This can be considered as finding an approximate polynomial GCD of degree $\deg(f) - k$, of $f(x)$ and $f'(x)$. However, $f(x)$ and $f'(x)$ are not independent hence most of algorithms for approximate polynomial GCD of independent $f(x)$ and $g(x)$ can not be used since they usually perturb the given two polynomials $f(x)$ and $g(x)$ independently. The one significant exception is the algorithm given by Kaltofen et al. (2006). Their algorithm can work with linearly constraints on the coefficients of $f(x)$ and $g(x)$, and is based on STLN (structured total least norm) by solving a minimization problem with a large penalty value. Moreover, other possible options are 1) the approach (Newton's method for a constrained optimization problem) by Giesbrecht et al. (2017), 2) modifying the method (Gauss-Newton method for unconstrained optimization problem) by Zeng (2011) to consider the dependency relation, and 3) solving a SLRA (structured low rank approximation) problem directly by a SLRA solver (Schost and Spaenlehauer, 2016). In this paper, we use these algorithms and compare their efficiency. We note that there are other approximate GCD algorithms for independent $f(x)$ and $g(x)$, however, according to our experiments (Nagasaka, 2017, 2020), the method by Zeng (2011) is the best among others.

3.1. Approximate square-free part with the given degree

Let $f(x) \in K[x]$ be the given polynomial with $n = \deg(f)$.We denote the (dense and ascending) coefficient vector of a polynomial $p(x) \in K[x]$ with $\deg(p) = k - 1$, by $\vec{p} \in K^k$, and by $C_k(f)$ we denote the convolution matrix of k-th order of $f(x)$ such that $C_k(f)\vec{p} = \vec{g}$ and $g(x) = f(x)p(x)$. Our formulation is based on the following well-known property of the subresultant mapping.

Definition 2 (subresultant mapping and matrix)**.**
For $f(x), g(x) \in K[x]$ with $\deg(f) = n$ and $\deg(g) = m$, and $r \in \mathbb{Z}_{\geq 0}$ satisfying $0 \leq r < \min\{n, m\}$, the subresultant mapping of order r is defined as

$$\phi_r(f, g) : \begin{array}{ccc} \mathcal{P}_{m-r} \times \mathcal{P}_{n-r} & \to & \mathcal{P}_{n+m-r} \\ (s(x), t(x)) & \mapsto & s(x)f(x) + t(x)g(x) \end{array}$$

where $\mathcal{P}_k$ denotes the set of polynomials of degree less than k. Moreover, the representation matrix of $\phi_r(f, g)$ is given as $S_r(f, g) = (C_{m-r}(f) \mid C_{n-r}(g)) \in K^{(n+m-r)\times(m+n-2r)}$, and we call it "subresultant matrix" of order r, of $f(x)$ and $g(x)$.

Lemma 2. *For the largest integer r such that $\phi_r(f, g)$ is not injective (i.e. $S_r(f, g)$ is not column full rank), let $(s(x), t(x))$ be any non-zero polynomial pair in the kernel of $\phi_r(f, g)$. Then, $f(x)/t(x)$ and $-g(x)/s(x)$ are the polynomial GCD of $f(x)$ and $g(x)$, and their degree is $r + 1$.*

By this lemma, computing an approximate square-free part of degree k of $f(x)$ is reduced to finding a polynomial $\Delta_f(x)$ such that $\mathrm{S}_{n-k-1}\left(f(x) + \Delta_f(x), \dfrac{\partial}{\partial x}(f(x) + \Delta_f(x))\right)$ is not column full rank. Hereafter we denote this subresultant matrix by $\mathrm{S}'_{n-k-1}(f + \Delta_f)$ for convenience sake. Then, the problem becomes the following constrained optimization.

$$\min\nolimits_{\Delta_f} \ \|\Delta_f\|_2 \ \text{ subject to } \ \mathrm{S}'_{n-k-1}(f + \Delta_f)\ \vec{w} = \vec{0} \ \text{ for some non-zero vector } \ \vec{w}. \qquad (3.1)$$

Please note that the last $k + 1$ elements of $\vec{w}$ is corresponding to $\vec{d}$, the coefficient vector of approximate square-free part $d(x)$.

Algorithm 1 Approximate square-free part with the given degree, based on the Newton's method

```
Require: $f(x) \in K[x]$ and $k \in \mathbb{N}$ with $n = \deg(f)$
Ensure: $f(x) + \Delta_f(x)$, $d(x) \in K[x]$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
        such that $\deg(d) = k$ and $d(x)$ is an approximate square-free part of $f(x)$ of tolerance $\varepsilon$
1: initialize the Newton's method;
   - compute an initial guess for $\vec{x}$ by the singular value decomposition of $\mathrm{S}'_{n-k-1}(f)$;
   - compute an initial guess for $\vec{\lambda}$ by solving a linear least squares problem;
2: repeat
3:     update the current values $\vec{x}$ and $\vec{\lambda}$ by the Newton iteration in (3.2);
4: until the norm of correction terms is sufficiently small or divergence is detected
5: construct $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$ from the resulting $\vec{x}$ of the Newton's method;
6: return $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
```

3.1.1. Algorithm based on Newton's method

To get $\Delta_f(x)$ by the Newton's method, we follow Giesbrecht et al. (2017) where a new method to compute the nearest rank-deficient matrix polynomial is proposed. Their algorithm embeds the given matrix polynomial into a structured matrix over $\mathbb{R}$, and tries to find the nearest rank-deficient structured matrix by the Newton's method. We can adapt it to our purpose that we minimize $\Psi = \| \vec{\Delta_f} \|_2^2 + \| \vec{w} \|_2^2 - 1$ subject to $\mathrm{S}'_{n-k-1}(f + \Delta_f)\ \vec{w} = \vec{0}$ and $\vec{w}^T \vec{w} = 1$ w.r.t. $\vec{x} = \left(\vec{\Delta_f}^T,\ \vec{w}^T \right)^T$ where $\cdot^T$ denotes the matrix transpose. The actual Newton iteration is as follows.

$$\begin{pmatrix} \vec{x}_{i+1} \\ \vec{\lambda}_{i+1} \end{pmatrix} = \begin{pmatrix} \vec{x}_i + \vec{\Delta_x} \\ \vec{\lambda}_i + \vec{\Delta_\lambda} \end{pmatrix}, \quad \nabla^2 L_i \begin{pmatrix} \vec{\Delta_x} \\ \vec{\Delta_\lambda} \end{pmatrix} = -\nabla L_i \tag{3.2}$$

where $L = \Psi + \ell$ with the Lagrangian multipliers $\ell = \left((\mathrm{S}'_{n-k-1}(f + \Delta_f)\vec{w})^T,\ \vec{w}^T \vec{w} - 1 \right)^T \vec{\lambda}$ and $\vec{\lambda} = (\lambda_0\ \ldots\ \lambda_{n+k})^T$. We note that $\nabla^2_{\vec{x}\vec{x}} \Psi$ is a scalar matrix (the diagonal element is 2) hence $\nabla^2_{\vec{x}\vec{x}} L_i$ can be always positive definite by re-scaling $\Psi(\vec{x})$ without changing the optimal value $\vec{x}$, and the Newton's method described here has the following convergent property as same as the method in Giesbrecht et al. (2017) since the objective function is quadratic and the subresultant matrix is a block Toeplitz matrix.

Lemma 3. *The Newton's method above converges quadratically with a suitable initial guess, if there is no approximate square-free part of degree less than k.*

The overall algorithm is described as Algorithm 1. Moreover, in our preliminary implementation, computing an initial guess for $\vec{x}$ is done by the lift-and-project method (Chu et al., 2003) using the singular value decomposition of $\mathrm{S}'_{n-k-1}(f)$, and for $\vec{\lambda}$ by solving a linear least squares problem $\min_{\vec{\lambda}} \|\nabla L\|_2$ for the computed initial value of $\vec{x}$.

Remark 1. *If the condition of Lemma 3 is not satisfied, $S'_{n-k-2}(f + \Delta_f)$ may be also rank deficient at a local optimum where $S'_{n-k-1}(f + \Delta_f)$ is rank deficient. In this case, the resulting $d(x)$ by Algorithm 1 includes a root that is not a root of $f(x) + \Delta_f(x)$, hence $d(x)$ is not an approximate square-free part. A workaround is checking whether $S'_{n-k-2}(f + \Delta_f)$ is rank deficient at the end of the algorithm.*

Algorithm 2 Approximate square-free part with the given degree, based on STLN

```
Require: $f(x) \in K[x]$ and $k \in \mathbb{N}$ with $n = \deg(f)$
Ensure: $f(x) + \Delta_f(x)$, $d(x) \in K[x]$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
        such that $\deg(d) = k$ and $d(x)$ is an approximate square-free part of $f(x)$ of tolerance $\varepsilon$
1: initialize the iteration;
   - compute an initial guess for $\vec{\Delta_f}$ and $\vec{v}$ by the singular value decomposition of $\mathrm{S}'_{n-k-1}(f)$;
2: repeat
3:     update the current values $\vec{\Delta_f}$ and $\vec{v}$ by the least squares in (3.3);
4: until the norm of correction terms is sufficiently small or divergence is detected
                                        (check w.r.t. only $\vec{\Delta_f}$ is a possible option)
5: construct $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$ from the resulting $\vec{\Delta_f}$ and $\vec{v}$ of the iterations;
6: return $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
```

3.1.2. Algorithm based on STLN

To get $\Delta_f(x)$ by STLN, we follow Kaltofen et al. (2006). Let ℓ be an integer satisfying $1 \leq \ell \leq (2n-1-2r)$, and $A_\ell(f) \in K^{(2n-1-r)\times(2n-2r-2)}$ and $\vec{b}_\ell(f) \in K^{2n-1-r}$ be the matrix and vector, respectively satisfying

$$\mathrm{S}'_{n-k-1}(f) = (A_{\ell,1}(f) \mid \vec{b}_\ell(f) \mid A_{\ell,2}(f)),\ A_\ell(f) = (A_{\ell,1}(f) \mid A_{\ell,2}(f))$$

where $\vec{b}_\ell(f)$ is the ℓ-th column vector of $\mathrm{S}'_{n-k-1}(f)$. Then, the problem becomes the following structure-preserving total least norm problem.

$$\min_{\Delta_f} \|\Delta_f\|_2 \text{ subject to } A_\ell(f+\Delta_f)\,\vec{v} = \vec{b}_\ell(f+\Delta_f) \text{ for some vector } \vec{v}.$$

Please note that the last $k+1$ elements of $\vec{w}$ that constructed by inserting -1 in front of the ℓ-th element of $\vec{v}$ is corresponding to $\vec{d}$, the coefficient vector of approximate square-free part $d(x)$.

The actual iteration is the following least squares.

$$\begin{pmatrix} \vec{\Delta}_{f_{i+1}} \\ \vec{v}_{i+1} \end{pmatrix} = \begin{pmatrix} \vec{\Delta}_{f_i} + \vec{\Delta}_\Delta \\ \vec{v}_i + \vec{\Delta}_v \end{pmatrix},\ \min_{\vec{\Delta}_\Delta,\,\vec{\Delta}_v} \left\| \begin{pmatrix} \omega(Y_i - P) & \omega A_i \\ I & O \end{pmatrix} \begin{pmatrix} \vec{\Delta}_\Delta \\ \vec{\Delta}_v \end{pmatrix} + \begin{pmatrix} -\omega\vec{r}_i \\ \vec{\Delta}_{f_i} \end{pmatrix} \right\|_2 \tag{3.3}$$

where $A_i = A_\ell(f+\Delta_{f_i})$, $\vec{r}_i = \vec{b}_\ell(f+\Delta_{f_i}) - A_i\vec{v}_i$, Y_i is a matrix such that $Y_i(\vec{f}+\vec{\Delta}_{f_i}) = A_i\vec{v}_i$, P is a matrix such that $P(\vec{f}+\vec{\Delta}_{f_i}) = \vec{b}_\ell(f+\Delta_{f_i})$, and ω is a penalty such that $\omega \gg 1$.

The overall algorithm is described as Algorithm 2. Moreover, in our preliminary implementation, computing an initial guess for $\vec{\Delta_f}$ is done by the lift-and-project method using the singular value decomposition of $\mathrm{S}'_{n-k-1}(f)$, and for $\vec{v}$ we use the first singular vector of $\mathrm{S}'_{n-k-1}(f)$.

Remark 2. *In Kaltofen et al. (2006), they also proposed a new initialization method based on Lagrangian multipliers (other than the lift-and-project method). It is better but unfortunately there are not so significant differences according to our experiments (see the table 11).*

3.1.3. Algorithm based on Gauss-Newton method

To get $\Delta_f(x)$ by the Gauss-Newton method based on Zeng (2011) where the algorithm for computing approximate GCD of independent $f(x)$ and $g(x)$, we simply introduce a penalty term that represents the dependency on $f(x)$ and $g(x) = f'(x)$. Let $F_h(s,t)$ and $\vec{b}$ be the following matrix and vector where $s(x)$ and $t(x)$ are corresponding to that in Lemma 2, the cofactors by the GCD of $f(x)+\Delta_f(x)$ and $\dfrac{\partial}{\partial x}(f(x)+\Delta_f(x))$.

$$F_h(s,t) = \begin{pmatrix} \vec{h}^T \\ C_{n-k+1}(t) \\ C'_{n-k+1}(s) \\ \omega(C_{n-k+1}(t)_L - C'_{n-k+1}(s)) \end{pmatrix} \in K^{(3n+2)\times(n-k+1)},\ \vec{b} = \begin{pmatrix} \beta \\ \vec{f} \\ \vec{f_L} \\ \vec{0} \end{pmatrix} \in K^{3n+2}$$

where $\beta \neq 0, \in \mathbb{R}$, $h(x) \in K[x]$ with $\deg(h) = n-k$, $*_L$ denotes its last n rows, $C'_{n-k+1}(s)$ denotes the matrix such that $C'_{n-k+1}(s)\vec{p} = \overrightarrow{(\int s(x)p(x)dx)_L}$ for a polynomial $p(x)$ with $\deg(p) = n-k$, and ω is a penalty such that $\omega \gg 1$. Then, the problem becomes the following unconstrained minimization problem where $u(x)$ represents an approximate GCD of $f(x)$ and $f'(x)$.

$$\min_{u,t,s} \|F_h(s,t)\vec{u} - \vec{b}\|_2 \ .$$

Please note that $t(x)$ is corresponding to the detected approximate square-free part $d(x)$.

The actual iteration is the following least squares.

$$\begin{pmatrix} \vec{u}_{i+1} \\ \vec{t}_{i+1} \\ \vec{s}_{i+1} \end{pmatrix} = \begin{pmatrix} \vec{u}_i - \vec{\Delta}_u \\ \vec{t}_i - \vec{\Delta}_t \\ \vec{s}_i - \vec{\Delta}_s \end{pmatrix},\ \min_{\vec{\Delta}_u,\,\vec{\Delta}_t,\,\vec{\Delta}_s} \left\| J_h(u_i,s_i,t_i) \begin{pmatrix} \vec{\Delta}_u \\ \vec{\Delta}_t \\ \vec{\Delta}_s \end{pmatrix} - \left(F_h(s_i,t_i)\vec{u}_i - \vec{b}\right) \right\|_2 \tag{3.4}$$

where $J_h(u_i,s_i,t_i)$ is the following Jacobian of $F_h(s,t)$.

$$J_h(u,s,t) = \begin{pmatrix} \vec{h}^T & & \\ C_{n-k+1}(t) & C_{k+1}(u) & \\ C'_{n-k+1}(s) & & C'_k(u) \\ \omega(C_{n-k+1}(t)_L - C'_{n-k+1}(s)) & \omega C_{k+1}(u)_L & -\omega C'_k(u) \end{pmatrix}.$$

The overall algorithm is described as Algorithm 3. Moreover, in our preliminary implementation, we use $\beta = 1$ and $h(x) = u_0(x)\ /\ \|u_0\|_2^2$ where $u_0(x)$ is the initial value of $u(x)$. We use the first singular vector of $\mathrm{S}'_{n-k-1}(f)$ as an initial guess for $s(x)$ and $t(x)$, and for $u(x)$ we solve the least squares $\min_{\vec{u}} \|C_{n-k+1}(t)\vec{u} - \vec{f}\|_2$ and $\min_{\vec{u}} \|C'_{n-k+1}(s)\vec{u} - \vec{f_L}\|_2$.

3.1.4. Algorithm based on SLRA directly

The structured low rank approximation is the problem that for the given matrix A of rank r, we find a nearby matrix $\tilde{A}$ of rank less than r where A and $\tilde{A}$ have the same structure (e.g. $\mathfrak{S}(\vec{a}) = A$ and $\mathfrak{S}(\vec{\tilde{a}}) = \tilde{A}$ for some function $\mathfrak{S}$ and vectors $\vec{a}, \vec{\tilde{a}}$). Since finding an approximate GCD is a typical SLRA problem, the constrained optimization problem in (3.1) can be considered as the following SLRA problem where $f(x)+\Delta_f(x)$ is treated as a polynomial of degree n even if its leading coefficient is zero and $\|\cdot\|_F$ denotes the Frobenius norm (sum of squares of all the elements). We note that the objective function is different from others in this subsection.

$$\min_{\Delta_f} \|\mathrm{S}'_{n-k-1}(f+\Delta_f) - \mathrm{S}'_{n-k-1}(f)\|_F \ \text{ subject to } \ \mathrm{rank}(\mathrm{S}'_{n-k-1}(f+\Delta_f)) < 2k+1. \tag{3.5}$$

For a SLRA problem, the lift-and-project method is a typical solver, however, its convergence speed is very slow. In this paper, we use the algorithm "`NewtonSLRA/1`" proposed by Schost and

Algorithm 3 Approximate square-free part with the given degree, based on the Gauss-Newton

```
Require: $f(x) \in K[x]$ and $k \in \mathbb{N}$ with $n = \deg(f)$
Ensure: $f(x) + \Delta_f(x)$, $d(x) \in K[x]$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
        such that $\deg(d) = k$ and $d(x)$ is an approximate square-free part of $f(x)$ of tolerance $\varepsilon$
1: initialize the iteration;
   - compute an initial guess for $\vec{s}$ and $\vec{t}$ by the singular value decomposition of $S'_{n-k-1}(f)$;
   - compute an initial guess for $\vec{u}$ by a least squares: $C_{n-k+1}(t)\vec{u} \approx \vec{f}$ and $C'_{n-k+1}(s)\vec{u} \approx \vec{f_L}$;
2: repeat
3:     update the current values $\vec{u}$, $\vec{t}$ and $\vec{s}$ by the Newton iteration in (3.4);
4: until the norm of correction terms is sufficiently small or divergence is detected
5: construct $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$ from the resulting $\vec{u}$, $\vec{t}$ and $\vec{s}$.
6: return $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
```

Spaenlehauer (2016) that has the local quadratic convergence property as the Newton's method though it is not proved that it will converge to the optimal of the given problem.

We briefly introduce their algorithm as follows. At first, we have to represent $S'_{n-k-1}(f) \in K^{(n+k)\times(2k+1)}$ in terms of an orthonormal basis of $K^{(n+k)\times(2k+1)}$ with the inner product: $\langle A_1, A_2\rangle = \mathrm{trace}(A_1 A_2^T)$ (i.e. the Frobenius norm). In this paper, we use the following orthonormal basis $\mathscr{B}$.

$$\mathscr{B} = \{B_0, B_1, \ldots, B_n\},\ B_i = \frac{\tilde{B}_i}{\|\tilde{B}_i\|_F},\ S'_{n-k-1}(p) = \sum_{i=0}^{n} \tilde{B}_i p_i,\ \tilde{B}_i \in K^{(n+k)\times(2k+1)},\ p(x) = \sum_{i=0}^{n} p_i x^i. \tag{3.6}$$

With this basis, let $D_{2k} \subset K^{(n+k)\times(2k+1)}$ be the set of matrices of rank $2k$, and $\mathcal{S}'_{n-k-1} \subset K^{(n+k)\times(2k+1)}$ be a subspace generated by $\mathscr{B}$. The SLRA problem in (3.5) now becomes

$$\min_{M^* \in \mathcal{S}'_{n-k-1} \cap D_{2k}} \|M - M^*\|_F$$

where $M = S'_{n-k-1}(f)$ and for $i = 0, 1, \ldots, n$, the coefficient of x^i in $f(x) + \Delta_f(x)$ can be recovered by $\langle M^*, B_i\rangle / \|\tilde{B}_i\|_F$. Their algorithm repeats the following iteration ($\Delta_f \to \Delta_f^*$), and the overall algorithm for computing an approximate square-free part is described as Algorithm 4.

Iteration 1 (One iteration of "`NewtonSLRA/1`" (Schost and Spaenlehauer, 2016))**.**

projection *Let $U\Sigma V^T$ be the singular value decomposition of $M = S'_{n-k-1}(f + \Delta_f)$ where $U = \{\overrightarrow{u_1}, \ldots, \overrightarrow{u_{n+k}}\}$, $V = \{\overrightarrow{v_1}, \ldots, \overrightarrow{v_{2k+1}}\}$ and the diagonal elements of Σ are the singular values sorted in the descending order. Then, compute the orthogonal projection $\tilde{M}$ onto D_{2k} by $U\Sigma_{2k}V^T$ where Σ_{2k} is Σ but the smallest singular value is replaced with* 0*.*

tangent intersection *Let $\vec{y} = (y_i) \in K^{n+1}$ be the solution of the least squares $A\vec{y} = \vec{b}$ where $A = (a_{ij}) \in K^{(n-k)\times(n+1)}$, $a_{ij} = \overrightarrow{u_{2k+i}}^T B_j \overrightarrow{v_{2k+1}}$ and $\vec{b} = (b_i) \in K^{n-k}$, $b_i = \overrightarrow{u_{2k+i}}^T (\tilde{M} - M)\overrightarrow{v_{2k+1}}$. Then, update $\Delta_f(x)$ by $\Delta_f^*(x) = \Delta_f(x) + \sum_{i=0}^{n} y_i / \|\tilde{B}_i\|_F\, x^i$.*

3.2. Approximate square-free part with the given tolerance

For the problem 2, we follow the unidirectional degree search approach in several approximate polynomial GCD algorithms (e.g. Zeng (2011)). We seek an approximate square-free part

Algorithm 4 Approximate square-free part with the given degree, based on SLRA

```
Require: $f(x) \in K[x]$ and $k \in \mathbb{N}$ with $n = \deg(f)$
Ensure: $f(x) + \Delta_f(x)$, $d(x) \in K[x]$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
        such that $\deg(d) = k$ and $d(x)$ is an approximate square-free part of $f(x)$ of tolerance $\varepsilon$
1: initialize the iteration;
   - compute the orthonormal basis $\mathcal{B} = \{B_0, B_1, \ldots, B_n\}$ defined in (3.6);
   - initialize the perturbation $\Delta_f(x) = 0$;
2: repeat
3:    do the iteration defined in Iteration 1;        (note: $\sum y_i / \|\tilde{B}_i\|_F \; x^i$ is the correction)
4: until the norm of correction terms is sufficiently small or divergence is detected
5: construct $d(x)$ and $\varepsilon$ from the resulting $f(x)+\Delta_f(x)$ and the singular vector of $\mathrm{S}'_{n-k-1}(f+\Delta_f)$;
6: return $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
```

Algorithm 5 Approximate square-free part with the given tolerance

```
Require: $f(x) \in K[x]$ and $\varepsilon_{\mathrm{in}} \in \mathbb{R}_{\geq 0}$ with $n = \deg(f)$
Ensure: $f(x) + \Delta_f(x)$, $d(x) \in K[x]$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
        such that $d(x)$ is an approximate square-free part of $f(x)$ of tolerance $\varepsilon$ satisfying $\varepsilon \leq \varepsilon_{\mathrm{in}}$
1: for $k = 1, 2, \ldots, n-1$ do
2:    compute an approximate square-free part of degree $k$, of $f(x)$ by Algorithms 1, 2 or 3,
                                      and let the result be $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
3:    if $\varepsilon \leq \varepsilon_{\mathrm{in}}$ then
4:       return $f(x) + \Delta_f(x)$, $d(x)$ and $\varepsilon$;
5:    end if
6: end for
7: return $f(x)$, $f(x)$ and 0.0;        (i.e. $\Delta_f(x) = 0$, $d(x) = f(x)$ and $\varepsilon = 0.0$)
```

of degree k for $k = 1, 2, \ldots, n-1$ until the resulting part satisfies the given tolerance condition. If not found even for $k = n-1$, the given polynomial is provisionally square-free within the given tolerance. The overall algorithm is described as Algorithm 5.

We note that one must compute the global minimum tolerance for each k during the unidirectional degree search if we want to completely determine the degree of approximate square-free part. Actually, for some k, if a computed tolerance is just a local minimum and it is larger than the given tolerance, it is possible that there exist another approximate square-free part of degree k within the tolerance. However, this kind of global minimization is not easy hence in our algorithms we just seek a local minimum (see Li et al. (2008); Nie et al. (2008) for further information on global minimization problems with constraints).

3.3. Approximate square-free decomposition

For the given $f(x)$ and ε it is still not a trivial task to compute an approximate square-free decomposition even if we have computed the approximate square-free part. The reason is as follows. Let the square-free decomposition of $f(x)+\Delta_f(x)$ be $f_1(x)f_2(x)^2 \cdots f_r(x)^r$ and its square-free part $d(x)$ be $f_1(x)f_2(x)\cdots f_r(x)$. The task is to compute each $f_i(x)$ from $f(x)+\Delta_f(x)$ and $d(x)$. However, $f(x) + \Delta_f(x)$ and $d(x)$ are computed by numerical arithmetics, hence we only have the

Algorithm 6 Approximate square-free decomposition based on the Musser's method

```
Require: $f(x)$, $f(x)+\Delta_f(x)$, $d(x)$ and $\varepsilon \in \mathbb{R}_{\geq 0}$
Ensure: $f_1(x)f_2(x)^2\cdots f_r(x)^r$ (the square-free decomposition of $f(x)+\Delta_f(x)$)
 1: $s_0(x) := \epsilon\text{-quo}\left(f(x)+\Delta_f(x),\ d(x),\ 0,\ 0\right)$; $t_0(x) := d(x)$; $i := 1$;
 2: while $t_{i-1}(x) \neq 1$ do
 3:     $s_i(x), f_i(x) := \epsilon\text{-cof}_\varepsilon(s_{i-1}, t_{i-1})$; $t_i(x) := \epsilon\text{-quo}(s_{i-1}, s_i, t_{i-1}, f_i)$; $i := i+1$;
 4: end while
 5: if $\deg(f_1) + 2\deg(f_2) + \cdots + (i-1)\deg(f_{i-1}) < \deg(f)$ then
 6:     return the result of the recursive call with $\varepsilon := \varepsilon \times 10$;
 7: else
 8:     the Newton refinement of $f_1(x), \ldots, f_{i-1}(x)$ by linearizing the relation
          $f(x) \approx (f_1(x)+\Delta_1(x))(f_2(x)+\Delta_2(x))^2\cdots(f_{i-1}(x)+\Delta_{i-1}(x))^{i-1}$ w.r.t. $\Delta_1(x), \ldots, \Delta_{i-1}(x)$;
 9:     return $f_1(x)f_2(x)^2\cdots f_{i-1}(x)^{i-1}$;
10: end if
```

following relations:

$$\left\{\begin{array}{rcl} f(x)+\Delta_f(x) & = & f_1(x)f_2(x)^2\cdots f_r(x)^r + \varepsilon_1(x) \\ d(x) & = & f_1(x)f_2(x)\cdots f_r(x) + \varepsilon_2(x) \\ f(x)+\Delta_f(x) & = & d(x) \times \gcd\left(f(x)+\Delta_f(x),\ \dfrac{\partial}{\partial x}(f(x)+\Delta_f(x))\right) + \varepsilon_3(x) \end{array}\right. \tag{3.7}$$

where each $\|\varepsilon_i(x)\|$ must be very tiny but not negligible for getting each $f_i(x)$.

To get each $f_i(x)$, the method by Musser (1971) is most straightforward. In this case all the computations of polynomial GCDs and quotients must be replaced with numerical variations that can handle tiny numerical errors, as in Sasaki and Noda (1989). For example, we can use any approximate GCD algorithm with tiny tolerance ($\approx$ *the machine epsilon*) for polynomial GCDs, and the least squares method or FFT for quotients.

In this paper, we simply use the singular value decomposition of the subresultant matrix for computing the cofactors by approximate GCD, based on Lemma 2, and the least squares for computing the quotients. Let $\epsilon\text{-cof}_\varepsilon(f,g)$ be the polynomials $t_\varepsilon(x)$, $s_\varepsilon(x)$ such that $(\vec{s_\varepsilon}^T \mid -\vec{t_\varepsilon}^T)^T$ is the last singular vector of $\mathrm{S}_r(f,g)$ and r is the largest integer for which the smallest singular value of $\mathrm{S}_r(f,g)$ is not larger than ε. By Lemma 2, $t_\varepsilon(x)$ and $s_\varepsilon(x)$ become a rough approximation of $t(x)$ and $s(x)$ that are the cofactors of $f(x)$ and $g(x)$ by their GCD, respectively. For computing the corresponding approximate GCD $u_\varepsilon(x)$, we solve the least squares $\min_{u_\varepsilon} \|t_\varepsilon(x)u_\varepsilon(x) - f(x)\|_2$ and $\min_{u_\varepsilon} \|s_\varepsilon(x)u_\varepsilon(x) - g(x)\|_2$. We denote it by $\epsilon\text{-quo}(f, t_\varepsilon, g, s_\varepsilon)$.

However, it is difficult to determine the suitable value of ε in advance. In our algorithm, we simply use the unidirectional search for the suitable value from $\|\Delta_f\|_2 \ / \ \|f\|_2$ by the recursive call. Then, the overall algorithm is described as Algorithm 6. Note that the number of recursive calls in Algorithm 6 is bounded, and the algorithm terminates since the 2-norm of $\mathrm{S}_r(s_{i-1}, t_{i-1})$ is bounded and for not small ε we have $\deg(s_i) < \deg(s_{i-1})$ and $\deg(t_i) < \deg(t_{i-1})$. Moreover, to make the numerical errors negligible, we refine the resulting $f_1(x), \ldots, f_r(x)$ by the Newton refinement steps of $f(x) \approx (f_1(x)+\Delta_1(x))(f_2(x)+\Delta_2(x))^2\cdots(f_r(x)+\Delta_r(x))^r$ w.r.t. $\Delta_1(x), \ldots, \Delta_r(x)$.

4. Numerical examples

We have prepared our preliminary implementation of our algorithms on Mathematica 12.1 where we have limited the number of iterations to 16 and used the penalty weight $\omega = 10^8$, and we have done some numerical experiments. Our preliminary implementation and examples in this paper can be available at the following URL:
`https://wwwmain.h.kobe-u.ac.jp/~nagasaka/research/snap/aSF.zip`

4.1. Comparison against another algorithm for approximate square-free decomposition

4.1.1. Example 4.5 from Sasaki and Noda (1989)

We have computed the approximate square-free part and decomposition for the following polynomial appeared in Sasaki and Noda (1989) and compare their results with ours.

$$f(x) = (x+1)(x-2)(x-0.5)^2(x-0.501)(x-0.503)(x+0.5)^2(x+0.501)(x+0.503)$$

In Sasaki and Noda (1989), they solved a problem similar to the Problem 2 for $f(x)$. However, their definition is different from ours. Briefly they use $\delta^2/c \le \|\Delta_f\|_\infty \le \delta^2 c$ (c is not much different from 1) for $\delta \in \mathbb{R}_{\ge 0}$ as tolerance, where $\|\cdot\|_\infty$ denotes the maximum absolute value of coefficients. Roughly speaking, we have $\varepsilon \approx \delta^2\ \|f\|_2$. Their result is as follows (note that for the latter result, there is no full expression hence we cite it as is).

$\delta = 0.01$ (tolerance for Problem 2: $0.01^2\ \|f\|_2$)

$d(x) = x^4 - 1.00002x^3 - 2.25099x^2 + 0.250999x + 0.502005$
$f(x) + \Delta_f = (x^2 - 1.00002242x - 1.99998004)(x^2 + 5.63119694\text{e-}6x - 2.5100488\text{e-}1)^4$
$\varepsilon = \|\Delta_f\|_2 / \|f\|_2 = 1.17899\text{e-}5,\quad \varepsilon\ \|f\|_2 = \|\Delta_f\|_2 = 5.07931\text{e-}5 < 0.01^2 = 1.0\text{e-}4$

$\delta = 0.0001$ (tolerance for Problem 2: $0.0011^2\ \|f\|_2$)

$f(x) + \Delta_f = (x^6 - 9.99999907\text{e-}1x^5 - 2.50401007x^4 + \cdots)$
$(x^2 - 4.6684407\text{e-}8x - 2.5000001\text{e-}1)^2$

Our result by Algorithm 2 (based on STLN) and Algorithm 6 is as follows.

$\delta = 0.01$ (tolerance for Problem 2: $0.01^2\ \|f\|_2$)

$d(x) = 1.00001x^4 - 1.00001x^3 - 2.25103x^2 + 0.251004x + 0.502009$
$f(x) + \Delta_f = (1.00001x^2 - 1.00001x - 2.00003)(0.999997x^2 - 2.94538\text{e-}7x - 0.251001)^4$
$\varepsilon = \|\Delta_f\|_2 / \|f\|_2 = 1.20262\text{e-}6,\quad \varepsilon\ \|f\|_2 = \|\Delta_f\|_2 = 5.18113\text{e-}6 < 0.01^2 = 1.0\text{e-}4$

$\delta = 0.0001$ (tolerance for Problem 2: $0.0001^2\ \|f\|_2$)

$d(x) = 1.0x^6 - 1.0x^5 - 2.50342x^4 + 0.503423x^3 + 1.0702x^2 - 0.0633568x - 0.126714$
$f(x) + \Delta_f = (1.0x^4 - 1.0x^3 - 2.25313x^2 + 0.25313x + 0.506261)$
$(1.0x^2 - 3.4945\text{e-}8x - 0.250293)^3$
$\varepsilon = \|\Delta_f\|_2 / \|f\|_2 = 3.53551\text{e-}10,\quad \varepsilon\ \|f\|_2 = \|\Delta_f\|_2 = 1.52316\text{e-}9 < 0.0001^2 = 1.0\text{e-}8$

Moreover, even if our algorithm does not refine the results, the resulting tolerances do not change up to the leading 10 and 3 figures, respectively.

4.1.2. *Example from Diaz-Toca and Gonzalez-Vega (2006)*

We have computed the approximate square-free part and decomposition for the following polynomial appeared in Diaz-Toca and Gonzalez-Vega (2006) and compare their and our results.

$$f(x) = x^6 - 21x^5 + 45x^4 + 1225x^3 - 3749.969x^2 - 22500.0021x + 24999.999999$$

In Diaz-Toca and Gonzalez-Vega (2006), they solved the Problem 2 for $f(x)$ with the tolerance $\varepsilon = 0.1/\|f\|_2$ and the result is as follows, where we show only leading 10 figures thought their result has 18 figures at most.

$$\begin{aligned}
&d(x) = 1.000007498x^3 - 6.000047938x^2 - 45.00017270x + 50.00016320\\
&f(x)+\Delta_f = (1.000006742x - 1.000005817)(1.000000756x + 4.999991457)^2(x - 9.999991556)^3\\
&\varepsilon = \|\Delta_f\|_2 / \|f\|_2 = 1.627600886\text{e-}6,\ \ \varepsilon\,\|f\|_2 = \|\Delta_f\|_2 = 0.05511815678 < 0.1
\end{aligned}$$

Our result by Algorithm 2 (based on STLN) and Algorithm 6 is as follows.

$$\begin{aligned}
&d(x) = 0.9999998918x^3 - 5.999976724x^2 - 45.00002709x + 50.00006154\\
&f(x) + \Delta_f = (0.9999999060x - 1.000000973)(0.9999999855x + 5.000008732)^2\\
&\qquad\qquad (1.000000000x - 9.999985113)^3\\
&\varepsilon = \|\Delta_f\|_2 / \|f\|_2 = 2.763805129\text{e-}8,\ \ \varepsilon\,\|f\|_2 = \|\Delta_f\|_2 = 0.0009359533148 < 0.1
\end{aligned}$$

Moreover, even if our algorithm does not refine the result, the resulting tolerance does not change up to the leading 10 figures.

4.2. *Comparison against algorithms for nearest singular polynomial*

We have computed the approximate square-free parts and decompositions for polynomials appeared in the related papers on nearest singular polynomial, by Algorithm 2 (based on STLN) and Algorithm 6. In the followings, ZNKW and STLN denote the results by the algorithms in Zhi et al. (2004) and Kaltofen et al. (2006), respectively. Please note that k denotes the degree of expected square-free part hence the specified multiplicity for computing nearest singular polynomial is $\deg(f) - k + 1$ (which is denoted by k in the related papers since their notations are different from ours).

4.2.1. *Example 2 from Zhi et al. (2004)*

$$f(x) = x^5 + 2.03x^4 - 0.9398x^3 - 2.0296x^2 - 0.0602x - 0.0004$$

The resulting approximate square-free part and decomposition are as follows. The significant difference from the others is the multiplicity structure. For example, in the case of $k = 3$, our method found the two double roots instead of a triple root.

k=4	$d(x) = 1.0x^4 + 2.01501x^3 - 0.970011x^2 - 2.01506x - 0.0299872$ $f(x) + \Delta_f = (1.0x^3 + 2.00001x^2 - 0.999998x - 2.00006)(1.0x + 0.0149931)^2$
k=3	$d(x) = 1.06412x^3 + 0.417207x^2 - 1.44132x - 0.011022$ $f(x) + \Delta_f = (1.02674x - 1.01505)(1.0364x^2 + 1.43094x + 0.0108586)^2$
k=2	$d(x) = 1.09327x^2 + 0.322898x - 0.69504$ $f(x) + \Delta_f = (1.04289x - 0.691668)^2(1.04831x + 1.00487)^3$

Table 2 shows the resulting tolerances and "n/a" denotes no data in the related papers. Please note that the values in the related papers are originally appeared as $\|\Delta_f\|_2^2$ hence we converted them.

k	4	3	2
ZNKW	0.0000155888	0.190673	n/a
STLN	0.0000155904	0.190673	n/a
Ours	0.0000155888	0.046158	0.190452

Table 2: The resulting tolerances of Example 2 from Zhi et al. (2004): $\|\Delta_f\|_2 / \|f\|_2$

k	5	4	3
ZNKW	5.43876e-7	0.000270659	0.209105
STLN	5.43876e-7	0.000270659	0.209105
Ours	5.43876e-7	0.000270659	0.045681

Table 3: The resulting tolerances of Example 4 from Zhi et al. (2004): $\|\Delta_f\|_2 / \|f\|_2$

4.2.2. *Example 4 from Zhi et al. (2004)*

$$f(x) = x^6 + 2.04x^5 - 0.9199x^4 - 2.03981x^3 - 0.080112x^2 - 0.000194x + 0.000012$$

The resulting approximate square-free part and decomposition are as follows. In this case, the computed multiplicity structure for $k = 3$ is different from the others. Moreover, Table 3 shows the resulting tolerances.

k=5	$d(x) = 1.0x^5 + 2.01466x^4 - 0.970961x^3 - 2.0152x^2 - 0.0290374x + 0.0005419$ $f(x) + \Delta_f = (1.0x^4 + 1.98931x^3 - 1.02138x^2 - 1.98931x + 0.0213812)(1.0x + 0.0253447)^2$
k=4	$d(x) = 1.0x^4 + 2.01349x^3 - 0.973449x^2 - 2.01436x - 0.0265223$ $f(x) + \Delta_f = (1.0x^3 + 2.00024x^2 - 0.999959x - 2.0011)(1.0x + 0.0132538)^3$
k=3	$d(x) = 1.03823x^3 + 0.408171x^2 - 1.40706x - 0.0117932$ $f(x) + \Delta_f = (1.00456x - 0.993426)(1.00625x + 1.38228)^2(1.0271x + 0.00858814)^3$

4.2.3. *Example 6 from Zhi et al. (2004)*

$$\begin{aligned} f(x) = {} & x^{21} - 1.14286x^{20} - 1.x^{19} + 2.71429x^{18} - 4.x^{17} + 4.14287x^{16} - 2.57137x^{15} + x^{14} \\ & + 0.857143x^{13} - 3.14286x^{12} + 2.x^{11} + 0.285714x^{10} + 0.571428x^8 - 1.2856x^7 + 2.85714x^6 \\ & - 4.71429x^5 + 2.14286x^4 + 0.428571x^3 + 0.857143x^2 - 0.714286x - 0.2857 \end{aligned}$$

k	20	19
ZNKW	4.26708e−6	0.000959838
STLN	4.25427e−6	0.000959829
Ours	4.15649e−6	0.000862634

Table 4: The resulting tolerances of Example 6 from Zhi et al. (2004): $\|\Delta_f\|_2 \,/\, \|f\|_2$

The resulting approximate square-free part and decomposition are as follows. Moreover, Table 4 shows the resulting tolerances.

k=20	$d(x) = 0.999995x^{20} - 0.142891x^{19} - 1.1429x^{18} + 1.5714x^{17} - 2.42865x^{16} + 1.71427x^{15} - 0.857146x^{14} + 0.142862x^{13} + 0.999997x^{12} - 2.1429x^{11} - 0.142866x^{10} + 0.142843x^{9} + 0.142831x^{8} + 0.714251x^{7} - 0.571375x^{6} + 2.28578x^{5} - 2.42857x^{4} - 0.285676x^{3} + 0.142892x^{2} + 1.00003x + 0.285715$
	$f(x) + \Delta_f = (1.0x^{19} + 0.857094x^{18} - 0.285826x^{17} + 1.28559x^{16} - 1.14309x^{15} + 0.571208x^{14} - 0.285951x^{13} - 0.143084x^{12} + 0.85692x^{11} - 1.286x^{10} - 1.42885x^{9} - 1.28598x^{8} - 1.14313x^{7} - 0.42886x^{6} - 1.00023x^{5} + 1.28557x^{4} - 1.14303x^{3} - 1.42869x^{2} - 1.28577x - 0.28572)(0.999996x - 0.999981)^2$
k=19	$d(x) = 1.00125x^{19} + 0.943787x^{18} - 0.114555x^{17} + 1.45277x^{16} - 0.845798x^{15} + 0.807891x^{14} + 0.0277988x^{13} + 0.183609x^{12} + 1.2067x^{11} - 0.825871x^{10} - 1.02432x^{9} - 0.952735x^{8} - 0.874991x^{7} - 0.220941x^{6} - 0.798356x^{5} + 1.42792x^{4} - 0.875235x^{3} - 1.22473x^{2} - 1.17477x - 0.266284$
	$f(x) + \Delta_f = (1.0x^{18} + 1.98164x^{17} + 1.94454x^{16} + 3.47137x^{15} + 2.76207x^{14} + 3.67672x^{13} + 3.84794x^{12} + 4.18146x^{11} + 5.5498x^{10} + 4.9415x^{9} + 4.11125x^{8} + 3.32011x^{7} + 2.57575x^{6} + 2.45558x^{5} + 1.75403x^{4} + 3.24861x^{3} + 2.50121x^{2} + 1.3756x + 0.255966)(1.00125x - 1.04031)^3$

4.2.4. Example 1 from Lihong and Wenda (1998)

$$f(x) = 1.0x^5 - 1.0x$$

The resulting approximate square-free part and decomposition are as follows.

k=4	$d(x) = 1.01179x^4 + 0.623449x^3 + 0.426389x^2 + 0.360152x - 0.585401$
	$f(x) + \Delta_f = (1.00149x^3 + 1.19661x^2 + 1.11446x + 1.00137)(1.01029x - 0.584603)^2$
k=3	$d(x) = 1.0x^3 + 8.50966\mathrm{e}{-8}x^2 + 7.52996\mathrm{e}{-12}x + 1.47227\mathrm{e}{-18}$
	$f(x) + \Delta_f = (1.0x + 1.94967\mathrm{e}{-7})(1.0x^2 - 1.09871\mathrm{e}{-7}x + 7.55138\mathrm{e}{-12})^2$

Please note that the result for $k = 3$ is still far from the others and is almost failed, and our algorithm could not detect any non-trivial square-free part for $k = 3$ if we use Mathematica 10.2 (our results are computed on Mathematica 12.1). Moreover, if we use the penalty weight $\omega = 10^9$ (and relax our internal iteration stop criterion to prevent any divergence in vectors), the result for $k = 3$ becomes $d(x) = 21.1923x^3 + 7.65314x^2 - 3.11509x + 5.32198$, $f(x) + \Delta_f = (476.655x + 412.716)(0.0444605x^2 - 0.0224406x + 0.012895)^2$ and $\|\Delta_f\|_2 \,/\, \|f\|_2 = 0.670147$ though this is still worse than the others.

k	4	3
ZNKW	0.296926	0.559514
STLN	0.296926	0.559514
Ours	0.296502	0.707107

Table 5: The resulting tolerances of Example 1 from Lihong and Wenda (1998): $\|\Delta_f\|_2 / \|f\|_2$

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)					Problem 2 (ε = 1.0e−5)			
		#Fs	#Ls	k-avg.	ε-avg.	ctime	#Fs	k-avg.	ε-avg.	ctime
$(0, 2, 0)$	4	0	0	2.00	6.5364e−6	0.0146	0	1.99	6.5895e−6	0.0209
$(0, 1, 2)$	8	0	9	3.00	8.0131e−4	0.0662	2	3.16	7.4270e−6	0.1264
$(8, 1, 0)$	10	0	0	9.00	2.5452e−6	1.4120	0	8.99	2.5232e−6	2.9293
$(1, 0, 3)$	10	5	37	4.30	2.7482e−2	0.1277	1	4.81	6.6023e−6	0.5161
(1,1,1,1,1)	15	0	75	5.00	4.5850e−1	2.3939	16	9.06	5.9689e−6	23.717
$(31, 0, 3)$	40	0	64	34.0	7.3832e−2	44.839	11	35.7	3.4199e−6	582.14

Table 6: The results of newly generated examples (with Algorithm 1)

4.3. Newly generated examples

We have generated the 6 sets of 100 polynomials (multiplicity structures are $(0, 2, 0)$, $(0, 1, 2)$, $(8, 1, 0)$, $(1, 0, 3)$, $(1, 1, 1, 1, 1)$ and $(31, 0, 3)$ where (e_1, e_2, e_3) and $(e_1, e_2, e_3, e_4, e_5)$ denote that the number of roots of multiplicity k is e_k) as follows, and computed their approximate square-free parts and decompositions by Algorithms 5 and 6 with Algorithms 1, 2, 3 or 4.

1. generate a monic polynomial $\tilde{f}(x) = \tilde{f}_1(x)\tilde{f}_*(x) \in \mathbb{R}[x]$ where $\tilde{f}_1(x)$ is a monic polynomial of degree e_1 whose coefficients are generated from $[-1, 1]$,and $\tilde{f}_*(x)$ is a monic polynomial that has the specified multiplicity structure $(e_2, e_3, \ldots)$ with real roots in $[-1, 1]$ randomly.
2. generate a polynomial $\tilde{\Delta}_f(x) \in \mathbb{R}[x]$ of degree $\deg(\tilde{f}) - 1$, whose coefficients are generated from $[-1, 1]$ randomly.
3. construct $f(x)$ as $\tilde{f}(x) + 10^{-5}\dfrac{\|\tilde{f}(x)\|_2}{\|\tilde{\Delta}_f(x)\|_2}\tilde{\Delta}_f(x)$.

The tables 6, 7, 8, 9 and 10 show the results where "#Fs" denotes the number of polynomials that our algorithms could not compute any non-trivial approximate square-free part, "#Ls" denotes the number of polynomials that our algorithms could not compute any non-trivial approximate square-free part of the specified degree k within the expected tolerance ε = 1.0e−5, "k-avg." denotes the average of degrees of resulting approximate square-free parts including trivial ones, "ε-avg." denotes the average of resulting tolerances (only non-trivial ones), and "ctime" denotes the average of cpu time (sec.) (measured by `Timing` on the machine with Intel Core i7-5960X). Please note that 1) a smaller value is better, 2) we show only leading 5 or 3 figures to save the space, and 3) the cpu time is just a reference since our implementations on Mathematica are really preliminary. For example, according to our experiments (the data set "half" in Nagasaka (2020), polynomials of degree 50), our implementations of the UVGCD (relevant to Algorithm 3), STLNGCD (relevant to Algorithm 2) and GHLGCD (relevant to Algorithm 1) algorithms on Mathematica are approximately 40, 850 and 3000 times slower than our C implementations, respectively.

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)					Problem 2 (ε = 1.0e−5)			
		#Fs	#Ls	k-avg.	ε-avg.	ctime	#Fs	k-avg.	ε-avg.	ctime
$(0, 2, 0)$	4	0	0	2.00	6.5364e−6	0.0051	0	1.99	6.5895e−6	0.0108
$(0, 1, 2)$	8	0	8	3.00	3.2260e−5	0.0104	0	3.09	7.4197e−6	0.0272
$(8, 1, 0)$	10	0	0	9.00	2.5448e−6	0.0119	0	8.99	2.5232e−6	0.1273
$(1, 0, 3)$	10	0	35	4.00	1.6707e−2	0.0172	0	4.64	6.5806e−6	0.0570
(1,1,1,1,1)	15	0	68	5.00	2.1282e−1	0.0314	16	8.24	6.8571e−6	0.2109
$(31, 0, 3)$	40	0	55	34.0	4.7313e−2	1.9198	23	36.2	3.6819e−6	44.754

Table 7: The results of newly generated examples (with Algorithm 2)

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)					Problem 2 (ε = 1.0e−5)			
		#Fs	#Ls	k-avg.	ε-avg.	ctime	#Fs	k-avg.	ε-avg.	ctime
$(0, 2, 0)$	4	0	0	2.00	6.5364e−6	0.0087	0	1.99	6.5895e−6	0.0136
$(0, 1, 2)$	8	0	10	3.00	8.6069e−6	0.0163	0	3.11	7.3061e−6	0.0374
$(8, 1, 0)$	10	0	0	9.00	2.5505e−6	0.0157	0	8.99	2.5232e−6	0.1181
$(1, 0, 3)$	10	0	28	4.00	5.3960e−4	0.0209	0	4.50	6.9000e−6	0.0690
(1,1,1,1,1)	15	0	71	5.00	1.3060e−1	0.0426	8	8.82	6.7322e−6	0.2631
$(31, 0, 3)$	40	0	75	34.0	3.3350e−2	1.4321	14	37.4	3.5735e−6	54.299

Table 8: The results of newly generated examples (with Algorithm 3)

5. Concluding Remarks

We have shown the issues of known algorithms for approximate square-free decomposition and nearest singular polynomial when we want to compute approximate square-free part and decomposition within the specified tolerance, or of the specified degree. To overcome this situation, we have given the new explicit definition and the algorithms for computing them, based on the approximate GCD indirectly. In fact, Algorithms 1, 2 and 4 do not need to compute any approximate GCD and just compute its approximate cofactors only.

For the polynomials that are appeared in the related known studies, we have shown that our algorithms can compute non-trivial approximate square-free decompositions whose tolerance are better than the known methods and multiplicity structures are more complicated ones (e.g. one double root and one triple root instead of one quadruple root). For the randomly generated polynomials, we have demonstrated that our algorithms work well for the multiplicity structures $(0, 2, 0)$, $(0, 1, 2)$, $(8, 1, 0)$, $(1, 0, 3)$, $(1, 1, 1, 1, 1)$ and $(31, 0, 3)$. Especially, for $(0, 2, 0)$ and $(8, 1, 0)$, we found approximate square-free parts of lower degrees (meaning better ones) than expected.

However, for $(0, 1, 2)$, $(1, 0, 3)$, $(1, 1, 1, 1, 1)$ and $(31, 0, 3)$, the detected degrees of approximate square-free parts (i.e. the detected number of distinct roots) are a little bit larger than the expected ones. The column of "#Ls" in the tables 6, 7, 8, 9 and 10 indicates that the reason is as follows. In general, polynomials with clusters of roots have many possible approximate square-free parts and decompositions (i.e. there are many local optimal points) since their locations of roots are hypersensitive to the changes in the coefficients like the well-known Wilkinson's polynomial. In fact, the table 11 shows only a little bit better results even with the new initialization method proposed in Kaltofen et al. (2006) and the larger maximum number of iterations (max 128 iterations).

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)					Problem 2 (ε = 1.0e−5)			
		#Fs	#Ls	k-avg.	ε-avg.	ctime	#Fs	k-avg.	ε-avg.	ctime
$(0, 2, 0)$	4	0	0	2.00	6.5364e−6	0.0041	0	2.07	6.2103e−6	0.0061
$(0, 1, 2)$	8	0	9	3.00	1.1833e−5	0.0160	0	3.20	7.2971e−6	0.0342
$(8, 1, 0)$	10	0	0	9.00	2.5477e−6	0.0138	0	9.00	2.5477e−6	0.1502
$(1, 0, 3)$	10	1	34	4.06	1.2994e−3	0.0286	0	4.86	6.5564e−6	0.0938
(1,1,1,1,1)	15	0	71	5.00	1.7052e−1	0.0853	0	7.70	6.7797e−6	0.4876
$(31, 0, 3)$	40	0	50	34.0	3.2408e−2	8.6485	5	34.9	4.4166e−6	261.93

Table 9: The results of newly generated examples (with Algorithm 4)

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)				Problem 2 (ε = 1.0e−5)		
		#Fs	#Ls	k-avg.	ε-avg.	#Fs	k-avg.	ε-avg.
$(0, 2, 0)$	4	0	0	2.00	6.5364e−6	0	1.99	6.2103e−6
$(0, 1, 2)$	8	0	5	3.00	7.8930e−6	0	3.04	7.1790e−6
$(8, 1, 0)$	10	0	0	9.00	2.5448e−6	0	8.99	2.5232e−6
$(1, 0, 3)$	10	0	18	4.00	6.2508e−5	0	4.13	5.8100e−6
(1,1,1,1,1)	15	0	51	5.00	2.4956e−2	0	6.10	4.9997e−6
$(31, 0, 3)$	40	0	43	34.0	8.8354e−3	0	34.2	2.6608e−6

Table 10: The results of newly generated examples (best for each polynomial, among the algorithms)

Acknowledgements

The author wishes to thank the anonymous reviewers for their constructive comments, and this work was supported by JSPS KAKENHI Grant Number 15K00016 and 19K11827.

References

Boito, P., 2011. Structured matrix based methods for approximate polynomial GCD. Vol. 15 of Tesi. Scuola Normale Superiore di Pisa (Nuova Series) [Theses of Scuola Normale Superiore di Pisa (New Series)]. Edizioni della Normale, Pisa.

Chu, M. T., Funderlic, R. E., Plemmons, R. J., 2003. Structured low rank approximation. Linear Algebra Appl. 366, 157–172, special issue on structured matrices: analysis, algorithms and applications (Cortona, 2000).

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

Dunaway, D. K., 1974. Calculation of zeros of a real polynomial through factorization using Euclid's algorithm. SIAM J. Numer. Anal. 11, 1087–1104.

Giesbrecht, M., Haraldson, J., Labahn, G., 2017. Computing the nearest rank-deficient matrix polynomial. In: ISSAC'17—Proceedings of the 2017 ACM International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 181–188.

Kaltofen, E., Yang, Z., Zhi, L., 2006. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In: ISSAC'06—Proceedings of the 2006 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 169–176.

Karmarkar, N., Lakshman, Y. N., 1996. Approximate polynomial greatest common divisors and nearest singular polynomials. In: ISSAC'96—Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation. ACM, New York, pp. 35–39.

Li, B., Nie, J., Zhi, L., 2008. Approximate GCDs of polynomials and sparse SOS relaxations. Theoret. Comput. Sci. 409 (2), 200–210.

Li, Z., Zhi, L., 2013. Computing the nearest singular univariate polynomials with given root multiplicities. Theoret. Comput. Sci. 479, 150–162.

structure	n	Problem 1 ($k = 2, 3, 9, 4, 5, 34$)				Problem 2 (ε = 1.0e−5)		
		#Fs	#Ls	k-avg.	ε-avg.	#Fs	k-avg.	ε-avg.
$(1, 0, 3)$	10	0	14	4.00	1.4712e−5	0	4.08	6.0149e−6
(1,1,1,1,1)	15	0	47	5.00	1.1049e−2	0	5.72	5.0380e−6

Table 11: The results with different configurations (best for each polynomial, among the algorithms)

Lihong, Z., Wenda, W., 1998. Nearest singular polynomials. J. Symbolic Comput. 26 (6), 667–675.

Maróti, A., 2003. On elementary lower bounds for the partition function. Integers 3, A10, 9.

Musser, D. R., 1971. Algorithms for Polynomial Factorization. Ph.D. Thesis, Technical Report #134. Computer Science Department, University of Wisconsin.

Nagasaka, K., 2017. Seeking better algorithms for approximate GCD. ACM Commun. Comput. Algebra 51 (1), 15–17.

Nagasaka, K., 2020. Toward the best algorithm for approximate gcd of univariate polynomials. J. Symbolic Comput.Special issue on MICA 2016. (in press).

Nie, J., Demmel, J., Gu, M., 2008. Global minimization of rational functions and the nearest GCDs. J. Global Optim. 40 (4), 697–718.

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

Schost, E., Spaenlehauer, P.-J., 2016. A quadratically convergent algorithm for structured low-rank approximation. Found. Comput. Math. 16 (2), 457–492.

Yun, D. Y., 1976. On square-free decomposition algorithms. In: Proceedings of the Third ACM Symposium on Symbolic and Algebraic C omputation. SYMSAC ’76. ACM, New York, NY, USA, pp. 26–35.

Zeng, Z., 2011. The numerical greatest common divisor of univariate polynomials. In: Randomization, relaxation, and complexity in polynomial equation solving. Vol. 556 of Contemp. Math. Amer. Math. Soc., Providence, RI, pp. 187–217.

Zhi, L., Noda, M.-T., Kai, H., Wu, W., 2004. Hybrid method for computing the nearest singular polynomials. Japan J. Indust. Appl. Math. 21 (2), 149–162.