An ODE-based method for computing the Approximate Greatest Common Divisor of polynomials.

Fazzi, Antonio; Guglielmi , N.; Markovsky, Ivan

Published in:
Numerical Algorithms

DOI:
10.1007/s11075-018-0569-0

Publication date:
2018

Document Version:
Submitted manuscript

Noname manuscript No.
(will be inserted by the editor)

An ODE based method for computing the Approximate Greatest Common Divisor of polynomials

Antonio Fazzi · Nicola Guglielmi · Ivan Markovsky

the date of receipt and acceptance should be inserted later

Abstract Computing the greatest common divisor of a set of polynomials is a problem which plays an important role in different fields, such as linear system, control and network theory. In practice, the polynomials are obtained through measurements and computations, so that their coefficients are inexact. This poses the problem of computing an approximate common factor. We propose an improvement and a generalization of the method recently proposed in [12], which restates the problem as a (structured) distance to singularity of the Sylvester matrix. We generalize the algorithm in order to work with more than 2 polynomials and to compute an Approximate GCD (Greatest Common Divisor) of degree $k \geq 1$; moreover we show that the algorithm becomes faster by replacing the eigenvalues by the singular values.

Keywords Sylvester matrix · iterative methods · Approximate GCD · polynomials

Mathematics Subject Classification (2000) 15A24 · 65K10

A. Fazzi · N. Guglielmi
Gran Sasso Science Institute (GSSI), Viale F. Crispi 7, 67010 L' Aquila, Italy
E-mail: antonio.fazzi@gssi.it

N. Guglielmi
Università degli Studi di L' Aquila, Via Vetoio 1 - Loc. Coppito, 67100 L' Aquila, Italy
E-mail: nicola.guglielmi@univaq.it

I. Markovsky
Vrije Universiteit Brussel (VUB), Department ELEC, Pleinlaan 2, 1050 Brussels, Belgium
E-mail: imarkovs@vub.ac.be

1 Introduction

1.1 Definition of the problem

Consider a set of l polynomials $p_1, \dots, p_l$ whose degrees are $n_1, \dots, n_l$ respectively. We assume that $p_1, \dots, p_l$ are coprime (they have no common roots). A challenging problem treated in the literature [14,9] is computing the closest set (in a specified sense) of polynomials $\hat{p}_1, \dots, \hat{p}_l$ which have an a priori specified number of common roots. The problem has been widely studied in the past years, especially in the case $l = 2$, and there exist several algorithms for its solution [1,3,31,4,28].

In the following we will assume that all the polynomials have real coefficients. The extension to complex polynomials is straightforward. A polynomial (of degree n) will be represented in its canonical form

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0$$

or equivalently by the vector of its coefficients

$$p = (a_n, a_{n-1}, \dots, a_1, a_0).$$

We will define the distance between two sets of l polynomials as follows:

$$dist(\{p_1, \dots, p_l\}, \{\hat{p}_1, \dots, \hat{p}_l\}) = \sqrt{\sum_{i=1}^{l} \sum_{j=0}^{n_i} (a_{ij} - \hat{a}_{ij})^2} \tag{1.1}$$

where a_{ij} stands for the j-th coefficient of the i-th polynomial.

Defining the set

$$\begin{aligned} D_k &= D_k(n_1, \dots, n_l) = \\ &= \{\{\hat{p}_1, \dots, \hat{p}_l\} : deg(\hat{p}_i) = n_i \ \forall i, \ \hat{p}_1, \dots, \hat{p}_l \text{ have k common roots}\}, \end{aligned}$$

the Approximate GCD (AGCD) problem is the following one:

Problem 1.1 Given $k \in \mathbb{N}$, a set of l polynomials $p_1, \dots, p_l$, and the distance given in (1.1), compute:

$$\inf_{\{\hat{p}_1, \dots, \hat{p}_l\} \in D_k} dist(\{p_1, \dots, p_l\}, \{\hat{p}_1, \dots, \hat{p}_l\}). \tag{1.2}$$

1.2 Previous work

Problem 1.1 has been studied in the past years and several algorithms have been proposed for its solution. Most methods for solving Problem 1.1 use local optimization approaches. We list here some of these algorithms, which can be divided in two main classes. One approach is based on the representation of the polynomial as a product of a common factor and a quotient, and on the minimization of the related cost function. This is a nonlinear least squares

problem, firstly analyzed in [6]. The problem is solved by different iterative methods in [31] and [2]. In this framework several authors use the variable projection principle, starting from [11] and [15]. Further analysis in the case of weighted and unweighted norms were developed in [7,23,21].

On the other hand there are algorithms which solve the problem approximating a given structured matrix by a structured matrix of lower rank (a structured low rank approximation, SLRA, problem [19]). This is done exploiting the link between the rank constraint on the resultant matrix and the degree of the GCD. It is considered a classical Sylvester matrix in the case of two polynomials [32], a generalized Sylvester matrix for more than two polynomials [13,14,24]. More algorithms for the AGCD computation in the same framework are the structured total least norm (STLN) approach [13, 16], the gradient projection method [26], and a recent Structured Low Rank Approximation algorithm [25] based on a Newton-like iteration.

Some new methods have been proposed in more recent papers [5,17,30].

1.3 Contributions and structure of this paper

A new algorithm that restates the problem as a structured distance to singularity of the Sylvester matrix associated to the data polynomials is presented in [12]. It exploits the well known equivalence between the degree of the GCD and the rank constraint on the Sylvester matrix. The method solves the nonconvex optimization Problem 1.1 by a local optimization approach in the sub-variety of polynomials which have a GCD: this is done by integrating a system of ordinary differential equations describing the dynamic on the singular values of the Sylvester matrix. The algorithm in [12] has been developed only for computing an AGCD of degree one between two polynomials.

The goal of this paper is to improve and generalize the algorithm proposed in [12]. Replacing the eigenvalues by the singular values speeds up the algorithm making it competitive with other existing methods (this was a deficiency in the case of eigenvalues). Moreover the singular values allow to work with rectangular matrices (needed if we consider more than two polynomials), and they simplify some computations since they are non negative real numbers. Hence we can generalize the method considering an arbitrary number of polynomials and an AGCD of fixed degree $k \geq 1$. Moreover, similarly to the case of eigenvalues, we can add constraints on the coefficients if some of them are known exactly (such constraints can not be treated by other methods).

The paper is organized as follows. In Sections 2 we introduce some useful contents in a general framework: the generalized Sylvester matrix and its singular values and the basic idea of the 2-level algorithm proposed in [12]. In Sections 3 and 4 we examine the two levels of the algorithm, which are the inner and the outer iterations; first we look for the stationary points of a gradient system of ordinary differential equations describing the dynamic on the singular values of the generalized Sylvester matrix, then we consider an iterative method to compute the perturbed Sylvester matrix (or equivalently the

set of non coprime polynomials). In Section 5 we recall some extensions of the method to complex polynomials and constrained systems. Finally in Section 6 we test the performances of the algorithm and we discuss some implementation issues.

2 Preliminaries

We introduce now some general topics which we use in the following.

2.1 Generalized Sylvester matrices

Consider the set of l polynomials $p_1, \dots, p_l$. Without loss of generality we can assume that all the polynomials have the same degree, adding some zeros for the missing coefficients if necessary. We will see that the method allows to add constraints on the coefficients of the data polynomials, so that the added zero coefficients can stay unchanged in the output solution in order to preserve the starting degrees.

In the framework of polynomials coprimeness, it is usual to introduce the Sylvester matrix, which is defined as follows:

Definition 2.1 Given the l polynomials $p_1, \dots, p_l$, assume without loss of generality that they have the same degree n. We define the vector $p = \{p_1, \dots, p_l\} \in \mathbb{R}^{ln}$. The generalized Sylvester matrix [22] is the following $ln \times 2n$ matrix

$$S(p) = \begin{pmatrix} R_1 \\ \vdots \\ R_l \end{pmatrix} \tag{2.1}$$

where

$$R_i = \begin{pmatrix} p_{i,n} & p_{i,n-1} & \cdots & p_{i,0} & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & p_{i,n} & p_{i,n-1} & \cdots & p_{i,0} \end{pmatrix} \tag{2.2}$$

for $i = 1, \dots, l$. All the blocks are $n \times 2n$ matrices assuming that some of the leading coefficients are 0 when the degree of the corresponding polynomial is less than n.

The main property of the Sylvester matrix is summarized in the next theorem [29].

Theorem 2.1 *The degree of the greatest common divisor of the polynomials $p_1, \dots, p_l$ is equal to the rank defect of the Sylvester matrix $S(p)$ in (2.1).*

This result suggests us how to proceed. We look for some perturbation $\delta p \in \mathbb{R}^{ln}$ such that the matrix $S(p + \delta p)$ is rank-deficient with co-rank k.

Remark 2.1 A perturbation of the polynomials which makes the associated Sylvester matrix singular will create (at least) a common root of the l polynomials. If this root is complex, its conjugate will be a common root too, so the Sylvester matrix will have one more null singular value. This is the same situation which happens for two polynomials.

2.2 Structured matrices and their singular values

In Section 2.1 we stated that Problem 1.1 can be written as a (structured) matrix perturbation problem. Essentially, given a matrix with a special stucture (a generalized Sylvester matrix) we need to compute its (structured) distance to singularity. We will denote by $\mathcal{S}$ the set of (generalized) real Sylvester matrices with the extra zero pattern due to the different degrees. In a general framework, the datum of the problem is a matrix $S \in \mathcal{S}$ and we look for a singular matrix $B = S + X$, where $X \in \mathcal{S}$ and its norm is minimal (since the distance is the one defined in (1.1) the corresponding matrix norm is the Frobenius norm).

The singularity of a matrix can be checked through the computation of its singular values, and in particular it holds that B is rank-deficient with co-rank k if and only if 0 is a singular value of B with multiplicity k. Since the singular values are non negative real numbers we can focus on the smallest ones; we assume the singular values are increasing and we denote by $\sigma_i(B), i = 1, \ldots, k$ the i-th smallest singular value. A further formulation of the problem in this framework is

Problem 1.1' *Given S, compute*

$$\min_{\substack{X \in \mathcal{S} \\ \sigma_i(S+X)=0}} \|X\|_F \quad i = 1, \ldots, k \tag{2.3}$$

Furthermore we write $X = \epsilon E$ for a Sylvester matrix $E \in \mathcal{S}$ whose norm is 1. We introduce then the set of structured singular values

$$\Sigma_\epsilon^{\mathcal{S}}(S) = \{\sigma \in svd(S + \epsilon E) : E \in \mathcal{S},\ \|E\|_F = 1\} \tag{2.4}$$

We aim to find the minimal value of ϵ such that there exists a matrix $E \in \mathcal{S}$ of norm 1, with $S + \epsilon E$ rank-deficient with co-rank k..

Remark 2.2 The constraint $\|E\|_F = 1$ in (2.4) seems restrictive and it should be replaced by the inequality $\|E\|_F \leq 1$. However it can be shown (see [12, Lemma 4.4]) that there is no loss of generality in considering E a norm 1 matrix.

For given ε, the goal is to minimize all the k smallest singular values of the matrix $S + \epsilon E$, where $E \in \mathcal{S}$ has norm 1. Since the singular values are real positive numbers, the function we take into account is the singular value $\sigma_k(S + \epsilon E)$ (remeber we assumed the singular values are increasing, so if this function vanishes, all the k smallest singular values automatically go to zero); the basic idea to minimize this function is to compute its gradient and then to apply a descent method.

2.3 A 2-level algorithm

We started from Problem 1.1 and we arrived to the equation $\sigma_k(S + \epsilon E) = 0$ and to the search for the value

$$\epsilon^* = min\{\epsilon : \sigma_k(S + \epsilon E) = 0\}.$$

This is a global optimization problem. We recall the method proposed in [12] which uses a local approach.

Definition 2.2 A matrix E with $\|E\|_F = 1$ and such that $S+\epsilon E$ has a smallest singular value σ that locally minimizes $\Sigma_\epsilon^{\mathcal{S}}(S)$, is called a local extremizer. The value σ is called a local minimum point.

The algorithm proposed in [12] has two levels: inner and outer. At the inner level we fix the value of ϵ. We compute a (local) minimum point of $\Sigma_\epsilon^{\mathcal{S}}(S)$ and we denote it by $\sigma_k(\epsilon)$. The proposed algorithm finds local optima looking for the stationary points of a system of ODEs (the gradient system associated to the functional $\sigma_k(S + \epsilon E)$).

At the outer level, we have $\sigma(\epsilon)$ which is a continuous branch of local minima of $\Sigma_\epsilon^{\mathcal{S}}(S)$: we need to compute

$$\epsilon^* = \min\{\epsilon : \sigma_k(\epsilon) = 0\}.$$

This computation can be done by a root finding algorithm (e.g. Newton' s method) exploiting the knowledge of the exact expression of the derivative of the objective function. From the numerical point of view we don't need $\sigma_k(\epsilon)$ to be exactly 0, but it's enough that $\sigma_k(\epsilon) < \theta$ where θ is a small fixed tolerance.

3 Approximation of local minima of $\Sigma_\epsilon^{\mathcal{S}}$

In this section we consider the inner iteration of the algorithm, where the value of ϵ is fixed, and we want to minimize the singular value σ_k of the matrix function $B(t) = S + \epsilon E(t)$, with $E(t) \in \mathcal{S}$ of norm 1.

The goal is to find an optimal direction $Z = \dot{E}(t)$ such that the singular value σ_k of $B(t) = S+\varepsilon E(t)$ is characterized (locally) by the maximal possible decrease. In order to compute this direction we remember that the squares of the singular values of a matrix A are the eigenvalues of the symmetric and positive semidefinite matrix $A^T A$. We can exploit therefore the result in Lemma 3.1 about differentials of eigenvalues for real symmetric matrices [18].

Lemma 3.1 *Let $C(t)$ be a differentiable real symmetric matrix function for t in a neighborhood of 0, and let $\lambda(t)$ be an eigenvalue of $C(t)$ converging to a simple eigenvalue λ_0 of $C(0)$ as $t \to 0$. Let x_0 be a normalized eigenvector of $C(0)$ associated to λ_0. Then the function $\lambda(t)$ is differentiable near $t = 0$ with*

$$\dot{\lambda} = x_0^T \dot{C} x_0.$$

We assume that $E(t)$ is a smooth function, and we apply the result in Lemma 3.1 to the symmetric and positive definite matrix $B(t)^T B(t)$, where $B(t) = S + \epsilon E(t)$. If we denote by σ a generic singular value of B (in general we can assume σ is differentiable, see Remark 3.2), and by u and v the corresponding left and right singular vectors (omitting the dependance on t, and remembering that S is a constant matrix), we have

$$\begin{aligned} \frac{d}{dt}\sigma^2 &= v^T \frac{d}{dt}(B^T B)v = 2\epsilon\sigma u^T \dot{E} v \\ \frac{d}{dt}\sigma &= \epsilon u^T \dot{E} v. \end{aligned} \tag{3.1}$$

So we found that the optimal direction $Z = \dot{E}$ is obtained (up to a constant term) by minimizing the function $u^T \dot{E} v$. Moreover we observe that if $\dot{E} \in \mathcal{S}$, it holds that

$$u^T \dot{E} v = \langle uv^T, \dot{E} \rangle = \langle P_{\mathcal{S}}(uv^T), \dot{E} \rangle, \tag{3.2}$$

where $P_S(\cdot)$ denotes the orthogonal projection onto $\mathcal{S}$. The following lemma provides an explicit formula for the projection $P_{\mathcal{S}}$ (check [12, Lemma 4.2] for the proof).

Lemma 3.2 *Let $p_1, \ldots, p_l$ be a set of l polynomials of degree n, and $B \in \mathbb{R}^{ln \times 2n}$. The orthogonal projection (with respect to the Frobenius inner product $\langle \cdot, \cdot \rangle$) $P_{\mathcal{S}}(B)$ is given by*

$$P_{\mathcal{S}}(B) = Syl(a) \tag{3.3}$$

where $a = \{a^1, \ldots, a^l\} \in \mathbb{R}^{ln}$ and

$$a^i_{n-k} = \frac{1}{n} \sum_{j=1}^{n} B_{j+(i-1)n, j+k}, \quad k = 0, \ldots, n, \ i = 1, \ldots, l.$$

Now we can derive the important property $P_S(uv^T) \neq 0$ (where u and v are the left and right singular vectors of B associated to its singular value σ).

Lemma 3.3 *Let $S, E \in \mathcal{S}$, with E of unit Frobenius norm, and $\epsilon > 0$. Set $B = S + \epsilon E$. If $\sigma > 0$ is a simple singular value of B and u and v are the corresponding left and right singular vectors, then*

$$P_S(uv^T) \neq 0 \tag{3.4}$$

Proof By assumption v is the eigenvector of $B^T B$ associated to σ^2. Assume, by contradiction, that $P_S(uv^T) = 0$. It holds true that

$$\begin{aligned} 0 = \langle P_S(uv^T), B \rangle &= \langle uv^T, B \rangle = \frac{1}{\sigma} \langle Bvv^T, S + \epsilon E \rangle \\ &= \frac{1}{\sigma} v^T (S + \epsilon E)^T (S + \epsilon E) v = v^T \sigma v = \sigma \|v\|^2 > 0 \end{aligned} \tag{3.5}$$

since v is a non-zero vector. Consequently (3.5) is false, and the claim follows.

Remark 3.1 Notice that Lemma 3.3 and the results in this last section are valid for all the singular values σ: this is important since we are taking into account the computation of an AGCD of fixed degree $k \geq 1$.

3.1 Optimal direction

Let σ_k be the k-th smallest singular value of the matrix $B = S + \epsilon E$, and u_k, v_k be the corresponding left and right singular vectors, respectively. The steepest descent direction for σ_k is given by

$$\begin{aligned} Z_* = arg \min_{\substack{\dot{E}\in\mathcal{S} \\ \|\dot{E}\|_F=1}} (u_k)^T \dot{E} v_k \\ \text{subject to } \langle E, \dot{E}\rangle = 0. \end{aligned} \tag{3.6}$$

where the constraint on the norm guarantees the uniqueness of the solution (since we look for a direction). We give the solution of the problem (3.6) in the following lemma, whose proof is similar to that given in [12, Section 4.2].

Lemma 3.4 *Let $E \in \mathcal{S}$ of unit Frobenius norm, and u_k, v_k non-zero real vectors. If $B = S + \epsilon E$ and E has full (column) rank and is not proportional to $P_S(u_k v_k^T)$, then the solution of the optimization problem* (3.6) *is given by*

$$\nu Z_* = -P_S(u_k v_k^T) + \langle E, P_S(u_k v_k^T)\rangle E \tag{3.7}$$

where ν is the norm of the matrix in the right hand side of (3.7).

Optimality conditions. The Karush-Kuhn-Tucker conditions applied to the optimization problem

$$\min_{\substack{\sigma_k\in\Sigma(S+\epsilon E) \\ E\in S,\ \|E\|_F=1}} \sigma_k,$$

in the case when the minimum is not zero, give:

$$\begin{aligned} P_S(u_k v_k^T) = sE_* \quad s < 0 \\ \|E_*\|_F = 1 \end{aligned} \tag{3.8}$$

where E_* is an extremizer. Hence $s < 0$ characterizes a minimum point.

Lemma 3.4 suggests to consider the differential equation

$$\dot{E} = -P_S(u_k v_k^T) + \langle E, P_S(u_k v_k^T)\rangle E \tag{3.9}$$

where $B = S + \epsilon E$ for a certain fixed value ϵ, and u_k, v_k are the left and right singular vectors of the matrix B associated to its singular value σ_k. It is possible to prove now that the function $\sigma_k(t)$ actually decreases along the solutions of (3.9).

Theorem 3.1 *Let $E(t) \in \mathcal{S}$ of unit Frobenius norm which satisfy the equation* (3.9). *If $\sigma_k(t)$ is a singular value of $B = S + \epsilon E(t)$, then*

$$\frac{d}{dt}\sigma_k(t) \leq 0 \tag{3.10}$$

Proof In the proof we omit the index k since the result is actually true for any singular value and the associated singular vectors. Recall that the expression of $\dot{\sigma}$ is given in (3.1) by $u^T \dot{E} x$ (ignoring the multiplication by constant factors). Replace $\dot{E}$ using the equation (3.9) and estimate the two terms.

$$u^T P_S(uv^T)v = \langle uv^T, P_S(uv^T)\rangle = \langle P_S(uv^T), P_S(uv^T)\rangle = \\ = \|P_S(uv^T)\|_F^2.$$

and

$$\langle E, P_S(uv^T)\rangle u^T E v = \langle E, P_S(uv^T)\rangle \langle E, uv^T\rangle = \\ = \langle E, P_S(uv^T)\rangle^2$$

Since E has norm 1, by the Cauchy-Schwartz inequality we have

$$u^T \dot{E} v = (-\|P_S(uv^T)\|_F^2 + \langle E, P_S(uv^T)\rangle^2) \leq 0. \tag{3.11}$$

Remark 3.2 We could have a problem with the differentiability of σ_k when the singular values σ_k and σ_{k-1} coalesce. From a theoretical point of view this problem can be solved allowing sign changes in the SVD factorization [8]. However, from the numerical point of view, we don't observe this coalescence generically.

Since the goal is to minimize σ_k, we look for the stationary points of the equation (3.9). Their characterization is given in the following theorem, whose proof is a consequence of the previous results.

Theorem 3.2 *Consider a solution of* (3.9) *and assume* $\sigma_k \neq 0$. *The following statements are equivalent:*

1. $\dot{\sigma}_k = 0$;
2. $\dot{E} = 0$;
3. E *is a multiple of* $P_S(u_k v_k^T)$.

Remark 3.3 There is no assurance that the computed values are global minima, although this seems to be the case for experiments of small dimension, where the solution computed by the proposed method is equal to or better than the one found by other algorithms.

3.2 The system of ODE

We consider now the algorithm to compute the stationary points of the system of ODEs. The scheme for the numerical integration of the equation and the computation of the quantities of interest is similar to the one presented in [12] with the replacement of eigenvalues by singular values. We summarize it in Algorithm 1.

The choice of the explicit Euler method is due to the expensive function evaluation (the computation of a SVD decomposition). This is the part of the algorithm which requires more computations (and consequently more time).

Algorithm 1: Numerical solution of the ODE (3.9) at the step j

Data: $p_1^j, \dots, p_l^j$ (or equivalently E^j), σ_k^j, u^j, v^j, h^j (step Euler method) and γ (step size reduction), ϵ.
Result: E^{j+1}, σ_k^{j+1}, u^{j+1}, v^{j+1} and $\bar{h}^{j+1}$
begin
1 Set $h = h^j$
2 Compute $\dot{E}^j = -P_S(u^j v^{jT}) + \langle E^j, P_S(u^j v^{jT})\rangle E^j$
3 Euler step $\rightarrow E^{j+1} = E^j + h\dot{E}^j$
4 Normalize E^{j+1} dividing it by its Frobenius norm
5 Compute the singular value σ_k of the matrix $B^{j+1} = S + \epsilon E^{j+1}$
6 Compute the singular vectors u and v of the matrix B^{j+1} associated to σ_k
7 **if** $\sigma_k > \sigma_k^j$ **then**
reject the result and reduce the step h by a factor γ
repeat from 3
else
accept the result; set $h^{j+1} = h$, $\sigma_k^{j+1} = \sigma_k$, $u^{j+1} = u$, $v^{j+1} = v$
8 **if** $\sigma_k^{j+1} - \sigma_k^j < tol$ ***or*** $\sigma_k^{j+1} \leq tol$ **then**
return
9 **if** $h^{j+1} = h^j$ **then**
increase the step size of γ, $h^{j+1} = \gamma h^j$
else
set $h^{j+1} = h^j$
10 Go to the next iteration

The steps of the algorithm which deserve attention are 5 and 6. We need to compute a singular value of the matrix B and the corresponding singular vectors. This can be done in two different ways:

1. compute the SVD decomposition of B (using the Matlab function *svd*);
2. compute the eigenvalue decomposition of B^TB (using the Matlab function *eig*) in order to obtain v and σ^2 and than compute $\sigma = \sqrt{\sigma^2}$ and $u = Bv/\sigma$.

The SVD computation is more stable than the computation (possibly ill conditioned) of the eigenvalue decomposition of B^TB and the square root of its eigenvalues. We will use the SVD decomposition in the experiments just to have a reference code, but (numerically) it's not always the faster method, probably because the eigenvectors of B^TB and the right singular vectors of B (computed by Matlab) are not exactly the same.

For what concern the step size control we are not able to find an optimal value or a strategy for its computation. After some investigation we found some examples where the reduction or the increasing of the step size parameter (the factor γ) leads to a remarkable improvement in terms of computational time, but this depends on the particular problem so it seems there is no a general rule.

4 Approximation of the distance

We are able to compute the minima of Σ_ϵ^S. The next step is to analyze the outer iteration of the algorithm, i.e. the optimization on ϵ. We will use the notation

$$\sigma_k(\epsilon) = \min_{\sigma_k \in \Sigma_\epsilon^S} \sigma_k$$

to denote a smooth branch of local minima parametrized by ϵ and computed through the solutions of the ODE (3.9).

In order to solve the optimization problem we can exploit the knowledge of the derivative of $\sigma_k(\epsilon)$ with respect to ϵ. The expression for this derivative is in the following theorem (whose proof can be obtained in a similar way to the one given in [12, Theorem 5.1]).

Theorem 4.1 *Assume that*

1. $\epsilon \in (0, \epsilon^*)$ *such that* $\sigma_k(\epsilon) \neq 0$ *for a simple singular value* σ_k*;*
2. $\sigma_k(\epsilon)$ *and* $E(\epsilon)$ *are smooth w.r.t.* ϵ *and let* $u_k(\epsilon), v_k(\epsilon)$ *be the singular vectors of* $B(\epsilon)$ *corresponding to* $\sigma_k(\epsilon)$*, where* $B(\epsilon) = S + \epsilon E(\epsilon)$*.*

Then it holds

$$\frac{d}{d\epsilon}\sigma_k(\epsilon) = -\left\| P_S(u_k(\epsilon)v_k(\epsilon)^T) \right\|_F. \tag{4.1}$$

The formula in (4.1) is useful in the computation of the optimal value for ϵ^*. We underline that the derivative is negative, since the projection doesn't vanish.

The current problem is

$$\epsilon^* = \min\{\epsilon : \sigma_k(\epsilon) = 0\}.$$

It can be seen as a root finding problem; we approach it by the Newton's method (since we know the expression for the derivative (4.1)). The function $\epsilon \to \sigma(\epsilon)$ is smooth, so we can apply a Newton' s iteration in order to find its zero (we stop when $\sigma_k(\epsilon) \leq \theta$, for a given tolerance θ). The formula is given by

$$\epsilon_{t+1} = \epsilon_t + \frac{\sigma_k(\epsilon_t)}{\|P_S(u_k(\epsilon_t)v_k(\epsilon_t)^T)\|_F}, \tag{4.2}$$

where $\sigma_k(\epsilon_t)$ is a singular value of the matrix $S + \epsilon_t E(\epsilon_t)$. and $E(\epsilon_t)$ is the extremizer obtained by integrating the ODE.

We now consider the algorithm for the computation of ϵ^*, analogous to the one presented in [12] (Algorithm 2), which uses a combined Newton-bisection step. When the value $\sigma(\epsilon_{t+1})$ is bigger than a fixed tolerance we update the value of ϵ, otherwise ϵ_{t+1} is computed by a bisection step. This prevents quadratic convergence when the algorithm takes several bisection steps. The upper bound for the value of ϵ is the norm of the starting Sylvester

matrix S while the lower bound is the smallest singular value of the matrix S, i.e. the unconstrained distance to singularity.

Algorithm 2: Computation of ϵ^*

Data: *tol* and $\epsilon_{\min}$. $\epsilon_{\max}$ lower and upper bounds, ϵ_t current value
Result: ϵ_f approximation of ϵ^*
begin
Set Bisect = False, $t = 1$
while $|\epsilon_t - \epsilon_{\max}| \geq tol$ **do**
  if *Bisect = False* **then**
    Store ϵ_t and $\sigma(\epsilon_t)$
    Compute $\tilde{\epsilon}_{t+1}$ by a Newton iteration
    if $\tilde{\epsilon}_{t+1} \notin (\epsilon_{\min}, \epsilon_{\max})$ **then**
      Set $\tilde{\epsilon}_{t+1} = (\epsilon_{\max} + \epsilon_t)/2$
  else
    Set $\tilde{\epsilon}_{t+1} = (\epsilon_{\max} + \epsilon_t)/2$

  Compute $\sigma(\tilde{\epsilon}_{t+1})$ by integrating the ODE
  if $\sigma(\tilde{\epsilon}_{t+1}) < tol$ **then**
    Set Bisect = True
    Set $\epsilon_{\max} = \tilde{\epsilon}_{t+1}$
  else
    Set Bisect = False
    Set $\epsilon_{t+1} = \tilde{\epsilon}_{t+1}$
    Set $t = t + 1$
Set $\epsilon_f = \epsilon_t$

5 Extensions of the algorithm

The proposed algorithm can naturally be extended to a bigger class of related problems (constrained systems, complex polynomials). They were just considered in [12] in the case of one common root between two polynomials, so we recall them briefly since they can be naturally adapted to the computation of k common roots for several polynomials.

5.1 Constrained systems

Suppose that only certain coefficients of the polynomials are allowed to be perturbed. Then the method has the same structure and we have only to pay attention to the projection operator. Indeed, if we call $\mathcal{I}^i$ the set of coefficients of the i-th polynomial that can be perturbed, we have to consider in Lemma

3.2 only the coefficients whose index is in $\mathcal{I}^i$.

$$for \;\; i \;=\; 1:l$$
$$a^i_{n-k} = \begin{cases} \frac{1}{n}\sum_{j=1}^{n} B_{j+(i-1)n,j+k} & k \in \mathcal{I}^i \\ 0 & k \notin \mathcal{I}^i \end{cases}$$

We remark that in this case we are not able to characterize the stationary points as in Theorem 3.2, except for keeping a polynomial monic [12, Lemma 4.3]. But the numerical experiments ensure that the result is correct.

5.2 Complex polynomials

If the starting polynomials have complex coefficients we can proceed exactly in the same way. In this case we have to consider the manifold of complex Sylvester matrices with the extra zero pattern (so the projection operator will change in the appropriate way). We recall that for real polynomials we could find a complex common root, and consequently an AGCD of degree higher than expected; this situation cannot happen if we work in the framework of complex coefficients.

6 Numerical experiments

In this section we check the performance of the proposed methodology. The main tasks are the following:

- make a comparison with the similar algorithm describing the dynamic on eigenvalues [12] (implemented for one common root between two polynomials);
- make a comparison with other existing algorithms which solve the same problem; mainly the SLRA toolbox [28], the *uvGCD* function of the Numerical Algebraic Computing Toolbox (NAClab) [31] and the subspace method [24];
- sum up the results and make some comments about the algorithm and its optimization

We recall that the available algorithms for *uvGCD* and SLRA are implemented only for two polynomials. The SLRA methodology has some limitations imposed by its solution method [27,10], and the algorithm requires a suitable initial approximation to compute an accurate solution. The subspace method needs to start "close" to the solution in order to converge to the sought local minimum; moreover it makes a bigger error on the coefficients of the computed solution. On the other side, the proposed method doesn't have any restriction and allows to include constraints on the coefficients; furthermore it seems to be more robust to the initial data, as it is able to compute an accurate solution independently on the starting polynomials.

6.1 Compariosons with other methods

We take into account some examples in order to test the performance of the proposed algorithm.

6.1.1 Example (AGCD of degree 1 with a constraint)

We consider the following two polynomials of degree 5, expressed by the vector of their coefficients:

$$\begin{aligned} p_1 &= (1, 0, 1, 0, 2, 1) \\ p_2 &= (-2, 1, 1, -1, 0, 1). \end{aligned} \tag{6.1}$$

We want to preserve the property that p_1 is monic. Since we need to add the constraint, we compare in this first example only the two ODE based methods, which consider the dynamic on eigenvalues and singular values.

The perturbed polynomials found by the two algorithms coincide up to the third decimal digit, and are

$$\begin{aligned} \hat{p}_1 &= (1, 0.014, 0.972, 0.051, 1.903, 1.181) \\ \hat{p}_2 &= (-1.977, 0.958, 1.078, -1.148, 0.279, 0.473), \end{aligned}$$

consequently also the common root and the computed distance are about the same. The difference is in the computational time, which is $0,171$ seconds considering the singular values and 2.328 seconds considering the eigenvalues, so the algorithm with singular values is about ten times faster!

6.1.2 Example (a complex common root)

We consider a second example from [12]; this time we consider also the comparison with SLRA and *uvGCD*. The datum is two polynomials of degree 3 whose coefficients are

$$\begin{aligned} p_1 &= (1, 2, 2, 2) \\ p_2 &= (2, 0, 1, -2). \end{aligned} \tag{6.2}$$

First consider the fully unconstrained case. We focus the attention mainly on the computed distance and the computational time; the results for the computation of an AGCD of degree 1 are in Table 6.1.

AGCD of degree 1

	SLRA	*uvGCD*	ODE-eig	ODE-svd
time	0.29 s	0.05 s	39.22 s	1.684 s
distance	2.1054	3.4039	0.3568	0.3568

Table 6.1: *Computation of the AGCD of degree* 1 *for the pair of polynomials in* (6.2)*. We compare the computational time (first row) and the distance (second row)*

It happens that SLRA and *uvGCD* compute one real common root, while the ODE methods find a closest complex common root: consequently the distances are different. Indeed if we look at the result of the computation of two common roots (Table 6.2), all the algorithms find the same value of the distance, but this means that SLRA and *uvGCD* need this information a priori. As expected, the approximation found by the proposed algorithm in the two cases is exactly the same.

AGCD of degree 2

	SLRA	*uvGCD*	ODE-svd
time	0.515 s	0.25 s	1.154 s
distance	0.3568	0.3568	0.3568

Table 6.2: *Performance of* 3 *different algorithms for computing the AGCD of degree* 2 *for the pair of polynomials in* (6.2)*. We compare the computational time (first row) and the distance (second row)*

After these examples we noticed that the dynamic on singular values turns out to be always faster than the one on eigenvalues, giving the same solution. Looking at the comparison with other methods (SLRA and *uvGCD*) it seems we are now able to achieve reasonable computational times.

6.1.3 Example (high degree polynomials)

We consider now two high degree polynomials whose roots are on two different circles in the complex plane. We see how the proposed method is still able to find an accurate solution, unlike it happens for the other algorithms. The two polynomials are

$$\begin{aligned} p_1(z) &= z^{15} + 1; \\ p_2(z) &= z^{15} + 3; \end{aligned} \tag{6.3}$$

We report the results of the computation of one closest common root between p_1 and p_2 in Table 6.3.

	SLRA	*uvGCD*	ODE-eig	ODE-svd
time	0.18 s	0.08 s	136.38 s	5.83 s
distance	0.5857	3.4503	0.3197	0.3201

Table 6.3: *Performance of* 4 *different algorithms for computing the AGCD of degree* 1 *for the pair of polynomials in* (6.3)*. We compare the computational time (first row) and the distance (second row)*

The distance computed by the ODE methods is the best one. The other two methods find different local minima of the objective function. In the case of the dynamic on eigenvalues the solution it is even better, but we notice that the computational time becomes huge! This example shows how the proposed method works better when the global optimization problem is more difficult. However we can only state that the proposed method computes a better approximation, but we are not sure it matches the global minimum.

6.1.4 Example: polynomials of increasing degree

In the last examples we noticed how different methods can converge to different local minima. However we dealt with matrices of small dimension, so the computational times could be misleading. In the following experiment we want to make a computation with matrices of increasing dimension in order to have more clear ideas about the performances. The problem is to find one common root between the following polynomials:

$$\begin{aligned} p_1 &= (1, zeros(1, 10n), ones(1, 10n), 5) \\ p_2 &= (1, ones(1, 10n), zeros(1, 10n), 1) \end{aligned} \tag{6.4}$$

for $n = 1, \dots, 10$. The results are in Table 6.4: we will notice how the proposed method computes an accurate solution in a reasonable computational time.

n		SLRA	*uvGCD*	ODE-svd
1	time	1.397 s	0.053 s	2.212 s
	distance	1.0182	2.6354	0.0352
2	time	2.535 s	0.047 s	2.646 s
	distance	1.0109	1.4747	0.0166
3	time	2.637 s	0.065 s	3.140 s
	distance	1.1984	0.1689	0.0124
4	time	8.586 s	0.068 s	3.780 s
	distance	1.0453	1.3561	0.0106
5	time	14.233 s	0.116 s	4.782 s
	distance	1.0550	1.2847	0.0095
6	time	20.952 s	0.140 s	6.726 s
	distance	1.0809	1.3967	0.0088
7	time	34.345 s	0.172 s	8.008 s
	distance	1.0938	0.0843	0.0082
8	time	42.678 s	0.194 s	9.920 s
	distance	1.1439	0.0891	0.0078
9	time	68.597 s	0.289 s	11.573 s
	distance	1.1580	0.0743	0.0074
10	time	96.992 s	0.243 s	14.877 s
	distance	1.1721	1.1493	0.0071

Table 6.4: *Performances of the different algorithms for polynomials of increasing degrees. We compute a common factor of degree* 1 *for the pair of polynomials in* (6.4).

We excluded from the comparison ODE-eig because it is too slow, and we notice that the distance computed by *uvGCD* is at least ten times larger than the one computed by the ODE method. Table 6.4 shows that SLRA is faster than ODE for small dimensions, but it becomes too slow for higher degree polynomials (and moreover it converges to a different local minimum corresponding to a larger distance). In all these examples the ODE method always does 2 iterations, so we can have an idea on how the computational time per iteration increases with the dimension of the problem. However it is quite difficult to check the complexity of the algorithm, since each iteration has a different computational cost.

6.2 Performance for different perturbation levels

We make now some comparisons with the *subspace method* proposed in [24]. This algorithm converges to a local minimum, so we need to choose the starting polynomials "close" to the sought solution. The idea is to start from a set of polynomials which have an exact GCD, and then to add some random perturbations in order to check how the two algorithms behave depending on the level of noise.

We do not dwell on the numerical values, but we briefly explain how we build the data and then we observe how the average error on the perturbed polynomials vary depending on the level of noise. The data are built in the following way:

- fix the number of polynomials, their degrees and their GCD
- multiply the GCD by some rundom polynomials to obtain a set p of polynomials which have an exact GCD
- the data come from the following perturbations: $\tilde{p_i} = p_i + s\|p_i\|r_i$ where s is the noise level and r_i is a random vector of norm 1

We vary the level of noise uniformly between 0 and 1; for each level of noise we run several experiments and we plot the average error (the distance divided by the norm of the data).

In the first example we want to compute one common root starting from three polynomials of degree eight.

In the second example we compute an AGCD of degree 2 of four polynomials of degree 8.

We notice that the error on the solution computed by the ODE algorithm is (on average) the smallest, independently of the level of noise. The advantage of the subspace method is the computational time. However it converges only to a local minimum, so if we start from an arbitrary set of polynomials it is not able to compute a good approximation, unlike it happens for the ODE method. Moreover we remark that several runs of the ODE algorithm are very fast (but this is not a general fact), so in this cases we have a method which is fast and accurate at the same time.

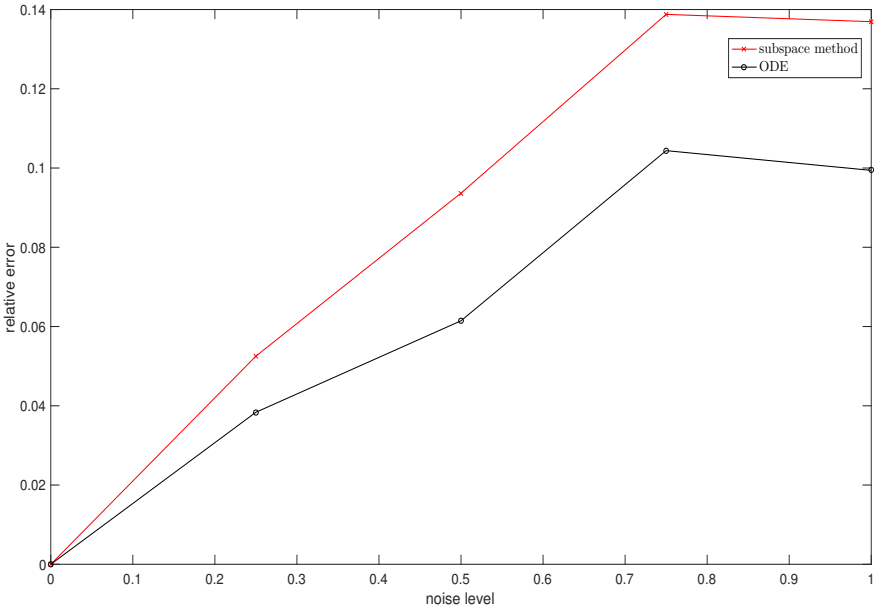

Fig. 6.1: *Computation of an AGCD of degree 1 for three polynomials: distribution of the average error depending on an increasing level of noise*

6.3 Final remarks and implementation issues

If we sum up briefly the results of the numerical experiments, we improved the algorithm proposed in [12] to the computation of an approximate gratest common divisor of degree k of several polynomials[1]. We did it taking into account the generalized Sylvester matrix associated to the data polynomials and considering the dynamic on the singular values, since the matrices are rectangular; although we noticed that in the case of two polynomials, when the Sylvester matrix is square, it's worth to consider the singular values instead of the eigenvalues since the algorithm runs faster. The singular values allow to work with rectangular matrices, as it happens for more than 2 polynomials, and the computation of an AGCD of degree $k \geq 1$ is simplified since the singular values are non negative real numbers. We noticed how the proposed method is able to compute better approximations for some problems where other algorithms converge to different local minima, and it appears to be efficient when the dimensions of the matrices increase considerably.

In the following we make some comments on the efficiency of the whole algorithm, which is briefly described in Algorithm 3 in the simple case of two polynomials p and q.

[1] The Matlab code is available upon request to the first author

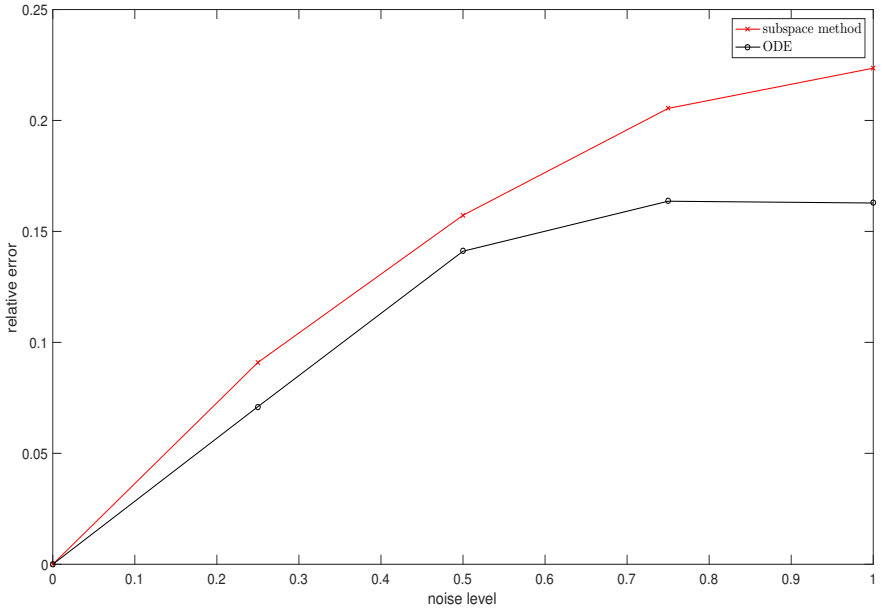

Fig. 6.2: *Computation of an AGCD of degree 2 for four polynomials: distribution of the average error depending on an increasing level of noise*

We notice that the algorithm contains several parameters:

- the value th is a tolerance on the computed singular value which determines if computing a new value of ϵ and consequently checking the convergence condition;
- $itmax$, the maximum number of iterations;
- c, which determines the weights of $\epsilon_{\min}$ and $\epsilon_{\max}$ in the bisection steps;
- γ, which doesn't appear explicitly in Algorithm 3, but is the step reduction in the Euler's method.

All these parameters can influence the speed of convergence, but it's too difficult to find an optimal value for each of them. This is because they depend on the particular problem, so apparently there is no a general technique which allows to optimize the set of all the parameters. Anyway we can test how their change can modify the performances.

In the numerical experiments of the previous section we fixed the values of the parameters; we list here some examples to understand how the performances can be influenced:

- changing the value of th can heavily modify the computational time (but be careful, because if the value is too high also the distance can increase considerably);

Algorithm 3: Scheme of the main algorithm

```
Data: p, q and tol (stopping tolerance)
Result: E and σ.
From the data we compute ε_min, ε_max
ε_0 = min{2ε_0, .5(ε_min + ε_max)}
c = 0.5
itmax = 40
begin
1   integrate the ODE → compute σ_k
2   for it = 1 : itmax do
3     if σ_k > th then
        Compute ε_1 by a Newton iteration
4       if ε_min < ε_1 < ε_max then
5         if |ε_1 − ε_0| < tol then
            ε_0 = ε_1
            break
          else
            ε_0 = ε_1
        else
          ε_0 = (1 − c)ε_max + c ε_min
      else
        bisection step, ε_max = ε_0
        ε_0 = (1 − c)ε_max + c ε_min
6     integrate the ODE → compute σ_k
```

– modifying the step reduction γ can speed up the algorithm (we found some examples where the code runs 20 times faster!).

The function which dominates the computational time of the algorithm is the integration of the ODE. The condition $\sigma_k > th$ is the crucial point of the algorithm, as suggested before. Changing the value of th determines if the inequality is verified (and consequently enter the cycle, update ϵ, and check the convergence condition) or not (in this last case, skip the cycle and integrate one more time the ODE without checking the convergence condition). And more integrations of the ODE mean more computational time. On the other side, changing the value of γ can avoid some integration of the ODE since we can reach the minimum faster modifying the step-length of the Euler's method.

The problem is that it's very hard to optimize all the parameters; probably we could do some statistics to determine which values are the best in the most of the cases, even if this can become computationally expensive and it would be only a statistic, not a proof. Maybe we could try to do some guesses if we know the solution in advance; we only tried to change some values randomly and observe if there is any improvement.

Finally, in all the experiments we always used the same algorithm, which uses the SVD decomposition of the matrix B in the integration of the ODE; but sometimes the code runs faster replacing the Matlab function *svd(B)* by *eig(B′ ∗ B)*. Hence not all the results shown in the previous sections are the optimal ones, but they are good enough to support the performance of the proposed method.

6.4 Outlook

The future work will focus on the applications of the method in system and control theory. However in many problems and practical applications we deal with multivariable control theory and consequently polynomial matrices, i.e. matrices whose elements are polynomials in an indeterminate. These matrices can be equivalently represented by polynomials with matrix coefficients. Consequently, a challenging task will be to adapt the method to polynomial matrices, in order to model more complicated systems.

Acknowledgements Nicola Guglielmi thanks INdAM GNCS for financial support. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant agreement number 258581 "Structured low-rank approximation: Theory, algorithms, and applications" and fund for Scientific Research (FWO-Vlaanderen), FWO projects G028015N "Decoupling multivariate polynomials in nonlinear system identification" and G090117N "Block-oriented nonlinear identification using Volterra series"; and the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme initiated by the Belgian Science Policy Office.

References

1. Beckermann, B., Labahn, G.: When are two numerical polynomials relatively prime?. J. Symb. Comput. 26 , 677–689 (1998)
2. Bini, D., Boito, P.: A fast algorithm for approximate polynomial gcd based on structured matrix computations. In: Numerical Methods for Structured Matrices and Applications, pp. 155–173. Birkäuser, Basel, 2010
3. Bini, D., Boito, P.: Structured matrix-based methods for polynomials ϵ-GCD: analysis and comparisons. In: ISSAC 2007, ACM, pp- 9–16. New York, USA, 2007
4. Boito P.: Structured matrix based methods for approximate polynomial GCD. Edizioni della Normale, Pisa (2011)
5. Chèze, G., Galligo, A., Mourrain, B., Yacoubsohn, J.: A subdivision method for computing nearest gcd with certification. Theoretical Computer Science 412, 4493–4503 (2011)
6. Chin, P., Corless, R. M., Corliss, G. F.: Optimization strategies for the approximate gcd problem. In: Proceedings of ISSAAC'98, ACM, pp. 228–235. New York, USA, 1998
7. Corless, R. M., Gianni, P. M., Trager, B. M., Watt, S. M.: The singular value decomposition for polynomial systems. In: Proceedings of ISSAAC'95, ACM, pp. 195–207. New York, USA, 1995
8. Dieci, L., Eirola, T.: On smooth decompositions of matrices. SIAM J. Matrix. Anal. and Appl. 20, 800–819 (1999)
9. Duan, X., Zhang, X., Wang, Q.: Computing Approximation GCD of Several Polynomials by Structured Total Least Norm. ALAMT 3, 39–46 (2013)
10. Golub, G., Pereyra, V.: Separable nonlinear least squares: the variable projection method and its applications. Inverse problems 19, 1–26 (2003)
11. Golub, G., Pereyra, V.: The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. SIAM Journal on Numerical Analysis 10, 413–432 (1973)
12. Guglielmi, N., Markovsky, I.: An ODE based method for computing the distance of coprime polynomials. SIAM J. Numer. Anal. 55, 1456–1482 (2017)
13. Kaltofen, E., Yang, Z., Zhi, L.: Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In: Proceedings of ISSAAC'06, ACM, pp. 169–176. New York, USA, 2006

14. Karcanias, N., Fatouros, S., Mitrouli, M., Halikias, G.: Approximate greatest common divisor of many polynomials and generalised resultants. Comput. Math. Appl. 51, 1817–1830 (2006)
15. Karmarkar, N. K., Lakshman, Y. N.: On approximate GCDs of univariate polynomials. J. Symbolic Comp. 26, 653–666 (1998)
16. Li, B., Liu, Z., Zhi, L: A fast algorithm for solving the sylvester structured total least squares problem. Signal Processing 87, 2313–2319 (2007)
17. Li, B., Nie, J., Zhi, L.: Approximate gcds of polynomials and sparse sos relaxations. Theoretical Computer Science 409, 200–210 (2008)
18. Magnus, J. R.: On differentiating eigenvalues and eigenvectors. Econometric theory 1, 179–191 (1985)
19. Markovsky, I.: Structured low-rank approximation and its applications. Automatica 44, 891–909 (2008)
20. Markovsky, I., Usevich, K.: Software for weighted structured low-rank approximation. Comput. Appl. Math. 256, 278–292 (2014)
21. Markovsky, I., Van Huffel, S: An algorithm for approximate common divisor computation. In: Proceedings of MTNS'06, pp. 274–279. Kyoto, Japan, 2006
22. Maroulas, J., Dascalopoulos, D.: Applications of the generalized Sylvester matrix. Appl. Math. Comput. 8, 121–135 (1981)
23. Pan, V. Y.: Computation of approximate polynomial GCDs and an extension. Information and Computation 167, 71–85 (2001)
24. Qiu, W., Hua, Y., Abed-Meraim, K.: A subspace method for the computation of the GCD of polynomials. Automatica 33, 741–743 (1997)
25. Schost, E., Spaenlehauer, P.-J.: A quadratically convergent algorithm for structured low-rank approximation. Found. Comput. Math. 16, 457–492 (2016)
26. Terui, A.: GPGCD: An iterative method for calculating approximate GCD of univariate polynomials. Theoretical Computer Science 479, 127–149 (2013)
27. Usevich, K., Markovsky, I.: Variable projection for affinely structured low-rank approximation in weighted 2-norms. J. Comput. Appl. Math. 272, 430–448 (2014)
28. Usevich, K., Markovsky, I.: Variable projection methods for approximate (greatest) common divisor computations. Theoretical Computer Science 681, 176–198 (2017)
29. Vardulakis, A. I. G., Stoyle, P. N. R.: Generalized Resultant Theorem. J. Inst. Maths. Applics. 22, 331–335 (1978)
30. Zarowsky, C. J., Xiaoyan Ma, Fairman, F. W.: QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. IEEE Trans Signal Processing 48, 3042–3051 (2000)
31. Zeng, Z.: The approximate GCD of inexact polynomials, Part 1: A univariate algorithm. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.645&rep=rep1&type=pdf` (2004)
32. Zeng, Z., The numerical greatest common divisor of univariate polynomials. Contemporary Mathematics 556, 187–217 (2011)