Structured Matrix Based Methods for Polynomial ϵ-gcd: Analysis and Comparisons

Dario A. Bini
bini@dm.unipi.it

Paola Boito
boito@mail.dm.unipi.it

Dipartimento di Matematica
Università di Pisa
Pisa, Italy

ABSTRACT

The relationship between univariate polynomial ϵ-gcd and factorization of resultant matrices is investigated and several stable and effective algorithms for the computation of an ϵ-gcd are proposed. The main result is the design of a practically stable algorithm whose arithmetic cost is quadratic in the degrees of the input polynomials. The algorithm relies on the displacement structure properties of Sylvester and Bézout matrices. Its effectiveness is confirmed by numerical experiments.

Categories and Subject Descriptors

G.1.3 [**Numerical Linear Algebra**]: Sparse, structured, and very large systems (direct and iterative methods); I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms

General Terms

Algorithms, experimentation

Keywords

Cauchy matrices, polynomial gcd, displacement structure, Sylvester matrix, Bézout matrix.

1. INTRODUCTION

The classical algebraic notion of polynomial gcd is known to be ill-suited to work in a numerical/applicative setting, where input data are represented as floating point numbers or derive from the results of physical experiments or previous computations, so that they are generally affected by errors. Indeed, if $u(x)$ and $v(x)$ have a nontrivial gcd, it turns out that arbitrarily small perturbations in the coefficients of $u(x)$ and $v(x)$ may transform $u(x)$ and $v(x)$ into relatively prime polynomials.

It is therefore necessary to introduce the notion of polynomial ϵ-gcd (or approximate gcd); for more details we refer the reader to [19] [7],[17], [22] and to the references therein.

Throughout this paper we use the following definition, where $\|\cdot\|$ denotes the Euclidean norm.

Definition 1.1. *A polynomial $g(x)$ is said to be an ϵ-divisor of $u(x)$ and $v(x)$ if there exist polynomials $\hat{u}(x)$ and $\hat{v}(x)$ of degree n and m, respectively, such that $\|u(x) - \hat{u}(x)\| \le \epsilon\|u(x)\|$, $\|v(x) - \hat{v}(x)\| \le \epsilon\|v(x)\|$ and $g(x)$ divides $\hat{u}(x)$ and $\hat{v}(x)$. If $g(x)$ is an ϵ-divisor of maximum degree of $u(x)$ and $v(x)$, then it is called* ϵ-gcd *of $u(x)$ and $v(x)$. The polynomials $p(x) = \hat{u}(x)/g(x)$ and $q(x) = \hat{v}(x)/g(x)$ are called ϵ-cofactors.*

Notice that, while the degree of an ϵ-gcd is uniquely defined, its coefficients are not.

The problem of ϵ-gcd computation can be stated as follows: given the coefficients of two univariate polynomials $u(x) = \sum_{i=0}^{n} u_i x^i$ and $v(x) = \sum_{i=0}^{m} v_i x^i$, compute the coefficients of an ϵ-gcd $g(x)$ of $u(x)$ and $v(x)$.

Several algorithms for the solution of this problem or its variants can be found in the literature; they rely on different techniques, such as the Euclidean algorithm [2],[1],[11],[16], optimization methods [14], SVD and factorization of resultant matrices [4], [5], [22], Padé approximation [3], [17], root grouping [17]. Some of them have been implemented inside numerical/symbolic packages like the algorithm of Zeng [22] in Matlab$^{\text{TM}}$ and the algorithms of Kaltofen [13], of Corless et al [5], of Labahn and Beckermann [12] in Maple$^{\text{TM}}$. These algorithms have a computational cost of $O(n^3)$ which makes them expensive for moderately large values of n. Algorithms based on the Euclidean scheme have a typical cost of $O(n^2)$ but they are prone to numerical instabilities; look-ahead strategies can improve the numerical stability with an increase of the complexity to $O(n^3)$. More recently, $O(n^2)$ algorithms have been proposed in [15] and [23]. They are based on the QR factorization of a displacement structured matrix obtained by means of the normal equations. The use of the normal equations generally squares the condition number of the original problem, with consequent deterioration of the stability.

In this paper we propose new algorithms for the computation of polynomial ϵ-gcd, based on structured matrices. The first approach relies on the formulation of gcd given in terms of the Bézout matrix $B(u, v)$ or of the Sylvester matrix $S(u, v)$ associated with the pair (u, v), and on their reduction to Cauchy-like matrices. It has a computational cost of $O(n^2)$ ops and, from the several numerical experiments performed so far, results robust and numerically stable. For exact gcd where $\epsilon = 0$, k_ϵ coincides with the nullity (i.e., the dimension of the kernel) of $B(u,v)$ and of $S(u,v)$, or equivalently, with the nullity of the Cauchy matrices obtained through the reduction.

The nullity of a Cauchy-like matrix can be computed by means of the GKO fast LU factorization algorithm [8] suitably modified in order to improve the numerical stability with a pivoting strategy. The same technique allows to compute an estimate on the approximate rank (and nullity) of $B(u,v)$ or $S(u,v)$ and therefore yields a tentative guess for k_ϵ. The cofactors (or the gcd) are then computed by solving a suitable Sylvester (Bézout) linear system. Once again, this system is solved by means of the modified GKO algorithm after reduction to Cauchy-like.

A refinement stage performed by means of Newton's iteration, followed by a check of the solution computed in this way, completes the algorithm.

The second approach to approximate gcd computation is presented in Section 4 and relies on the tridiagonalization of the Bézout matrix. We prove that tridiagonalization provides effective means of estimating the approximate rank of $B(u,v)$ and computing a set of coefficients for an approximate common divisor of given degree. In Section 4 we also outline a third algorithm, based on the QR factorization with pivoting of $S(u,v)$ or $B(u,v)$. Though the presentation does not go into much detail, it should be pointed out that these methods have proven to be quite effective and have the merit of highlighting less known aspects of the interplay between polynomial gcd and resultant matrices.

Numerical tests and the results of numerical experiments are reported in Section 5 where we compare our approach with the existing software implemented in Matlab and in Maple.

2. RESULTANT MATRICES AND ϵ-GCD

We recall the definitions of Bézout and Sylvester matrices, along with their interplay with gcd.

2.1 Sylvester and Bézout matrices

The Sylvester matrix of $u(x)$ and $v(x)$ is the $(m+n) \times (m+n)$ matrix

$$S(u,v) = \begin{pmatrix} u_n & u_{n-1} & \dots & u_0 & & 0 \\ & \ddots & & & \ddots & \\ 0 & & u_n & u_{n-1} & \dots & u_0 \\ v_m & v_{m-1} & \dots & v_0 & & 0 \\ & \ddots & & & \ddots & \\ 0 & & v_m & v_{m-1} & \dots & v_0 \end{pmatrix}. \quad (1)$$

where the coefficients of $u(x)$ appear in the first m rows.

Assume that $n \geq m$ and observe that the rational function $(u(x)v(y) - u(y)v(x))/(x-y)$ is actually a polynomial $\sum_{i,j=1}^{n} x^{i-1}y^{j-1}b_{i,j}$ in the variables x, y. The coefficient matrix $B(u,v) = (b_{i,j})$ is called the Bézout matrix of $u(x)$ and $v(x)$.

The following property is well known:

Lemma 2.1. *The nullities of $S(u,v)$ and of $B(u,v)$ coincide with deg(g).*

The next two results show how the gcd of $u(x)$ and $v(x)$ and the corresponding cofactors are related to Sylvester and Bézout submatrices. Recall that, for an integer $\nu \geq 2$ and a polynomial $a(x) = \sum_{j=0}^{\mu} a_j x^j$, the ν-th *convolution matrix* associated with $a(x)$ is the $(\mu+\nu)\times\nu$ Toeplitz matrix having $[a_0, \dots, a_\mu, 0 \dots 0]^T$ as its first column and $[a_0, 0, \dots 0]$ as its first row.

Lemma 2.2. *Let $u(x) = g(x)p(x)$, $v(x) = g(x)q(x)$, then the vector $[q_0, \dots, q_{m-k}, -p_0, \dots, -p_{n-k}]^T$ belongs to the null space of the matrix $S_k = [\mathcal{C}_u \quad \mathcal{C}_v]$, where $\mathcal{C}_u$ is the $(m-k+1)$-st convolution matrix associated with $u(x)$ and $\mathcal{C}_v$ is the $(n-k+1)$-st convolution matrix associated with $v(x)$.*

Theorem 2.3. *([6]) Assume that $B(u,v)$ has rank $n-k$ and denote by $c_1, \dots, c_n$ its columns. Then $c_{k+1}, \dots, c_n$ are linearly independent. Moreover writing each c_i $(1 \leq i \leq k)$ as a linear combination of $c_{k+1}, \dots, c_n$*

$$c_{k-i} = h_{k-i}^{k+1} c_{k+1} + \sum_{j=k+2}^{n} h_{k-i}^{j} c_j, \qquad i = 0, \dots, k-1,$$

one finds that $D(x) = d_0x^k + d_1x^{k-1} + \dots + d_{k-1}x + d_k$ is a gcd for $u(x)$ and $v(x)$, where $d_1, \dots, d_k$ are given by $d_j = d_0 h_{k-j+1}^{k+1}$, with $d_0 \in \mathbb{R}$ or $\mathbb{C}$.

Moreover, we have:

Remark 2.4. *Let $g(x) = \sum_{i=0}^{k} g_i x^i = \gcd(u,v)$, and let $\hat{u}(x)$ and $\hat{v}(x)$ be such that $u(x) = \hat{u}(x)g(x)$, $v(x) = \hat{v}(x)g(x)$. Then we have $B(u,v) = GB(\hat{u},\hat{v})G^T$, where G is the $(n-k)$ convolution matrix associated with $g(x)$.*

2.2 Cauchy-like matrices

An $n \times n$ matrix C is called Cauchy-like of rank r if it has the form

$$C = \left[\frac{\mathbf{u}_i \mathbf{v}_j^H}{f_i - \bar{a}_j}\right]_{i,j=0}^{n-1}, \qquad (2)$$

with $\mathbf{u}_i$ and $\mathbf{v}_j$ row vectors of length $r \leq n$, and f_i and a_j complex scalars such that $f_i - \bar{a}_j \neq 0$ for all i, j. The matrix G whose columns are given by the $\mathbf{u}_i$'s and the matrix B whose rows are given by the $\mathbf{v}_i$'s are called the generators of C.

Equivalently, C is Cauchy-like of rank r if the matrix

$$\nabla_C C = FC - CA^H, \qquad (3)$$

where $F = \text{diag}(f_0, \dots, f_{n-1})$ and $A = \text{diag}(a_0, \dots, a_{n-1})$, has rank r. The operator ∇_C defined in (3) is a displacement operator associated with the Cauchy-like structure, and C is said to have displacement rank equal to r.

The algorithm that we now present is due to Gohberg, Kailath and Olshevsky [8], and is therefore known as *GKO algorithm*; it computes the Gaussian elimination with partial pivoting (GEPP) of a Cauchy-like matrix and can be extended to other classes of displacement structured matrices. The algorithm relies on the following facts: performing Gaussian elimination on an arbitrary matrix is equivalent to applying recursive Schur complementation; Schur complementation preserves the displacement structure; permutations of rows and columns preserve the Cauchy-like structure.

It is therefore possible to directly apply Gaussian elimination with partial pivoting to the generators rather than to the whole matrix C, resulting in increased computational speed and less storage requirements.

So, a step of the fast GEPP algorithm for a Cauchy-like matrix $C = C_1$ can be summarized as follows (we assume that generators (G_1, B_1) of the matrix are given):

(i) Use (2) to recover the first column $\begin{bmatrix} d_1 \\ \ell_1 \end{bmatrix}$ of C_1 from the generators.

(ii) Determine the position (say, $(k, 1)$) of the entry of maximum magnitude in the first column.

(iii) Let P_1 be the permutation matrix that interchanges the first and k-th rows. Interchange the first and k-th diagonal entries of F_1; interchange the first and k-th rows of G_1.

(iv) Recover from the generators the first row $\begin{bmatrix} \tilde{d}_1 & u_1 \end{bmatrix}$ of P_1C_1. Now one has the first column $\begin{bmatrix} 1 \\ \frac{1}{\tilde{d}_1}\tilde{\ell}_1 \end{bmatrix}$ of L and the first row $\begin{bmatrix} \tilde{d}_1 & u_1 \end{bmatrix}$ of U in the LU factorization of P_1C_1.

(v) Compute generators (G_2, B_2) of the Schur complement C_2 of P_1C_1 as follows:

$$\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} 1 \\ \frac{1}{\tilde{d}_1}\tilde{\ell}_1 \end{bmatrix} g_1,$$

$$\begin{bmatrix} 0 & B_2 \end{bmatrix} = B_1 - b_1 \begin{bmatrix} 1 & \frac{1}{\tilde{d}_1}u_1 \end{bmatrix},$$

where g_1 is the first row of G_1 and b_1 is the first column of B_1.

Proceeding recursively, one obtains the factorization $C_1 = PLU$, where P is the product of the permutation matrices used in the process.

Now, let

$$Z_\phi = \begin{pmatrix} \mathbf{0}^T & \phi \\ I_{n-1} & \mathbf{0} \end{pmatrix}, \tag{4}$$

where I_{n-1} is the identity matrix of order $n-1$, and define the matrix operator

$$\nabla_T T = Z_1 T - T Z_{-1}. \tag{5}$$

An $n \times n$ matrix T having low displacement rank with respect to the operator ∇_T (i.e., such that $\nabla_T = GB$, with $G \in \mathbb{C}^{n\times r}$ and $B \in \mathbb{C}^{r \times n}$) is called Toeplitz-like. Sylvester and Bézout matrices are Toeplitz-like.

Toeplitz-like matrices can be transformed into Cauchy-like as follows [10]. Here and hereafter $\hat{\imath}$ denotes the imaginary unit such that $\hat{\imath}^2 = -1$.

Theorem 2.5. *Let T be an $n \times n$ Toeplitz-like matrix. Then $C = \mathcal{F} T D_0^{-1} \mathcal{F}^H$ is a Cauchy-like matrix, i.e.,*

$$\nabla_{D_1, D_{-1}}(C) = D_1 C - C D_{-1} = \hat{G}\hat{B}, \tag{6}$$

where

$$\mathcal{F} = \frac{1}{\sqrt{n}}[e^{\frac{2\pi\hat{\imath}}{n}(k-1)(j-1)}]_{k,j}$$

is the normalized $n \times n$ Discrete Fourier Transform matrix

$$D_1 = \mathrm{diag}(1, e^{\frac{2\pi\hat{\imath}}{n}}, \ldots, e^{\frac{2\pi\hat{\imath}}{n}(n-1)}),$$
$$D_{-1} = \mathrm{diag}(e^{\frac{\pi\hat{\imath}}{n}}, e^{\frac{3\pi\hat{\imath}}{n}}, \ldots, e^{\frac{(2n-1)\pi\hat{\imath}}{n}}),$$
$$D_0 = \mathrm{diag}(1, e^{\frac{\pi\hat{\imath}}{n}}, \ldots, e^{\frac{(n-1)\pi\hat{\imath}}{n}}),$$

and

$$\hat{G} = \mathcal{F}G, \qquad \hat{B}^H = \mathcal{F} D_0 B^H. \tag{7}$$

Therefore the GKO algorithm can be also applied to Toeplitz-like matrices, provided that reduction to Cauchy-like form is applied beforehand. In particular, the generators (G, B) of the matrix $S(u, v)$ can be chosen as follows. Let $N = n+m$; then G is the $N \times 2$ matrix having all zero entries except the entries $(1, 1)$ and $(m + 1, 2)$ which are equal to 1; the matrix B is $2 \times N$, its first row is $[-u_{n-1}, \ldots, -u_1, v_m - u_0, v_{m-1}, \ldots, v_1, v_0 + u_n]$ and its second row is $[-v_{m-1}, \ldots, -v_1, u_n - v_0, u_{n-1}, \ldots, u_1, u_0 + v_m]$. Generators for $B(u, v)$ can be similarly recovered from the representation of the Bézout matrix as sum of products of Toeplitz/Hankel triangular matrices. Generators for the associated Cauchy-like matrix are computed from (G, B) by using (7).

2.3 Modified GKO algorithm

Gaussian elimination with partial pivoting (GEPP) is usually regarded as a reliable method for solving linear systems. Its fast version, though, raises more stability issues.

Sweet and Brent [21] have done an error analysis of the GKO algorithm applied to a Cauchy-like matrix C. They point out that the error propagation depends not only on the magnitude of the triangular factors in the LU factorization of C (as is expected for ordinary Gaussian elimination), but also on the magnitude of the generators. In some cases, the generators can suffer large internal growth, even if the triangular factors do not grow too large, and therefore cause a corresponding growth in the backward and forward error. Experimental evidence shows that this is the case for Cauchy-like matrices derived from Sylvester and Bézout matrices.

However, it is possible to modify the GKO algorithm so as to prevent generator growth, as suggested for example in [20] and [9]. In particular, the latter paper proposes to orthogonalize the first generator before each elimination step; this guarantees that the first generator is well conditioned and allows a good choice of a pivot. In order to orthogonalize G, we need to:

– QR-factorize G, obtaining $G = \mathcal{G}R$, where $\mathcal{G}$ is an $n \times r$ column orthogonal matrix and R is upper triangular;

– define new generators $\tilde{G} = \mathcal{G}$ and $\tilde{B} = RB$.

This method performs partial pivoting on the column of C corresponding to the column of B with maximum norm. This technique is not equivalent to complete pivoting, but allows a good choice of pivots and effectively reduces element growth in the generators, as well as in the triangular factors.

3. FAST ϵ-GCD COMPUTATION

3.1 Estimating degree and coefficients of the ϵ-gcd

We first examine the following problem: find a fast method to determine whether two given polynomials $u(x)$ and $v(x)$ have an ϵ-divisor of given degree k. Throughout we assume that the input polynomials have unitary Euclidean norm.

The coefficients of the cofactors $p(x)$ and $q(x)$ can be obtained by applying Lemma 2.2. A tentative gcd can then be computed as $g(x) = u(x)/p(x)$ or $g(x) = v(x)/q(x)$. Exact or nearly exact polynomial division (i.e., with a remainder of small norm) can be performed in a fast and stable way via evaluation/interpolation techniques [3], which exploit the properties of the discrete Fourier transform.

Alternatively, Theorem 2.3 can be employed to determine the coefficients of a gcd; the cofactors, if required, are computed as $p(x) = u(x)/g(x)$ and $q(x) = v(x)/g(x)$.

The matrix in Lemma 2.2 is formed by two Toeplitz blocks and has displacement rank 2 with respect to the straightforward generalization of the operator ∇_T defined in (5) to the case of rectangular matrices. We seek to employ the modified GKO algorithm to solve the system that arises when applying Lemma 2.2, or the linear system that yields the coefficients of a gcd as suggested by Theorem 2.3.

In order to ensure that the matrices F and A defining the displacement operator ∇_C associated with the reduced matrix have well-separated spectra, a modified version of Theorem 2.5 is needed. Observe that a Toeplitz-like matrix T also has low displacement rank with respect to the operator $\nabla_{Z_1,Z_\theta}(T) = Z_1T - TZ_\theta$, for any $\theta \in \mathbb{C}$, $|\theta| = 1$. Then we have:

THEOREM 3.1. *Let $T \in \mathbb{C}^{n\times m}$ be a Toeplitz-like matrix, satisfying $\nabla_{Z_1,Z_\theta}(T) = Z_1T - TZ_\theta = GB$, where $G \in \mathbb{C}^{n\times\alpha}$, $B \in \mathbb{C}^{\alpha\times m}$ and Z_1, Z_θ are as in (4). Let $N = lcm\ (n, m)$. Then $C = \mathcal{F}_nTD_\theta\mathcal{F}_m$ is a Cauchy-like matrix, i.e. $\nabla_{D_1,D_\theta}(C) = D_1C - CD_\theta = \hat{G}\hat{B}$, where $\mathcal{F}_n$ and $\mathcal{F}_m$ are the normalized Discrete Fourier Transform matrices of order n and m respectively, $D_\theta = \theta D_1$, $D = \mathrm{diag}(1, e^{\frac{\pi \hat{\imath}}{Nm}}, e^{\frac{2\pi\hat{\imath}}{Nm}}, \dots)$, $D_1 = \mathrm{diag}(1, e^{\frac{2\pi\hat{\imath}}{n}}, \dots, e^{\frac{2\pi\hat{\imath}}{n}(n-1)})$ and $\hat{G} = \mathcal{F}_nG$, $\hat{B}^H = \mathcal{F}_mDB^H$.*

The optimal choice for θ is then $\theta = e^{\frac{\pi\hat{\imath}}{N}}$.

The gcd and cofactors obtained from Lemma 2.2 or Theorem 2.3 can be subsequently refined as described in the next section. After the refining step, it is easy to check whether an ϵ-divisor has actually been computed.

We are left with the problem of choosing a tentative gcd degree k_ϵ. A possibility is to employ a bisection technique, which requires to test the existence of an approximate divisor $\log_2 n$ times and therefore preserves the overall quadratic cost of the method.

Alternatively, we propose a heuristic method: the choice of a tentative value for k_ϵ is mainly a matter of approximate rank determination, and it can be performed by relying on the fast LU factorization of $S(u, v)$ or $B(u, v)$. Indeed, observe that the incomplete fast LU factorization computes a Cauchy-like perturbation matrix ΔC such that $C - \Delta C$ has rank $n - k$. If a is the last pivot computed in the incomplete factorization, then as a consequence of Lemma 2.2 in [9], $|a| \le \|\Delta C\|_2$.

Now, let $u_\epsilon(x)$ and $v_\epsilon(x)$ be polynomials of minimum norm and same degrees as $u(x)$ and $v(x)$, such that $u + u_\epsilon$ and $v + v_\epsilon$ have an exact gcd of degree k. Assume $\|u_\epsilon\|_2 \le \epsilon$ and $\|v_\epsilon\|_2 \le \epsilon$. Let C_ϵ be the Cauchy-like matrix obtained via Theorem 2.5 from the Sylvester matrix $S_\epsilon = S(u_\epsilon, v_\epsilon)$. Then $C + C_\epsilon$ has rank $n - k$, too.

If we assume that $\|\Delta C\|_2$ is very close to the minimum norm of a Cauchy-like perturbation that decreases the rank of C to $n - k$, then we have

$$|a| \le \|\Delta C\|_2 \le \|C_\epsilon\|_2 = \|S_\epsilon\|_2 \le \epsilon\sqrt{n+m}, \qquad (8)$$

where the last inequality follows from the structure of the Sylvester matrix. Therefore, if $|a| > \epsilon/\sqrt{n+m}$, then $u(x)$ and $v(x)$ cannot have an ϵ-divisor of degree k. This gives an upper bound on the ϵ-gcd degree based on the absolute values of the pivots found while applying the fast Gaussian elimination to C. The same idea can be applied to the Bézout matrix.

This is clearly a heuristic criterion since it assumes that some uncheckable condition on $||\Delta C||_2$ is satisfied. However, this criterion seems to work quite well in practice and experimental evidence shows that it is more efficient in practice than the bisection strategy, though in principle it does not guarantee that the quadratic cost of the overall algorithm is preserved. When this criterion is applied, the gcd algorithm should check whether it actually provides an upper bound on the gcd degree.

3.2 Refinement

Since the computed value of k_ϵ is the result of a tentative guess, it might happen in principle that the output provided by the algorithm of Section 3.1 is not an ϵ-divisor, is an ϵ-divisor of lower degree, or is a poor approximation of the sought divisor. In order to get rid of this uncertainty, it is suitable to refine this output by means of an *ad hoc* iterative technique followed by a test on the correctness of the ϵ-degree. For this purpose we apply Newton's iteration to the least squares problem

$$F(\mathbf{z}) = \begin{bmatrix} \mathcal{C}_p\mathbf{g} - \mathbf{u} \\ \mathcal{C}_q\mathbf{g} - \mathbf{v} \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} \mathbf{g} \\ \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \qquad (9)$$

where the Euclidean norm of the function $F(\mathbf{z})$ is to be minimized. Here, in boldface we denote the coefficient vectors of the associated polynomials. The matrices $\mathcal{C}_p$ and $\mathcal{C}_q$ are convolution matrices of suitable size associated with the polynomials $p(x)$ and $q(x)$ respectively.

The Jacobian matrix J associated with the problem (9) has the form

$$J = \begin{pmatrix} \mathcal{C}_p & \mathcal{C}_g & 0 \\ \mathcal{C}_q & 0 & \mathcal{C}_g \end{pmatrix}, \qquad (10)$$

where each block is a convolution matrix associated with a polynomial; $\mathcal{C}_p$ is of size $(n+1)\times(k+1)$, $\mathcal{C}_q$ is $(m+1)\times(k+1)$, $\mathcal{C}_g$ in the first block row is $(n+1)\times(n-k+1)$ and $\mathcal{C}_g$ in the second block row is $(m+1)\times(m-k+1)$. This Jacobian matrix, however, is always rank deficient in the exact case, because of the lack of a normalization for the gcd.

REMARK 3.2. *Under the hypotheses stated above, the Jacobian matrix (10) computed at any point $\mathbf{z} = [\mathbf{g}^T \quad \mathbf{p}^T \quad \mathbf{q}^T]^T$ is singular. Moreover, the nullity of J is 1 if and only if $p(x)$, $q(x)$ and $g(x)$ have no common factors. In particular, if $\mathbf{z}$ is a solution of $F(\mathbf{z}) = 0$ and $g(x)$ has maximum degree, i.e. it is a gcd, then J has nullity one and any vector in the null space of J is a multiple of $\mathbf{w} = [\mathbf{g}^T \quad \mathbf{p}^T \quad \mathbf{q}^T]^T$, where $p(x)$ and $q(x)$ are cofactors.*

In order to achieve better stability and convergence properties, we force the Jacobian to have full rank by adding a row, given by $\mathbf{w}^T$. Nevertheless, it can be proved, by relying on the results of [18], that the quadratic convergence of Newton's method in the case of zero residual also holds, in this case, with a rank deficient Jacobian. This property is useful when the initial guess for k_ϵ is too small, since in this case the rank deficiency of the Jacobian is unavoidable.

The new Jacobian $\tilde{J} = \left[\binom{J}{\mathbf{w}^T}\right]$ is associated with the least squares problem of minimizing $\tilde{F(\mathbf{z})} = \left[\binom{F(\mathbf{z})}{\|\mathbf{g}\|^2 - \|\mathbf{p}\|^2 - \|\mathbf{q}\|^2 - K}\right]$, where K is a constant. The choice of $\mathbf{w}^T$ as an additional row helps to ensure that the solution of each Newton's step

$$\mathbf{z}_{j+1} = \mathbf{z}_j - \tilde{J}(\mathbf{z}_j)^\dagger \tilde{F}(\mathbf{z}_j) \qquad (11)$$

is nearly orthogonal to $\ker J$. Here $\tilde{J}(\mathbf{z}_j)^\dagger$ is the Moore-Penrose pseudoinverse of the matrix $\tilde{J}(\mathbf{z}_j)$. For ease of notation, the new Jacobian will be denoted simply as J in the following.

The matrix J has a Toeplitz-like structure, with displacement rank 5. We propose to exploit this property by approximating the solution of each linear least squares problem (11) via fast LU factorization still preserving the quadratic convergence of the modified Newton's iteration obtained in this way. We proceed as follows:

– Compute the factorization $J = LU$, where $J \in \mathbb{C}^{N\times M}$, $L \in \mathbb{C}^{N\times N}$ and $U \in \mathbb{C}^{N\times M}$. For the sake of simplicity, we are overlooking here the presence of permutation matrices due to the pivoting procedure; we can assume that either J or the vectors η_j and $\mathbf{x}_j = \tilde{F}(\mathbf{z}_j)$ have already undergone appropriate permutations.

Consider the following block subdivision of the matrices L e U, where the left upper block has size $M \times M$:

$$L = \left[\begin{array}{c|c} L_1 & 0 \\ \hline L_2 & I \end{array}\right], \qquad U = \left[\begin{array}{c} U_1 \\ \hline 0 \end{array}\right].$$

Analogously, let $\mathbf{x}_j = \left[\begin{array}{c} \mathbf{x}_j^{(1)} \\ \hline \mathbf{x}_j^{(2)} \end{array}\right]$ and observe that $L^{-1} = \left[\begin{array}{c|c} L_1^{-1} & 0 \\ \hline -L_2L_1^{-1} & I \end{array}\right]$.

– Let $\mathbf{y}_j = L_1^{-1}\mathbf{x}_j^{(1)}$. If U_1 is nonsingular, then compute $\mathbf{w}_j$ as solution of $U_1\mathbf{w}_j = \mathbf{y}_j$. Else, consider the block subdivision

$$U_1 = \left[\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & 0 \end{array}\right], \quad \mathbf{w}_j = \left[\begin{array}{c} \mathbf{w}_j^{(1)} \\ \hline \mathbf{w}_j^{(2)} \end{array}\right], \quad \mathbf{y}_j = \left[\begin{array}{c} \mathbf{y}_j^{(1)} \\ \hline \mathbf{y}_j^{(2)} \end{array}\right],$$

such that U_{11} is nonsingular; set all the entries of $\mathbf{w}_j^{(2)}$ equal to zero, and compute $\mathbf{w}_j^{(1)}$ as solution of $U_{11}\mathbf{w}_j^{(1)} = \mathbf{y}_j^{(1)}$.

– If J is rank deficient, find a basis for $\mathcal{K} = \ker J$.

– Subtract from $\mathbf{w}_j$ its projection on $\mathcal{K}$, thus obtaining a vector χ_j. This is the vector that will be used as approximation of a solution of the linear least squares system in the iterative refinement process.

Let $\mathcal{R}$ be the subspace of $\mathbb{C}^N$ spanned by the columns of J. We have

$$\mathbb{C}^N = \mathcal{R} \oplus \mathcal{R}^\perp. \tag{12}$$

Let $\mathbf{x}_j = \alpha_j + \beta_j$ be the decomposition of $\mathbf{x}_j$ with respect to (12), i.e., we have $\alpha_j \in \mathcal{R}$ and $\beta_j \in \mathcal{R}^\perp$.

The Moore-Penrose pseudoinverse of J acts on $\mathbf{x}_j$ as follows: $J^\dagger\alpha_j$ is the preimage of α_j with respect to J and it is orthogonal to $\mathcal{K} = \ker J$, whereas $J^\dagger\beta_j$ is equal to zero.

The LU-based procedure, on the other hand, acts exactly like $J^\dagger$ on α_j, whereas the component β_j is not necessarily sent to 0. Therefore, χ_j is the sum of η_j and of the preimage of β_j with respect to the LU decomposition.

In a general linear least squares problem, there is no reason for $\|\beta_j\|_2$ to be significantly smaller than $\|\mathbf{x}_j\|_2$. In our case, though, the Taylor expansion of $F(\mathbf{z})$ yields:

$$0 = F(\mathbf{z}^*) = F(\mathbf{z}_j) - J(\mathbf{z}_j)\epsilon_j + \mathcal{O}(\|\epsilon_j\|_2^2), \tag{13}$$

where $\epsilon_j = \mathbf{z}_j - \mathbf{z}^*$ and $\mathbf{z}^*$ is such that $F(\mathbf{z}^*) = 0$. It follows from (13) that $\mathbf{x}_j = J(\mathbf{z}_j)\epsilon_j + \mathcal{O}(\|\epsilon_j\|_2^2)$. Since $J(\mathbf{z}_j)\epsilon_j \in \mathcal{R}$, we conclude that $\|\beta_j\|_2 = \mathcal{O}(\|\epsilon_j\|_2^2)$. Therefore, Newton's method applied to the iterative refinement of the polynomial gcd preserves its quadratic convergence rate, even though the linear least squares problems (11) are solved via the LU factorization of the Jacobian.

3.3 The overall algorithm

Algorithm Fastgcd

Input: the coefficients of polynomials $u(x)$ and $v(x)$ and a tolerance ϵ.

Output: an ϵ-gcd $g(x)$; a backward error (residual of the gcd system); possibly perturbed polynomials $\hat{u}(x)$ and $\hat{v}(x)$ and cofactors $p(x)$ and $q(x)$.

Computation:

– Compute the Sylvester matrix $S = S(u, v)$;

– Use Lemma 2.5 to turn S into a Cauchy-like matrix C;

– Perform fast Gaussian elimination with almost complete pivoting on C; stop when a pivot a such that $|a| < \epsilon/\sqrt{n+m}$ is found; let k_0 be the order of the not-yet-factored submatrix $\tilde{U}$ that has a as upper left entry;

– Choose $k = k_0$ as tentative gcd degree;

– Is there an ϵ-divisor of degree k? The answer is found as follows:

- find tentative cofactors by applying the modified GKO algorithm to the system given by Lemma 2.2,
- compute a tentative gcd by performing polynomial division via evaluation/interpolation,
- perform iterative refinement and check whether the backward error is smaller than ϵ;

– If yes, check for $k+1$; if there is also an ϵ-divisor of degree $k+1$, keep checking for increasing values of the degree until a maximum is reached (i.e. a degree is found for which there is no ϵ-divisor);

– If not, keep checking for decreasing values of the degree, until an ϵ-divisor (and gcd) is found.

Observe that a slightly different version of the above algorithm is still valid by replacing the Sylvester matrix with the Bézout matrix. The size of the problem is then roughly reduced by a factor of 2 with clear computational advantage.

It should also be pointed out that the algorithm generally outputs an approximate gcd with complex coefficients, even if $u(x)$ and $v(x)$ are real polynomials. This usually allows for a higher gcd degree or a smaller backward error.

4. QR AND TRIDIAGONALIZATION

The QR factorization and tridiagonalization techniques, borrowed from numerical linear algebra, provide other effective tools for computing an ϵ-gcd.

4.1 QR factorization with pivoting

The algorithm for approximate gcd proposed in [5] exploits the fact that, if the QR factorization of $S(u, v)$ is performed, then the last nonzero row of the triangular factor gives a gcd of $u(x)$ and $v(x)$. We point out here that a similar property holds for the Bézout matrix.

The straightforward application of this result to the approximate case (that is, to the problem of finding an approximate gcd rather than an exact one) involves computing the

QR factorization of the Sylvester or Bézout matrix and taking as coefficients of an ϵ-gcd the entries of the last row of magnitude larger than a fixed tolerance. This method might not lead to a correct approximate gcd because the QR factorization process may suffer from instability.

We propose to overcome this difficulty by using the QR factorization with column pivoting and compute $S(u,v) = QR\Pi$ or $B(u,v) = QR\Pi$, where Π is a permutation matrix and the triangular factor R has diagonal entries of decreasing absolute value. Denote by N the order of the factorized matrix. An upper bound on the ϵ-gcd degree is given by the maximum value of the integer k such that

$$\|R_{(1:N-k,1:n-k)}\| \leq \epsilon\sqrt{N}(1 + \frac{\|R_{(1:N-k,N-k+1:N)}\|}{R_{(N-k+1,N-k+1)}}),$$

where a Matlab-like notation has been used.

The coefficients of an ϵ-gcd are no longer readily available from R because pivoting has been applied. However, as explained in Section 3.1, cofactors can be computed through Lemma 2.2 and an ϵ-gcd is obtained through polynomial division; or we can apply Theorem 2.3 to compute an ϵ-gcd from a Bézout submatrix. A subsequent refinement stage certifies the ϵ-gcd.

4.2 Tridiagonalization of the Bezout matrix

If the input polynomials have real coefficients (and therefore the associated Bézout matrix is real symmetric), then a gcd may be found through Householder tridiagonalization of the Bézout matrix. Assume that $u(x)$ and $v(x)$ are not coprime; then we have:

THEOREM 4.1. *Let $T = HB(u,v)H^T$ be the Householder tridiagonalization of $B(u,v)$, where $u(x)$ and $v(x)$ are real polynomials with $u_0v_0 \neq 0$. Then for almost any choice of $u(x)$ and $v(x)$, the tridiagonal matrix T can be split as the direct sum of a singular irreducible $(n-k)\times(n-k)$ tridiagonal matrix and a null $k \times k$ matrix, where $k+1$ is the degree of* $\gcd(u,v)$.

Besides being useful for rank determination, tridiagonalizing $B(u,v)$ allows to calculate the coefficients of $\gcd(u,v)$. Indeed, observe that in the hypotheses of Theorem 4.1 the last k rows and columns of T are zero. Apply Remark 2.4. Since G, $B(\hat{u},\hat{v})$ and H have maximum rank, it follows that the last l rows of HG must be zero. Let $h = [h_1 \dots h_n]$ be a row vector such that $hG = [0 \dots 0]$. Such a condition can be expressed through the following Hankel linear system:

$$\begin{cases} h_1g_0 + h_2g_1 + \ldots + h_{k+1}g_k = 0 \\ \ldots \\ h_{n-k}g_0 + h_{n-k+1}g_1 + \ldots + h_ng_k = 0 \end{cases}$$

If we assume the gcd to be monic, i.e., $g_k = 1$, the above linear system becomes $A\hat{g} = b$, where $\hat{g} = [g_0 \dots g_{k-1}]^T$, $b = -[h_{k+1} \dots h_n]^T$ and $A = (h_{i+j-1})_{i=1,n-k,j=1,k+1}$

Each of the last l rows of H, which we will call h_i, with $i = n-l+1, \dots, n$, gives a linear system $A_i\hat{g} = b_i$ built like (4.2). Besides, an additional vector in the null space of T is easily computed if necessary, and it yields a system of the type (4.2) as well. So we obtain a system $K\hat{g} = p$, where

$$K = \begin{bmatrix} A_{n-l+1} \\ A_{n-l+2} \\ \vdots \\ A_n \end{bmatrix} \quad \text{and} \quad p = \begin{bmatrix} b_{n-l+1} \\ b_{n-l+2} \\ \vdots \\ b_n \end{bmatrix},$$

which is basically derived from a well-conditioned (in fact, orthogonal) set of generators for the null space of $B(u,v)$. Solving $K\hat{g} = p$ yields the coefficients of $g(x)$. This procedure can be adapted to the approximate case and used, with the addition of iterative refinement, to compute an ϵ-gcd.

5. NUMERICAL EXPERIMENTS

The algorithm Fastgcd has been implemented in Matlab and tested on many polynomials, with satisfactory results. Some of these results are shown in this section and compared to the performance of other implemented methods that are found in the literature, namely UVGCD by Zeng [22], STLN by Kaltofen et al. [13] and QRGCD by Corless et al. [5]. It must be pointed out that comparison with the STLN method is not straightforward, since this methods follows an optimization approach, i.e., it takes two (or more) polynomials and the desired gcd degree k as input, and seeks a perturbation of minimum norm such that the perturbed polynomials have an exact gcd of degree k. Moreover, the algorithms UVGCD and STLN do not normalize the input polynomials, whereas QRGCD and Fastgcd do; therefore all test polynomials are normalized (with unitary Euclidean norm) beforehand.

In the following tests, the residual (denoted as "res") associated with the gcd system is usually shown. In some examples a nearly exact gcd is sought; in these cases it can also be interesting to show the coefficient-wise error on the computed gcd (denoted as "cwe"), since the "correct" gcd is known.

5.1 Badly conditioned polynomials

The test polynomials in this section are taken from [22]. The polynomials in the first example are specifically chosen so that the gcd problem is badly conditioned.

EXAMPLE 5.1. *Let n be an even positive integer and $k = n/2$. Define polynomials $p_n = u_nv_n$ and $q_n = u_nw_n$, where $u_n = \prod_{j=1}^{k}[(x-r_1\alpha_j)^2 + r_1^2\beta_j^2]$, $v_n = \prod_{j=1}^{k}[(x-r_2\alpha_j)^2 + r_2^2\beta_j^2]$, $w_n = \prod_{j=k+1}^{n}[(x-r_1\alpha_j)^2 + r_1^2\beta_j^2]$, $\alpha_j = \cos\frac{j\pi}{n}$, $\beta_j = \sin\frac{j\pi}{n}$, for $r_1 = 0.5$ and $r_2 = 1.5$. The roots of p_n and q_n lie on the circles of radius r_1 and r_2.*

The following table shows the errors given by the examined gcd methods as n increases.

n	Fastgcd	UVGCD	QRGCD
10	6.50×10^{-14}	3.91×10^{-13}	1.57×10^{-12}
12	9.53×10^{-12}	3.87×10^{-12}	3.28×10^{-4}
14	1.32×10^{-11}	2.08×10^{-11}	(*)
16	3.22×10^{-10}	4.28×10^{-10}	(*)
18	4.77×10^{-9}	6.98×10^{-9}	(*)

(*) Here QRGCD fails to find a gcd of correct degree.

In this case, there are no substantial differences between the (good) results provided by Fastgcd and by UVGCD, while QRGCD outputs failure for very ill-conditioned cases.

In the following test, the gcd degree is very sensitive to the choice of the tolerance ϵ.

EXAMPLE 5.2. *Let*

$$p(x) = \prod_{1}^{10}(x - x_j), \qquad q(x) = \prod_{1}^{10}(x - x_j + 10^{-j}),$$

with $x_j = (-1)^j(j/2)$. The roots of p and q have decreasing distances 0.1, 0.01, 0.001, *etc.*

The table shows, for several values of the tolerance, the corresponding gcd degree and residual found by Fastgcd and UVGCD. Fastgcd gives better results, since it generally finds gcds of higher degree. The algorithm QRGCD, on the contrary, outputs failure for all values of ϵ smaller than 10^{-2}.

ϵ	Fastgcd		UVGCD	
	deg	res	deg	res
10^{-2}	9	0.0045	9	0.0040
10^{-3}	8	2.63×10^{-4}	8	1.73×10^{-4}
10^{-5}	7	9.73×10^{-6}	4	1.77×10^{-5}
10^{-7}	5	8.59×10^{-9}	2	2.25×10^{-14}
10^{-9}	1	3.98×10^{-11}		

We have also studied this example using the STLN method; though the employed approach is entirely different. The following table shows the residuals computed by STLN for several values of the degree.

deg gcd	res	deg gcd	res
9	5.65×10^{-3}	6	2.58×10^{-7}
8	2.44×10^{-4}	5	6.34×10^{-9}
7	1.00×10^{-5}	4	1.20×10^{-10}

5.2 High gcd degree

This example, also taken from [22], uses polynomials such that their gcd has a large degree.

EXAMPLE 5.3. *Let $p_n = u_n v$ and $q_n = u_n w$, where $v(x) = \sum_{j=0}^{3} x^j$ and $w(x) = \sum_{j=0}^{4}(-x)^j$ are fixed polynomials and u_n is a polynomial of degree n whose coefficients are random integer numbers in the range* $[-5, 5]$.

The following table shows the coefficient-wise errors on the computed gcd for large values of n. Fastgcd and UVGCD perform similarly, with errors of the same order of magnitude, while QRGCD provides a worse coefficient-wise error.

n	Fastgcd	UVGCD	QRGCD
50	9.82×10^{-15}	8.88×10^{-16}	1.72×10^{-12}
100	1.04×10^{-15}	6.66×10^{-16}	4.80×10^{-8}
200	1.30×10^{-15}	9.71×10^{-16}	2.39×10^{-11}
500	2.87×10^{-15}	1.22×10^{-15}	

5.3 Unbalanced coefficients

This is another example taken from [22].

EXAMPLE 5.4. *Let $p = uv$ and $q = uw$, where $v(x)$ and $w(x)$ are as in Example 5.3 and $u(x) = \sum_{j=0}^{15} c_j 10^{e_j} x^j$, where c_j and e_j are random integers in $[-5, 5]$ and $[0, 6]$ respectively.*

In this example $u(x)$ is the gcd of $p(x)$ and $q(x)$ and the magnitude of its coefficients varies between 0 and 5×10^6. If an approximate gcd algorithm is applied and the coefficient-wise relative error θ is calculated, then $N = \log_{10} \theta$ is roughly the minimum number of correct digits for the coefficients of $u(x)$ given by the chosen method. 100 repetitions of this test are performed. The average number of correct digits is 10.70 for Fastgcd and 10.98 for UVGCD.

5.4 Multiple roots

EXAMPLE 5.5. *Let $u(x) = (x^3 + 3x - 1)(x - 1)^k$ for a positive integer k, and let $v(x) = u'(x)$. The gcd of $u(x)$ and $v(x)$ is $g(x) = (x - 1)^{k-1}$.*

The residuals computed for several values of k and for $\epsilon = 10^{-6}$ are shown here. The computed gcd degrees are understood to be correct.

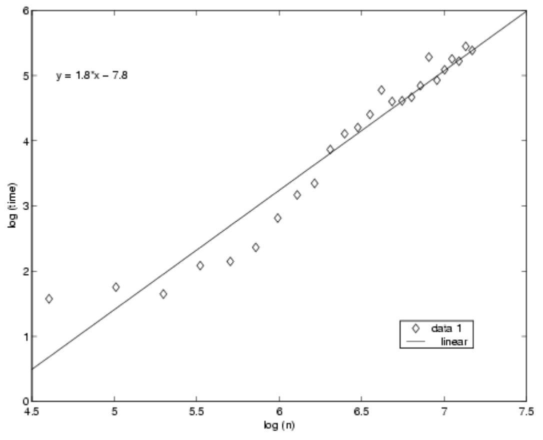

Figure 1: Running time of the algorithm Fastgcd

k	Fastgcd	UVGCD	QRGCD
15	1.40×10^{-13}	3.84×10^{-13}	7.04×10^{-7}
25	1.14×10^{-10}	3.61×10^{-12}	(*)
35	1.36×10^{-8}	1.03×10^{-9}	(*)
45	1.85×10^{-5}	1.72×10^{-9}	(*)

(*) Here QRGCD does not detect a gcd of correct degree.

5.5 Small leading coefficient

A gcd with a small leading coefficient may represent in many cases a source of instability.

EXAMPLE 5.6. *For a given (small) parameter $\alpha \in \mathbb{R}$, let $g(x) = \alpha x^3 + 2x^2 - x + 5$, $p(x) = x^4 + 7x^2 - x + 1$ and $q(x) = x^3 - x^2 + 4x - 2$ and set $u(x) = g(x)p(x)$, $v(x) = g(x)q(x)$.*

We applied Fastgcd and QRGCD to this example, with α ranging between 10^{-5} and 10^{-10}. It turns out that, for $\alpha < 10^{-5}$, QRGCD fails to recognize the correct gcd degree and outputs a gcd of degree 2. Fastgcd, on the contrary, always outputs a correct gcd, with a residual of 2.40×10^{-16}.

5.6 Running time

We have checked the growth rate of the running time of the algorithm Fastgcd on pairs of polynomials whose GCD and cofactors are defined as the polynomials $u_n(x)$ introduced in Section 5.2. Polynomials of degree $N = 2n$ ranging between 100 and 1300 have been used. Figure 1 shows the running time (in seconds) versus the degree in log-log scale, with a linear fit and its equation. Roughly speaking, the running time grows as $\mathcal{O}(N^\alpha)$, where α is the coefficient of the linear term in the equation, i.e. 1.8 in our case. This computation has been done using Matlab 7.1.

We next show a comparison between the running times of Fastgcd and UVGCD. In order to avoid randomly chosen coefficients, we define a family of test polynomials as follows. Let k be a positive integer and let $n_1 = 25k$, $n_2 = 15k$ and $n_3 = 10k$. For each value of k define the cofactors $p_k(x) = (x^{n_1} - 1)(x^{n_2} - 2)(x^{n_3} - 3)$ and $q_k(x) = (x^{n_1} + 1)(x^{n_2} + 5)(x^{n_3} + \hat{\imath})$. The test polynomials are $u_k(x) = g(x)p_k(x)$ and $v_k(x) = g(x)q_k(x)$, where the gcd $g(x) = x^4 + 10x^3 + x - 1$ is a fixed polynomial.

Figure 2 shows the computing times required by Fastgcd and UVGCD on $u_k(x)$ and $v_k(x)$ for $k = 1, \dots 7$. This experiment has been done using Matlab 6.1, in order to be able to run the program UVGCD. In this Matlab version, our

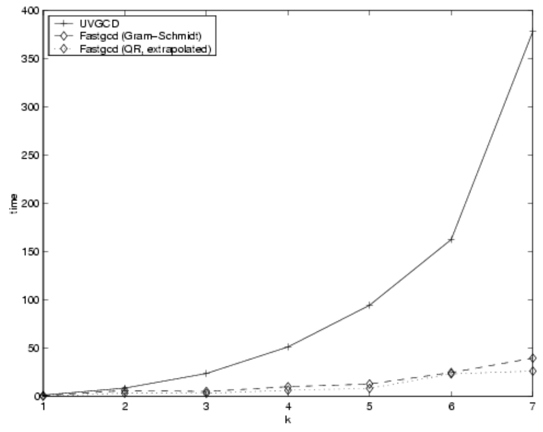

Figure 2: Comparison between the running times of Fastgcd and UVGCD.

implementation of Fastgcd that uses the built-in command `qr` to compute the reduced QR factorization of displacement generators is computationally expensive, therefore we show the results given by an implementation that uses modified Gram-Schmidt orthogonalization instead. The figure also shows computing times for the QR-based implementation that are extrapolated from a comparison between the two implementations of Fastgcd run in Matlab 7.1. The plot clearly shows that the time growth for Fastgcd is much slower than for UVGCD. For $N = 350$ our method is faster than UVGCD by a factor of about 15.

The Matlab software for Fastgcd is available upon request.

Acknowledgements

The authors wish to thank anonymous referees for their valuable remarks which allowed to substantially improve the presentation of this paper.

6. REFERENCES

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

[2] B. Beckermann and G. Labahn. When are two numerical polynomials relatively prime? *J. Symbolic Comput.*, 26(6):677–689, 1998.

[3] D. A. Bini and V. Y. Pan. *Polynomial and Matrix Computations, vol. I: Fundamental Algorithms.* Birkhäuser, Boston, 1994.

[4] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for approximate polynomial systems. In *Proc. International Symposium on Symbolic and Algebraic Computation*, pages 195–207, 1995.

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

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

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

[8] I. Gohberg, T. Kailath, and V. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. Comp.*, 64(212):1557–1576, 1995.

[9] M. Gu. Stable and efficient algorithms for structured systems of linear equations. *SIAM J. Matrix Anal. Appl.*, 19(2):279–306, 1998.

[10] G. Heinig. Inversion of generalized Cauchy matrices and other classes of structured matrices. In *IMA volumes in Mathematics and its Applications.* Springer, New York, 1995.

[11] V. Hribernig and H. J. Stetter. Detection and validation of clusters of polynomial zeros. *J. Symb. Comp.*, 24(6):667–681, 1997.

[12] C.-P. Jeannerod and G. Labahn. SNAP user's guide. Technical Report CS-2002-22, University of Waterloo, 2002.

[13] E. Kaltofen, Z. Yang, and L. Zhi. Approximate greatest common divisors of several polynomials with linearly constrained coefficients and singular polynomials. In *Proc. International Symposium on Symbolic and Algebraic Computations*, 2006.

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

[15] B. Li, Z. Yang, and L. Zhi. Fast low rank approximation of a Sylvester matrix by structure total least norm. *Journal of Japan Society for Symbolic and Algebraic Computation*, 11:165–174, 2005.

[16] M.-T. Noda and T. Sasaki. Approximate GCD and its application to ill-conditioned algebraic equations. *J. Comput. Appl. Math.*, 38(1-3):335–351, 1991.

[17] V. Y. Pan. Computation of approximate polynomial gcds and an extension. *Information and Computation*, 167(2):71–85, 2001.

[18] L. B. Rall. Convergence of the Newton process to multiple solutions. *Num. Math*, 9:23–27, 1966.

[19] A. Schönhage. Quasi-GCD computations. *J. Complexity*, 1:118–137, 1985.

[20] M. Stewart. Stable pivoting for the fast factorization of Cauchy-like matrices. preprint, 1997.

[21] D. R. Sweet and R. P. Brent. Error analysis of a fast partial pivoting method for structured matrices. In T. Luk, editor, *Adv. Signal Proc. Algorithms, Proc. of SPIE*, pages 266–280, 1995.

[22] Z. Zeng. The approximate GCD of inexact polynomials. Part I: a univariate algorithm. To appear.

[23] L. Zhi. Displacement structure in computing the approximate GCD of univariate polynomials. In W. Sit and Z. Li, editors, *Lecture Notes Series on Computing*, pages 228–298. World Scientific, 2003.