A fast algorithm for approximate polynomial gcd based on structured matrix computations

Dario A. Bini and Paola Boito

In Memory of Georg Heinig

Abstract. An $O(n^2)$ complexity algorithm for computing an ϵ-greatest common divisor (gcd) of two polynomials of degree at most n is presented. The algorithm is based on the formulation of polynomial gcd given in terms of resultant (Bézout, Sylvester) matrices, on their displacement structure and on the reduction of displacement structured matrices to Cauchy-like form originally pointed out by Georg Heinig. A Matlab implementation is provided. Numerical experiments performed with a wide variety of test problems, show the effectiveness of this algorithm in terms of speed, stability and robustness, together with its better reliability with respect to the available software.

Mathematics Subject Classification (2000). 68W30, 65F05, 15A23.

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

1. Introduction

A basic problem in algebraic computing is the evaluation of polynomial gcd: given the coefficients of two univariate polynomials

$$u(x) = \sum_{i=0}^{n} u_i x^i, \quad v(x) = \sum_{i=0}^{m} v_i x^i,$$

compute the coefficients of their greatest common divisor $g(x)$.

In many applications, 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. If $u(x)$ and $v(x)$ have a nontrivial gcd, it turns out that arbitrarily small perturbations in the coefficients of $u(x)$ and $v(x)$

Supported by MIUR grant 2006017542.

may transform $u(x)$ and $v(x)$ into relatively prime polynomials. Therefore, it is clear that the concept of gcd is not well suited to deal with applications where data are approximatively known. This is why the notion of approximate gcd, or ϵ-gcd, has been introduced. For more details on this topic we refer the reader to [19] [7],[18], [23] and to the references therein.

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 $\|u(x)-\hat{u}(x)\| \leq \epsilon\|u(x)\|$, $\|v(x)-\hat{v}(x)\| \leq \epsilon\|v(x)\|$ 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.*

Several algorithms for the computation of an approximate polynomial gcd can be found in the literature; they rely on different techniques, such as the Euclidean algorithm [1], [2], [12], [17], optimization methods [15], SVD and factorization of resultant matrices [5], [4], [23], Padé approximation [3], [18], root grouping [18]. Some of them have been implemented inside numerical/symbolic packages like the algorithm of Zeng [23] in Matlab$^{\mathrm{TM}}$ and the algorithms of Kaltofen [14], of Corless et al [4], of Labahn and Beckermann [13] in Maple$^{\mathrm{TM}}$. 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 [24] and [16]. 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 a consequent deterioration of the stability.

In this paper we present an algorithm for (approximate) gcd computation which has a cost of $O(n^2)$ arithmetic operations and, from the several numerical experiments performed so far, results robust and numerically stable. The algorithm relies on the formulation of the gcd problem given in terms of the Bézout matrix $B(u,v)$ or of the Sylvester matrix $S(u,v)$ associated with the pair of polynomials (u,v), and on their reduction to Cauchy-like matrices by means of unitary transforms. This kind of reduction, which is fundamental for our algorithm, was discovered and analyzed by Georg Heinig in [10] in the case of general Toeplitz-like matrices.

For exact gcd, where $\epsilon = 0$, the degree k_ϵ of the ϵ-gcd 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.

Our algorithm can be divided into two stages. In the first stage, from the coefficients of the input polynomials a resultant matrix (Sylvester or Bézout) is computed and reduced to Cauchy-like form. The GKO algorithm of Gohberg,

Kailath and Olshevsky [8] for the PLU factorization is applied to the Cauchy-like matrix obtained in this way. The algorithm relies on the pivoting strategy and on a suitable technique used to control the growth of the generators. This algorithm is rank-revealing since in exact arithmetic it provides a matrix U with the last k rows equal to zero, where k is the nullity of the matrix. In our case, where the computation is performed in floating point arithmetic with precision μ and where $\epsilon > \mu$, the algorithm is halted if the last computed pivot a is such that $|a| \leq \epsilon\sqrt{m+n}$. This provides an estimate of the value k_ϵ and a candidate $g_\epsilon(x)$ to an ϵ-divisor of $u(x)$ and $v(x)$.

In the second stage, the tentative ϵ-divisor $g_\epsilon(x)$ is refined by means of Newton's iteration and a test is applied to check that $g_\epsilon(x)$ is an ϵ-common divisor. In this part, the value of k_ϵ can be adaptively modified in the case where $g_\epsilon(x)$ has not the maximum degree, or if $g_\epsilon(x)$ is not an ϵ-divisor of $u(x)$ and $v(x)$.

It is important to point out that the Jacobian system, which has to be solved at each Newton's iteration step, is still a Toeplitz-like linear system which can be reduced once again to Cauchy-like form and solved by means of the pivoted GKO algorithm.

The algorithm has been implemented in Matlab, tested with a wide set of polynomials and compared with the currently available software, in particular the Matlab and Maple packages UVGCD by Zeng [23], STLN by Kaltofen et al. [14] and QRGCD by Corless et al. [4]. We did not compare our algorithm to the ones of [16], [24] since the software of the latter algorithms is not available.

We have considered the test polynomials of [23] and some new additional tests which are representative of difficult situations. In all the problems tested so far our algorithm has shown a high reliability and effectiveness, moreover, its $O(n^2)$ complexity makes it much faster than the currently available algorithms already for moderately large values of the degree. Our Matlab code is available upon request.

The paper is organized as follows. In Section 2 we recall the main tools used in the paper, among which, the properties of Sylvester and Bézout matrices, their interplay with gcd, the reduction to Cauchy-like matrices and a modified version of the GKO algorithm. In Section 3 we present the algorithms for estimating the degree and the coefficients of the ϵ-gcd together with the refinement stage based on Newton's iteraton. Section 4 reports the results of the numerical experiments together with the comparison of our Matlab implementation with the currently available software.

2. Resultant matrices and ϵ-gcd

We recall the definitions of Bézout and Sylvester matrices and 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

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

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

Assume that $n\ge m$ and observe that the rational function

$$b(x,y)=\frac{u(x)v(y)-u(y)v(x)}{x-y}$$

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 $\nu\ge 2$ and a polynomial $a(x)=\sum_{j=0}^{\mu} a_j x^j$, the ν-th *convolution matrix* associated with $a(x)$ is the Toeplitz matrix having $[a_0,\dots,a_\mu,\underbrace{0\dots 0}_{\nu-1}]^T$ as its first column and $[a_0,\underbrace{0,\dots 0}_{\nu-1}]$ 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 $(1\le i\le k)$ 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 $D(x)=d_0x^k+d_1x^{k-1}+\cdots+d_{k-1}x+d_k$ is a gcd for $u(x)$ and $v(x)$, where $d_1,\dots,d_k$ are given by $d_j=d_0h_{k-j+1}^{k+1}$, with $d_0\in\mathbb{R}$ or $\mathbb{C}$.

Moreover, we have:

Remark 2.4. *Let $g(x) = \sum_{i=0}^{k} g_i x^i$ be the gcd of $u(x)$ and $v(x)$, 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 $(n-k)$-th 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

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

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 rows are given by the $\mathbf{u}_i$'s and the matrix B whose columns 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

$$\nabla_C C = FC - CA^H, \tag{3}$$

where $F = \mathrm{diag}(f_0, \ldots, f_{n-1})$ and $A = \mathrm{diag}(a_0, \ldots, a_{n-1})$, 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

Fact 2.5. *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 $\begin{bmatrix} d_1 \\ l_1 \end{bmatrix}$ 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 $P_1 C_1$. Now one has

the first column $\begin{bmatrix} 1 \\ \frac{1}{\tilde{d}_1}\tilde{l}_1 \end{bmatrix}$ 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 $P_1 \cdot C_1$ as follows:

$$\begin{bmatrix} 0 \\ G_2 \end{bmatrix} = G_1 - \begin{bmatrix} 1 \\ \frac{1}{\tilde{d}_1}\tilde{l}_1 \end{bmatrix} g_1, \qquad \begin{bmatrix} 0 & B_2 \end{bmatrix} = B_1 - b_1 \begin{bmatrix} 1 & \frac{1}{\tilde{d}_1}u_1 \end{bmatrix}, \tag{4}$$

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 $C_1 = P \cdot L \cdot U$, where P is the product of the permutation matrices used in the process.

Now, let

$$Z_\phi = \begin{pmatrix} 0 & \dots & \dots & 0 & \phi \\ 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix} \tag{5}$$

and define the matrix operator

$$\nabla_T T = Z_1 T - T Z_{-1}. \tag{6}$$

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, with displacement rank 2.

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 2.6. *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.,*

$$\nabla_{D_1, D_{-1}}(C) = D_1 C - C D_{-1} = \hat{G}\hat{B}, \tag{7}$$

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

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

and

$$\hat{G} = \mathcal{F} G, \qquad \hat{B}^H = \mathcal{F} D_0 B^H. \tag{8}$$

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 $S(u, v)$ with respect to the Toeplitz-like structure 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 and second rows are

$$[-u_{n-1}, \ldots, -u_1, v_m - u_0, v_{m-1}, \ldots, v_1, v_0 + u_n],$$
$$[-v_{m-1}, \ldots, -v_1, u_n - v_0, u_{n-1}, \ldots, u_1, u_0 + v_m],$$

respectively. 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 (8).

2.3. Modified GKO algorithm

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

Sweet and Brent [22] 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 [21] 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 nevertheless 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. Once the cofactors are known, a tentative gcd can 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 (see [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 (6) 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 2.6 is needed. Observe that a Toeplitz-like matrix T also has low displacement rank with respect to the operator $\nabla_{Z_1,Z_\theta}(T) = Z_1T - T \cdot Z_\theta$, 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 (5). Let $N = lcm\ (n, m)$. Then $C = \mathcal{F}_nTD_\theta\mathcal{F}_m$ is a Cauchy-like matrix, i.e.

$$\nabla_{D_1,D_\theta}(C) = D_1C - CD_\theta = \hat{G}\hat{B}, \tag{9}$$

where $\mathcal{F}_n$ and $\mathcal{F}_m$ are the normalized Discrete Fourier Transform matrices of order n and m respectively,

$$D_\theta = \theta \cdot D_1,$$
$$D = \mathrm{diag}(1, e^{\frac{\pi\hat{\imath}}{Nm}}, e^{\frac{2\pi\hat{\imath}}{Nm}}, \dots)$$
$$D_1 = \mathrm{diag}(1,\ e^{\frac{2\pi\hat{\imath}}{n}}, \dots, e^{\frac{2\pi\hat{\imath}}{n}(n-1)})$$

and $\hat{G} = \mathcal{F}_nG, \qquad \hat{B}^H = \mathcal{F}_mDB^H.$

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, a heuristic method of choosing a tentative value for k_ϵ can be designed by observing that, as a consequence of the properties of resultant matrices presented in Section 2.1, the choice of a suitable k_ϵ is mainly a matter of approximate rank determination, and the fast LU factorization of the Sylvester or Bézout matrix might provide reasonably useful values for k_ϵ.

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], $|a| \leq \|\Delta C\|_2$.

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 $\|u_\epsilon\|_2 \leq \epsilon$ and $\|v_\epsilon\|_2 \leq \epsilon$. Let C_ϵ be the Cauchy-like matrix obtained via Theorem 2.6 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

$$|a| \leq \|\Delta C\|_2 \leq \|C_\epsilon\|_2 = \|S_\epsilon\|_2 \leq \epsilon\sqrt{n+m}, \tag{10}$$

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. When it is applied, the gcd algorithm should check whether it actually provides an upper bound on the gcd degree. We use this criterion for the determination of a tentative gcd degree in our implementation of the algorithm. In fact, experimental evidence shows that this criterion is usually 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.

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 defined by

$$F(\mathbf{z}) = \begin{bmatrix} \mathcal{C}_p\mathbf{g} - \mathbf{u} \\ \mathcal{C}_q\mathbf{g} - \mathbf{v} \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} \mathbf{g} \\ \mathbf{p} \\ \mathbf{q} \end{bmatrix}, \tag{11}$$

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 (11) has the form

$$J = \begin{pmatrix} \mathcal{C}_p & \mathcal{C}_g & 0 \\ \mathcal{C}_q & 0 & \mathcal{C}_g \end{pmatrix}, \tag{12}$$

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, because of the lack of a normalization for the gcd.

Remark 3.2. *Under the hypotheses stated above, the Jacobian matrix (12) 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 [20], 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 $\tilde{J} = \begin{bmatrix} \binom{J}{\mathbf{w}^T} \end{bmatrix}$ is associated with the least squares problem that minimizes $\tilde{F}(\mathbf{z}) = \begin{bmatrix} \binom{F(\mathbf{z})}{\|\mathbf{g}\|^2 - \|\mathbf{p}\|^2 - \|\mathbf{q}\|^2 - K} \end{bmatrix}$, 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

$$\mathbf{z}_{j+1} = \mathbf{z}_j - \tilde{J}(\mathbf{z}_j)^\dagger \tilde{F}(\mathbf{z}_j) \tag{13}$$

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

$$J\eta_j = \tilde{F}(\mathbf{z}_j), \quad \eta_j = \mathbf{z}_j - \mathbf{z}_{j+1}$$

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

$$\mathbb{C}^N = \mathcal{R} \oplus \mathcal{R}^{\perp}. \tag{14}$$

Let $\mathbf{x}_j = \alpha_j + \beta_j$ be the decomposition of $\mathbf{x}_j$ with respect to (14), 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:

$$0 = F(\mathbf{z}^*) = F(\mathbf{z}_j) - J(\mathbf{z}_j)\epsilon_j + \mathcal{O}(\|\epsilon_j\|_2^2), \tag{15}$$

where $\epsilon_j = \mathbf{z}_j - \mathbf{z}^*$ and $\mathbf{z}^*$ is such that $F(\mathbf{z}^*) = 0$. It follows from (15) 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 (13) are treated using via the LU factorization of the Jacobian.

The iterative process ends when at least one of the following criteria is satisfied:

1. the residual (that is, the Euclidean norm of the function $F(\mathbf{z})$) becomes smaller that a fixed threshold,
2. the number of iteration reaches a fixed maximum,
3. the residual given by the last iteration is greater that the residual given by the previous iteration.

The purpose of the third criterion is to avoid spending computational effort on tentative gcds that are not in fact suitable candidates. However, its use with Newton's method may pose some difficulties, because it is generally difficult to predict the global behaviour of this method; in particular, it might happen that the residual does not decrease monotonically. The usual way to overcome this obstacle is to use instead a relaxed version of Newton that includes a line search. More precisely, instead of the iteration (13) one computes

$$\mathbf{z}_{j+1} = \mathbf{z}_j - \alpha_j \tilde{J}(\mathbf{z}_j)^\dagger \tilde{F}(\mathbf{z}_j), \tag{16}$$

where α_j is chosen – using a one-dimensional minimization method – so as to approximately minimize the norm of $\tilde{F}(\mathbf{z}_j)$.

The drawback of this technique is that it slows down convergence: the quadratic convergence that was one of the main interesting points of Newton's method is lost if one consistently performs iterations of the type (16). For this reason we employ here a hybrid method: At each step, the algorithm evaluates the descent direction $\tilde{J}(\mathbf{z}_j)^\dagger \tilde{F}(\mathbf{z}_j)$ and checks if a pure Newton step (that is, (16) with $\alpha_j = 1$) decreases the residual. If this is the case, then the pure Newton step is actually performed; otherwise, α_j and subsequently $\mathbf{z}_{j+1}$ are computed by calling a line search routine. In this way, most of the optimization work is still performed by pure Newton iterations, so that the overall method remains computationally cheap; the line search, called only when necessary, is helpful in some difficult cases and ensures that the method has a sound theoretical basis.

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 associated with $u(x)$ and $v(x)$;

– Use Lemma 2.6 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. With this replacement the size of the problem is 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. 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 [23], STLN by Kaltofen et al. [14] and QRGCD by Corless et al. [4]. Matlab experiments with Fastgcd and UVGCD are performed using version 7.5.0 running under Windows; we use here the P-code for UVGCD contained in the Apalab toolbox.

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, we generally display the residual (denoted as "res") associated with the gcd system (recall that the residual is defined here as the Euclidean norm of the function $F(\mathbf{z})$ and it may slightly differ from the residual as defined by other authors). In some examples, where a nearly exact gcd is sought, we report the coefficient-wise error on the computed gcd (denoted as "cwe"), since the "correct" gcd is known.

4.1. Badly conditioned polynomials

The test polynomials in this section are taken from [23]. The polynomials in the first example are specifically chosen so that the gcd problem is badly conditioned.

Example 4.1. *Let n be an even positive integer and $k = n/2$; define $p_n = u_n v_n$ and $q_n = u_n w_n$, where*

$$u_n = \prod_{j=1}^{k}[(x - r_1\alpha_j)^2 + r_1^2\beta_j^2], \quad v_n = \prod_{j=1}^{k}[(x - r_2\alpha_j)^2 + r_2^2\beta_j^2],$$

$$w_n = \prod_{j=k+1}^{n}[(x - r_1\alpha_j)^2 + r_1^2\beta_j^2], \quad \alpha_j = \cos\frac{j\pi}{n}, \quad \beta_j = \sin\frac{j\pi}{n},$$

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 coefficient-wise errors given by the examined gcd methods as n increases.

n	Fastgcd	UVGCD	QRGCD
10	6.44×10^{-13}	3.24×10^{-13}	1.57×10^{-12}
12	5.23×10^{-12}	1.40×10^{-12}	3.28×10^{-4}
14	1.79×10^{-11}	2.27×10^{-11}	(*)
16	5.27×10^{-10}	4.41×10^{-11}	(*)
18	6.11×10^{-9}	3.63×10^{-10}	(*)

(*) 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. It should be pointed out, however, that the results given by UVGCD vary between trials, which makes comparisons more difficult.

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

Example 4.2. *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	1.72×10^{-4}
10^{-4}	7	9.73×10^{-6}	(*)	
10^{-6}	6	2.78×10^{-7}	1	3.34×10^{-16}
10^{-7}	5	8.59×10^{-9}	1	3.34×10^{-16}

(*)Here UVGCD outputs the same result as above due to a different definition of residual.

It is interesting to observe that for $\epsilon \leq 10^{-5}$, UVGCD computes a common ϵ-divisor which does not have the maximum degree, while Fastgcd always provides an ϵ-divisor of higher degree.

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}

4.2. High gcd degree

In this example, also taken from [23], the gcd has a large degree.

Example 4.3. *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 residuals and the coefficient-wise errors on the computed gcd for large values of n. Here, Fastgcd and UVGCD perform similarly while QRGCD provides a worse coefficient-wise error.

n	Fastgcd		UVGCD		QRGCD
	res	cwe	res	cwe	cwe
50	2.97×10^{-16}	5.04×10^{-16}	2.43×10^{-16}	8.32×10^{-16}	1.72×10^{-12}
100	2.91×10^{-16}	1.41×10^{-15}	1.83×10^{-16}	7.77×10^{-16}	4.80×10^{-8}
200	5.08×10^{-16}	7.29×10^{-15}	1.72×10^{-16}	9.99×10^{-16}	2.39×10^{-11}
500	4.04×10^{-16}	3.12×10^{-15}	2.10×10^{-15}	1.35×10^{-14}	
1000	3.98×10^{-16}	3.28×10^{-15}	2.26×10^{-16}	1.67×10^{-15}	

4.3. Unbalanced coefficients

This is another example taken from [23].

Example 4.4. *Let $p = uv$ and $q = uw$, where $v(x)$ and $w(x)$ are as in Example* 4.3 *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 found in an experiment of this type is 10.63 for Fastgcd and 10.83 for UVGCD. Therefore the two algorithms give comparable results. Residuals are always about 10^{-16}. QRGCD, on the contrary, achieves an average of 7.46 correct digits.

4.4. Multiple roots

Example 4.5. *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 coefficient-wise errors computed by Fastgcd, UVGCD and QRGCD for several values of k and for $\epsilon = 10^{-6}$ are shown in the following table. Unless otherwise specified, the computed gcd degrees are understood to be correct.

k	Fastgcd	UVGCD	QRGCD
15	5.18×10^{-13}	4.27×10^{-13}	7.04×10^{-7}
25	9.31×10^{-11}	1.99×10^{-11}	(*)
35	1.53×10^{-8}	4.44×10^{-9}	(*)
45	6.61×10^{-6}	4.04×10^{-8}	(*)

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

The algorithm UVGCD has been specifically designed for polynomials with multiple roots and is therefore very efficient. Fastgcd also provides good results, with backward errors (residuals) always of the order of the machine epsilon, whereas QRGCD fails to find a gcd of correct degree as soon as the root multiplicity is larger than 15.

4.5. Small leading coefficient

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

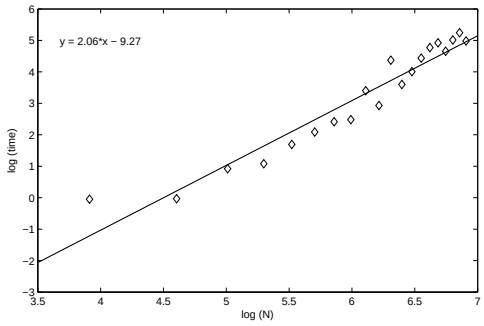

FIGURE 1. Running time of the algorithm Fastgcd

Example 4.6. *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^{-15}. 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 recognizes the correct gcd degree, with a residual of the order of the machine epsilon.

4.6. 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 like the polynomials $u_n(x)$ introduced in Section 4.2. Polynomials of degree $N = 2n$ ranging between 50 and 1000 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., 2.06 in our case.

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 $k = 1, \ldots 8$. The plot clearly shows that the time growth for Fastgcd is much slower than for UVGCD.

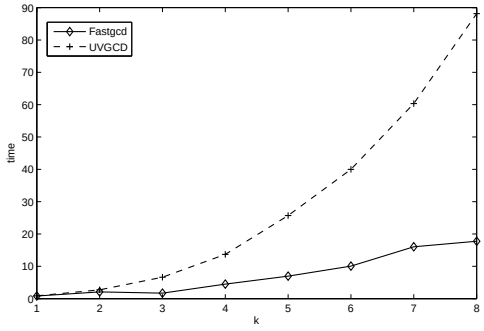

FIGURE 2. Comparison between the running times of Fastgcd and UVGCD.

References

[1] B. Beckermann, G. Labahn, *When are two numerical polynomials relatively prime?*, J. Symbolic Comput. **26**, 677-689 (1998).

[2] B. Beckermann, G. Labahn, *A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials*, J. Symb. Comp. **26**, 691-714 (1998).

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

[4] R. M. Corless, S. M. Watt, L. Zhi, *QR Factoring to Compute the GCD of Univariate Approximate Polynomials*, IEEE Trans. Signal Processing **52**, 3394-3402 (2004).

[5] R. M. Corless, P. M. Gianni, B. M. Trager, S. M. Watt, *The Singular Value Decomposition for Approximate Polynomial Systems*, Proc. International Symposium on Symbolic and Algebraic Computation, July 10-12 1995, Montreal, Canada, ACM Press 1995, pp. 195-207.

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

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

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

[9] M. Gu, *Stable and Efficient Algorithms for Structured Systems of Linear Equations*, SIAM J. Matrix Anal. Appl. **19**, 279-306 (1998).

[10] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, Linear Algebra in Signal Processing, IMA volumes in Mathematics and its Applications **69**, 95-114 (1994).

[11] G. Heinig, P. Jankowsky, K. Rost, *Fast inversion of Toeplitz-plus-Hankel matrices*, Numer. Math. **52**, 665-682 (1988).

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

[13] C.-P. Jeannerod, G. Labahn, *SNAP User's Guide*, UW Technical Report no. CS-2002-22 (2002).

[14] E. Kaltofen, Z. Yang, L. Zhi, *Approximate Greatest Common Divisors of Several Polynomials with Linearly Constrained Coefficients and Simgular Polynomials*, Proc. International Symposium on Symbolic and Algebraic Computations, 2006.

[15] N. K. Karmarkar, Y. N. Lakshman, *On Approximate GCDs of Univariate Polynomials*, J. Symbolic Comp. **26**, 653-666 (1998).

[16] B. Li, Z. Yang, 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).

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

[18] V. Y. Pan, *Numerical computation of a polynomial GCD and extensions*, Information and Computation **167**, 71-85 (2001).

[19] A. Schönhage, *Quasi-GCD Computations*, J. Complexity, **1**, 118-137 (1985).

[20] L. B. Rall, *Convergence of the Newton Process to Multiple Solutions*, Num. Math. **9**, 23-37 (1966).

[21] M. Stewart, *Stable Pivoting for the Fast Factorization of Cauchy-Like Matrices*, preprint (1997).

[22] D. R. Sweet, R. P. Brent, *Error analysis of a fast partial pivoting method for structured matrices*, in Adv. Signal Proc. Algorithms, Proc. of SPIE, T. Luk, ed., 266-280 (1995).

[23] Z. Zeng, *The approximate GCD of inexact polynomials Part I: a univariate algorithm*, to appear

[24] L. Zhi, *Displacement Structure in computing the Approximate GCD of Univariate Polynomials*, Mathematics, W. Sit and Z. Li eds., World Scientific (Lecture Notes Series on Computing), 288-298 (2003).

Dario A. Bini
Dipartimento di Matematica, Università di Pisa
e-mail: bini@dm.unipi.it

Paola Boito
Dipartimento di Matematica, Università di Pisa
e-mail: boito@mail.dm.unipi.it