Numerical Computation of a Polynomial GCD and Extensions

Victor Y. Y. Pan

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

NUMERICAL COMPUTATION OF A POLYNOMIAL GCD AND EXTENSIONS

Victor Y. Pan

N° 2969

Août 1996

THÈME 2

ISSN 0249-6399

NUMERICAL COMPUTATION OF A POLYNOMIAL GCD AND EXTENSIONS

Victor Y. Pan*

Thème 2 — Génie logiciel
et calcul symbolique
Projet SAFIR

Rapport de recherche n° 2969 — Août 1996 — 38 pages

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Abstract: In the first part of this paper, we define approximate polynomial gcds (greatest common divisors) and extended gcds provided that approximations to the zeros of the input polynomials are available. We relate our novel definition to the older and weaker ones, based on perturbation of the coefficients of the input polynomials, we demonstrate some deficiency of the latter definitions (which our definition avoids), and we propose new effective sequential and parallel (RNC and NC) algorithms for computing approximate gcds and extended gcds. Our stronger results are obtained with no increase of the asymptotic bounds on the computational cost. This is partly due to application of our recent nearly optimal algorithms for approximating polynomial zeros. In the second part of our paper, working under the older and more customary definition of approximate gcds, we modify and develop an alternative approach, which was previously based on the computation of the Singular Value Decomposition (SVD) of the associated Sylvester (resultant) matrix. We observe that only a small part of the SVD computation is needed in our case, and we also yield further simplification by using the techniques of Padé approximation and computations with Hankel and Bezout matrices. Finally, in the last part of our paper, we show an extension of the numerical computation of the gcd to the problem of computing numerical rank of a Hankel matrix, which is a bottleneck of Padé and Berlekamp-Massey computations, having important applications to coding and transmission of information.

Key-words: polynomial greatest common divisors (gcds) and extended gcds, approximate gcds, Sylvester matrices, subresultant matrices, Hankel matrices, Bezout matrices, Berlekamp-Massey computations, Padé approximations, Hankel numerical rank.

1991 Mathematics Subject Classification: 68Q40, 47B35, 65D99, 68Q25, 65Y20, 65F30.

(Résumé : tsvp)

* Current address: Mathematics and Computer Science Department Lehman College, City University of New York Bronx, NY 10468, USA Internet: VPAN@LCVAX.LEHMAN.CUNY.EDU (Supported by NSF Grant CCR 9020690 and PSC CUNY Awards Nos. 665301 and 666327)

CALCUL NUMERIQUE DE PGCD DE POLYNOMES ET EXTENSIONS

Résumé : Nous définissons les pgcd (plus grand commn diviseur) approchés et généralisés de polynômes en fonction des valeurs approchées des racines des polynômes en entrée. Nous relions cette nouvelle définition aux plus anciennes et plus faibles en termes de perturbation des coefficients des polynômes en entrée ; nous montrons des lacunes des définitions déjà existantes (lacunes qu'évite notre méthode) ; puis nous proposons de nouveaux algorithmes effectifs séquentiels et parallèles (RNC et NC) pour le calcul des pgcd approchés et généralisés. Nos meilleurs résultats n'améliorent pas les bornes asymptotiques du coût des calculs. Ceci est partiellement dû à une application de notre récent algorithme presque optimal pour l'approximation des racines d'un polynôme. Nous étudions aussi et développons une autre approche inspirée des techniques d'approximation de Padé et de calcul de matrices de Hanhel et de Bezout. Nous montrons ensuite une nouvelle extension du problème de calcul numérique de rang de matrice de Hankel, goulet d'étranglement des calculs de Padé et de Berlekamp-Massey, ayant d'importantes applications en traitement du signal, théorie du codage et de la transmission d'information.

Mots-clé : Plus grand diviseur commun de polynômes, pgcd approchés, matrices de Sylvester, matrices de sous-résultants, matrices de Hankel, matrices de Bézout, calculs de Berlekamp-Massey, approximation de Padé, rank numérique de Hankel

1. Introduction

Historically, the fields of algebraic computation and numerical computation have been developed by two distinct groups of people, having relatively little interaction and overlaps with each other. Recently, however, substantial efforts have been spent in order to bridge the gap between the two fields (compare [AS88], [BP94], [C88], [C90], [CGTW95], [H90], [HS,a], [KM94], [KM95], [M94], [M94a], [MC93], [MD92], [MD94], [MD95], [MS95], [NS91], [P87], [P92], [PRT92], [PSLT93], [P94], [P95], [PZDH,a], [R94], [S93], [S93a]). Our present paper should also contribute to these efforts: we study numerical approach to computing polynomial greatest common divisors (gcds) and extended gcds, and we show some further relations and applications to Toeplitz-Hankel computations, Padé approximation, and solution of the Berlekamp-Massey problem.

Computation of polynomial gcds is one of the fundamental problems of algebraic computing. On the other hand, this is an excellent example of numerically ill-posed problems. Consider, for instance, two polynomials $u(x)$ and $v(x)$, where $v(x)$ has a positive degree and divides $u(x)$. Then, gcd $(u(x), v(x)) = v(x)$ has a positive degree, but gcd $(u(x) + \delta, v(x)) = 1$ for any constant $\delta \neq 0$. Thus, even an arbitrarily small perturbation of the input polynomial $u(x)$ may cause a dramatic jump of the output gcd of $u(x)$ and $v(x)$.

The study of this important subject has become popular in recent years due to the pioneering papers [Sc85], [NS91], [KM94], [HS,a], and [CGTW95], in which approximate gcds have been defined so as to avoid the latter deficiency. Namely, for two polynomials $u(x)$ and $v(x)$, for the polynomial norm $\| \sum_i p_i x^i \| = \sum_i |p_i|$, and for a real b, an approximate gcd, $d^*(x) = \gcd^*(u(x), v(x))$, has been nonuniquely defined as gcd $(u^*(x), v^*(x))$, where

$$\deg u^*(x) \leq \deg u(x), \quad \deg v^*(x) \leq \deg v(x) , \tag{1.1}$$

$$\| u^*(x) - u(x) \| \leq 2^{-b} \| u(x) \|, \quad \| v^*(x) - v(x) \| \leq 2^{-b} \| v(x) \|, \tag{1.2}$$

and $u^*(x)$ and $v^*(x)$ satisfying (1.1) and (1.2) are (nonuniquely) chosen so as to maximize deg $d^*(x)$. (In [Sc85], a similar definition applies to bivariate homogeneous polynomials $u(x, z)$ and $v(x, z)$ obtained from $u(x)$ and $v(x)$, so that $u(x, 1) = u(x)$, $v(x, 1) = v(x)$; this allows one to treat degenerated $u(x)$ and $v(x)$, of smaller degrees, by adding their zeros at the infinity.)

The algorithms of [Sc85], [NS91], and [HS,a] for computation of such approximate gcds modify the Euclidean algorithm, which is the classical and most common algorithm for the gcd computation, whereas [CGTW95] relies on computing the Singular Value Decomposition (SVD) of the resultant (Sylvester) matrix, $S = S(u, v)$, associated with the equation

$$f(x)u(x) + g(x)v(x) = d(x), \tag{1.3}$$

where

$$d(x) = \gcd(u(x), v(x)), \tag{1.4}$$

$$\deg\ f(x) + \deg\ d(x) < \deg\ v(x), \tag{1.5}$$

$$\deg\ g(x) + \deg\ d(x) < \deg\ u(x). \tag{1.6}$$

[The triple of $d(x)$, $f(x)$, and $g(x)$ is called *extended gcd* of $u(x)$ and $v(x)$.] The SVD reveals numerical rank r of S, which defines an upper bound d_+ on the degree d^* of $d^*(x)$ [G95]. Assuming that, furthermore, the degree d^* itself is known, one may choose among four approaches to computing $d^*(x)$ listed in [CGTW95]. In particular, according to one of the latter recipes listed in [CGTW95], the cofactor polynomials $f(x)$ and $g(x)$ can be computed from a nonsingular linear system of r equations defined by (1.3), and then, the gcd $d(x)$ is easily obtained from the polynomial equation (1.3). The three other approaches listed in [CGTW95] amount to the Euclidean algorithm, to a least-squares approach, which apparently was included into the list for the sake of completeness since, in its rudimentary form shown in [CGTW95], it leads to severe numerical stability problems ([GL89], Section 3.5.5), and to a version of Lazard's algorithm [L81], which is equivalent to the matrix pencil algorithm of [KM94]. In [KM94] the authors use terminology of automatic control to describe their approach, which involves the resultant and companion matrices associated with the input polynomials $u(x)$ and $v(x)$.

All of the cited papers, except for [Sc85], only present heuristic algorithms that may fail for a large class of input pairs of polynomials. Furthermore, the papers give no recipe for certification that the output is, indeed, a desired approximate gcd. Neither of the papers, including [Sc85], shows a good parallel algorithm for approximate gcds, and in fact, the power of parallelism seems to be inherently limited, at least, for the Euclidean algorithm approach.

The record sequential estimate for the number of bit-operations required for computing such an approximate gcd is $\tilde{O}((b+n)n^2)$. [Here and hereafter, $\tilde{O}(f(b,n))$ denotes $O(f(b,n)\log^c(bn))$ for some constant c, that is, denotes the values of order $f(b,n)$ up to a polylogarithmic factor in bn.] More precisely, the latter bit-cost bound has been obtained (in [Sc85]) under the assumptions that

$$m = \deg\, v(x) < \deg\, u(x) = n,$$
$$0.5 <\| u(x) \|< 1, \quad 0.5 <\| v(x) \|< 1,$$

and all the zeros of $u(x)$ lie in the disc $\{x:\ |x| \le 0.25\}$.

The next example shows another problem with all the cited approaches. Namely, the idea of shifting from the input polynomials $u(x)$ and $v(x)$ to their 2^{-b}-neighbors appears to be quite natural but may lead to some undesirable consequences.

Example 1.1. Let $b = n$ be large, $u(x) = x^n$, $v(x) = (x-0.5)^{n-1}$. Then $\gcd\,(u(x), v(x)) = 1$. Moreover, the distance between the only zero $x = 0$ of $u(x)$ and the only zero $x = 0.5$ of $v(x)$ is large enough to suggest that, under any reasonable definition, an approximate gcd of $u(x)$ and $v(x)$ should be a constant too. Under the cited definition of [Sc85], [NS91], [KM94], [HS,a], and [CGTW95], however, such an approximate gcd equals $x - 0.5$, since we may choose, say, $u^*(x) = u(x) - 2^{-n}$ and $v^*(x) = v(x)$. Thus, perturbation of $u(x)$ by 2^{-b} (for $b = n$) may cause a gigantic jump of an approximate gcd if it is defined as above.

Our present paper extends the study of numerical computation of the gcd into three directions. Pursuing our first direction, we impose some requirements on approximate gcds that both imply (1.2) (cf. our Lemma 3.1) and enable us to exclude the above undesired phenomena of amplification of small input perturbations, heuristic character of the computation and absence of the output certification. In spite of such stronger requirements, we will still compute an approximate gcd, for any pair of input polynomials $u(x)$, $v(x)$, by using $\tilde{O}((b+n)n^2)$ bit-operations, as under the assumptions of [Sc85]. In terms of the number of arithmetic operations and comparisons involved, we reach the bound $\tilde{O}(n^2)$ (see the end of Section 4).

Moreover, unlike the previously known algorithms, our algorithms allow their substantial parallel acceleration. Namely, they only require to use polylogarithmic parallel time and polynomial number of processors (thus being

in NC or RNC), under the customary PRAM models of parallel computing, which we will assume when we state our parallel complexity estimates. (The reader is referred to [KR90], [BP94] on definitions of PRAM models, NC and RNC.) Furthermore, except for some auxiliary computation of matchings or connected components in graphs, our algorithms only require some computations with polynomials, which are essentially reduced to application of FFTs on the n-th roots of 1, and such an FFT only requires $O(\log n)$ time and similtaneously n processors, under PRAM models, as well as under some more realistic models of parallel computer architecture, such as hypercube, butterfly, and shuffle-exchange processor array models (see e.g. [Le92], Section 3.7, or [Q94], Chapter 8).

Besides various computational advantages, our approach provides an alternative insight into the problem of numerical computation of the gcds since the essence of the problem is most clearly revealed via the study of the correlations between the perturbation of the zeros of the input polynomials and the perturbation of their gcd.

The approach of our present paper that has lead us to the cited results on approximate gcds relies on the reduction of the problem to approximating polynomial zeros, where we apply the recent effective algorithms of [P95], [P96] (cf. Remark 3.2 in Section 3). In spite of high attention of the researchers to polynomial zeros and gcd, it seems that the latter direction, involving the computation of maximum or maximal matchings and/or connected components in bipartite graphs, has never been explored.

The second direction of our study is described in Sections 6 and 7, where we relate the gcd of $u(x)$ and $v(x)$ to Padé approximation of the formal power series defined by the ratio $u(1/x)/v(1/x)$ and to the associated Hankel matrices and Bezout linear systems. While developing this approach, we demonstrate its four substantial advantages over the earlier SVD approach of [CGTW95] (in particular, in terms of both numerical stability and computational cost). Namely, we avoid computation of the SVD, since, in our case, we only need to compute the sign sequence for the values of the characteristic polynomials of the associated matrix and its leading principal submatrices, which only amounts to one of the several stages of the customary algorithms for the SVD (cf. [GL89], [Par80]). Moreover, unlike the SVD computation (which involves irrational values even where the input is integer), our modification enables us

to perform all our computations by using rational arithmetic (with no roundoff errors) and a few comparisons (cf. Remark 7.1 in Section 7). Furthermore, in the major case where the input is real, the computations can be further simplified since we operate with Hankel and Bezout matrices, which are symmetric, unlike Sylvester matrices used in [CGTW95]. Finally, application of Bezout matrices enables us to improve numerical stability of the computations dramatically (cf. [BP94], Sections 9 and 10 of Chapter 2). It should be noted that in our second approach, of Sections 6 and 7, we have yielded *model independent* improvements of the earlier approach of [CGTW95], since in these sections we have assumed a customary definition of approximate gcd, used in [CGTW95]; this assumption has facilitated the comparison of our approach with the one of [CGTW95] but has also implied that, in these sections, like in [CGTW95], we have only computed an upper bound d_+ on the maximum degree d^* of approximate gcds, instead of computing the value d^* itself. Unlike [CGTW95], however, we have supplemented (at the end of our Section 7) an extension of our approach of Sections 6 and 7 to computing an approximate gcd of degree d^*, based on a certification algorithm that relied on a simplified application of our techniques of Section 3 (cf. Remark 3.3).

Our third approach, presented in Section 8, may have substantial practical impact. Here, our goal is the computation of the *Hankel numerical rank of a Hankel matrix* H. We use this term in order to define the minimum rank of Hankel matrices H^* that approximate H sufficiently closely (under a fixed rule of measurement). The latter computation is the bottleneck of numerical computation of Padé approximation of an analytic function, which is computationally equivalent to numerical recovery of the coefficients of a linear recurrence (of a length at most n) from its $2n$ first terms, known as the Berlekamp-Massey problem (see [Gr72], [BGY80], and Problems 1.5.2b and 1.5.3 and their solution algorithms in [BP94]). The Padé-Berlekamp-Massey computations have many highly important applications in the areas of symbolic computing (for instance, to sparse multivariate polynomial interpolation [BP94], Section 1.9), signal and image processing, and coding theory and practice (in particular, Berlekamp-Massey computations are crucial stages of the linear feedback register synthesis and the BCH decoding [Be68], [BGY80]). In Section 8, we reverse the direction of our Sections 6 and 7 and show an extension of our first approach (as well as of any black box algorithm for numerical polynomial

gcds) to designing effective algorithms for the computation of Hankel numerical ranks of Hankel matrices. The algorithm of Section 8 is immediately extended to computing the Bezout, Toeplitz, and Sylvester numerical ranks of the Bezout, Toeplitz, and Sylvester matrices, respectively.

Our approach of Section 8 has some distant technical resemblance to one stage of one of the algorithms of [P96] for splitting a polynomial into two factors over a fixed annulus free of the zeros of the input polynomial. In the present paper, however, we reverse the direction of [P96], where the gcd and Padé computations are used as auxiliary tools for approximating polynomial zeros.

More recently, the present author has learned about two other interesting papers on approximate gcds, [EGL96] and [KL96]. In the former paper, a lower bound on the degree d^* of the approximate gcds has been computed via Euclidean algorithm, and an upper bound has been computed via SVD of the associated Sylvester matrix; for a large class of input pairs of polynomials (though not for all inputs) these two bounds coincide with each other, which completes the soluton. In the latter paper, quadratic programming techniques (which can be viewed as an extension of the least-squares approach of [CGTW95]) are applied in order to compute the degree d^* itself, as in our present work, though the computational cost bound, stated in [KL96] in terms of the number of the arithmetic operations involved, is a polynomial of a higher (unspecified) degree in $m+n$ and is exponential in d^*, that is, this cost bound is generally much higher than the arithmetic bound $\tilde{O}((m+n)^2)$ of our paper. In both papers [EGL96] and [KL96], approximate gcds have been defined based on the relations (1.1), (1.2).

In the next six sections, starting with some preliminaries in Section 2, we will describe and analyze our first two approaches. In Section 8, we will show our extension of the second presented approach to computing Hankel numerical rank of a square Hankel matrix. Section 9 is left for a summary and brief discussion.

Acknowledgements. I gratefully acknowledge receiving reprints of [CGTW95] from Andre Galligo and Erich Kaltofen and a preprint of [HS,a] from the former. My present work was substantially motivated by Andre Galligo's comments on [CGTW95], delivered in [G95], and partly by Erich Kaltofen's interest to the computaton of numerical rank of a Toeplitz matrix.

2. GCDs of Polynomials Represented by Their Zeros

Suppose that we have precomputed all the zeros of the input polynomials $u(x)$ and $v(x)$ so as to represent these polynomials as the products of linear factors:

$$u(x) = u \prod_{i=1}^{h} (x - y_i)^{u_i}, \tag{2.1}$$

$$v(x) = v \prod_{j=1}^{k} (x - z_j)^{v_j}, \tag{2.2}$$

for complex u, v, y_i, and z_j and for positive integers u_i and v_j, where $y_1, \ldots, y_h$ are pairwise distinct, $z_1, \ldots, z_k$ are pairwise distinct, and $n = \sum_i u_i > m = \sum_j v_j$. Then, we may compute a unique (up to scaling) polynomial, gcd $u(x), v(x))$, as follows:

Algorithm 2.1, *computation of gcd.*

Input: positive integers $h, k, u_1, \ldots, u_h, v_1, \ldots, v_k$, and complex $u, v, y_1, \ldots, y_h$, $z_1, \ldots, z_k$, where $n = \sum_i u_i > m = \sum_j v_j$.

Output: $d(x) = \gcd\,(u(x), v(x))$ for $u(x)$ and $v(x)$ defined by (2.1) and (2.2).

Computation:

1. Do for $j = 1, \ldots, k$

   if there exists $i = i(j)$ such that $y_i = z_j$,

   then store the pair of z_j and $\mu_j = \min(u_i, v_j)$.

2. Compute and output the coefficients of the polynomial $d(x) = \prod_j (x - z_j)^{\mu_j}$, where the product is over all j, for which the pairs of z_j and μ_j have been stored. (If no such pairs have been stored, output $d(x) = 1$.)

Correctness of the algorithm is immediately verified.

Let us estimate the *computational cost* of performing it.

Hereafter, $O(t, p)$ denotes the simultaneous bounds $O(t)$ on time and $O(p)$ on the number of processors involved, where we allow performing an arithmetic operation or a comparison by any processor in unit time and where we assume

that $O(t, sp)$ implies the bound $O(st, p)$ for any $s > 1$ [KR90], [BP94], so that, in particular, the bound $O(t, p)$ also implies the bound $O(tp, 1)$, that is, the sequential time bound $O(tp)$.

Stage 1 of Algorithm 2.1 can be performed (in a single parallel step) by using at most $hk \leq mn < n^2$ comparisons, that is, at the cost $O(1, n^2)$.

Stage 2 can be reduced to recursive polynomial multiplications that can be performed by using $O(m \log^2 m)$ arithmetic operations executed in $O(\log^2 m)$ parallel steps (compare, e.g. [BP94]), that is, at the cost $O(\log^2 m, m)$.

3. GCDs of Polynomials Represented by Approximations to Their Zeros

Now, assume that the input polynomials $u(x)$ and $v(x)$ are represented by their zeros y_i and z_j, defined within a fixed absolute error bound δ, that is, instead of $u(x)$ and $v(x)$, we are given a class of pairs of approximation polynomials, $\tilde{u}(x)$ and $\tilde{v}(x)$, with their zeros lying in the δ-neighborhoods of y_i and z_j, respectively. Then, the gcds, $\tilde{d}(x)$, of all such pairs $\tilde{u}(x)$ and $\tilde{v}(x)$ will be called *δ-gcds* of $u(x)$ and $v(x)$, and among them, we will seek ones of the maximum degree, d_δ, which we will call *maximum δ-gcds*. Alternatively, one may seek a *maximal δ-gcd*, which divides no δ-gcd of a higher degree. For $\delta = 0$, both classes coincide, but not always so in the case of a positive δ.

Let us next relate δ and b, for which polynomials with the zeros in the δ-neighborhood of the zeros of a given polynomial have their coefficients in the 2^{-b}-neighborhood of its coefficients [compare (1.2)].

Lemma 3.1. *Let $u(x)$ and $\tilde{u}(x)$ denote two polynomials of a degree n, with the same leading coefficient u and with their zeros y_i and $\tilde{y}_i$, respectively, such that*

$$|y_i - \tilde{y}_i| \leq \delta, \qquad i = 1, 2, \ldots, n \ .$$

Then

$$\| \tilde{u}(x) - u(x) \| \leq \| u(x) \| ((1 + \delta)^n - 1) \ .$$

Proof. The value $\| \tilde{u}(x) - u(x) \| / \| u(x) \|$ reaches its maximum for $u(x) = u \prod_{i=1}^{n} (x - y_i), \tilde{u}(x) = u \prod_{i=1}^{n} (x - y_i - \delta) = u(x - \delta)$ and for some nonne-

gative y_i, $i = 1, \ldots, n$. For such $u(x)$ and $\tilde{u}(x)$, we have

$$\| \tilde{u}(x) - u(x) \| = \| \sum_{i=1}^{n} u^{(i)}(x)\delta^i/i! \| \leq \sum_{i=1}^{n} \| u^{(i)}(x)/i! \| \delta^i \leq$$

$$\| u(x) \| \sum_{i=1}^{n} \delta^i \begin{pmatrix} n \\ i \end{pmatrix} = \| u(x) \| ((1+\delta)^n - 1) .$$

□

In particular, if

$$\delta \leq (1 + 2^{-b})^{1/n} - 1 , \tag{3.1}$$

then $(1+\delta)^n \leq 1 + 2^{-b}$, and consequently,

$$\| \tilde{u}(x) - u(x) \| \leq 2^{-b} \| u \| .$$

Similarly, we have $\| \tilde{v}(x) - v(x) \| \leq 2^{-b}$, thus satisfying (1.1) and (1.2) for $u^*(x) = \tilde{u}(x)$ and $v^*(x) = \tilde{v}(x)$, having their zeros in the δ-neighborhoods of the respective zeros of $u(x)$ and $v(x)$ for δ of (3.1).

Corollary 3.2. *Let $\delta \leq (1+2^{-b})^{1/n} - 1$. Then, any δ-gcd of $u(x)$ and $v(x)$ is a gcd of two polynomials $u^*(x)$ and $v^*(x)$ satisfying* (1.1) *and* (1.2), provided that *deg $u(x) \leq n$, deg$v(x) \leq n$,* provided that *deg $u(x) \leq m$, deg $v(x) \leq n$.*

Next, we will extend Algorithm 2.1 to computation of $\tilde{d}(x) = \tilde{d}_\delta(x)$, a maximum δ-gcd of $u(x)$ and $v(x)$.

Algorithm 3.1, *computation of a maximum δ-gcd.*

Input: positive δ and complex $u, v, y_1, \ldots, y_n, z_1, \ldots, z_m$, $m < n$.

Output: coefficients of a maximum δ-gcd, $\tilde{d}_\delta(x)$, of the two polynomials,

$$u(x) = u \prod_{i=1}^{n} (x - y_i), \quad v(x) = v \prod_{j=1}^{m} (x - z_j) . \tag{3.2}$$

Computation:

1. For all pairs (i, j), $i = 1, \ldots, n$, $j = 1, \ldots, m$, test if $|y_i - z_j| \leq 2\delta$. If so, store the pair (i, j).

2. Define a bipartite graph, G, by two sets of its vertices, $Y = \{y_1, \ldots, y_n\}$ and $Z = \{z_1, \ldots, z_m\}$, connected by an edge (y_i, z_j) if and only if the pair (i, j) has been stored at Stage 1. In this graph, compute a maximum matching $(y_{i_1}, z_{j_1}), \ldots, (y_{i_r}, z_{j_r})$, that is, one having the maximum cardinality, r.

3. Compute and output the coefficients of $\tilde{d}(x) = \tilde{d}_\delta(x)$, a δ-gcd of $u(x)$ and $v(x)$, defined by the equations

$$\tilde{d}_\delta(x) = \prod_{q=1}^{r}(x - x_q), \quad x_q = (y_{i_q} + z_{j_q})/2, \quad q = 1, \ldots, r \ .$$

Correctness of the algorithm is immediately verified.

Let us estimate the *computational cost* of performing this algorithm.

Stage 1 involves less than n^2 subtractions and as many comparisons, which all can be performed in two parallel steps, that is, at the cost $O(1, n^2)$.

Stage 2 can be performed sequentially by using $O(n^{2.5})$ comparisons [HK73], that is, at the cost $O(n^{2.5}, 1)$, or by using $O(\log^3 n)$ parallel steps that perform $O(n^{3.38})$ comparisons and arithmetic operations [GP88], [CW90], that is, at the cost $O(\log^3 n, n^{3.38})$.

Stage 3 is similar to Stage 2 of Algorithm 2.1 and has the same cost estimates, that is, $O(n \log^2 n)$ arithmetic operations that can be performed in $O(\log^2 n)$ parallel steps.

The precision of the values involved in this computation (for $\delta = 2^{-B}/(\|u(x)\| + \|v(x)\|)$) is bounded by $O(B)$ at Stage 1, by $O(\log n)$ at Stage 2, and by $O(B\tilde{d}_\delta)$, $\tilde{d}_\delta = \deg \tilde{d}_\delta(x)$, at Stage 3. Under these bit-precision bounds, the cited complexity estimates are easily translated into Boolean (bit-) complexity bounds. In particular, the overall sequential bit-operation cost of performing Algorithm 3.1 is bounded by $\tilde{O}(n^2 B + n^{2.5})$.

If one is satisfied with having a maximal (rather than maximum) δ-gcd, then Algorithm 3.1 can be simplified since at its Stage 2, one will only need to compute a maximal (rather than maximum) matching, that is, a matching not being a subset of a larger matching in G. A maximal matching can be computed by using $O(n^2 \log^3 n)$ comparisons performed in $O(\log^3 n)$ parallel steps [IS86], that is, at the cost $O(\log^3 n, n^2)$, or with randomization (under the CRCW PRAM models) in $O(n^2 \log n)$ comparisons performed in $O(\log n)$ parallel steps [II86], that is, at the cost $O(\log n, n^2)$.

Remark 3.1. δ-gcd of $u(x)$ and $v(x)$ is constant if and only if the graph G has no edges, so Stage 2 of Algorithm 3.1 can be removed if one only needs to test numerically whether $u(x)$ and $v(x)$ are relatively prime.

Remark 3.2 Let us recall the complexity estimates for approximating polynomial zeros, which can be viewed as a preconditioning stage for the com-

putations of this and the two next sections. We will recall that $\deg u(x) = n > \deg v(x) = m$ and will further assume that all the zeros of $u(x)$ and $v(x)$ lie in the unit disc $\{x : |x| = 1\}$. (The latter assumption can be satisfied by means of scaling the variable x, since the maximum distance from the origin to the zeros of $u(x)$ and $v(x)$ is readily estimated [P95a], [P96].) Then, the algorithm of [P95a], [P96] enables us to approximate all the zeros of $u(x)$ and $v(x)$ within $\delta = 2^{-B}$, at the cost of involving $O(n)$ arithmetic operations or $\tilde{O}((B+n)n^2)$ bit-operations, which all can be performed in polylogarithmic parallel arithmetic time or in polylogarithmic Boolean time, respectively.

Remark 3.3 The pairwise distances between the zeros of the polynomials $u(x)$ and $v(x)$ can be quite easily approximated even if we only know approximations to the zeros of one of the input polynomials. This is because an effective algorithm is available that enables us to compute approximations, d_j^*, to the distances, d_j, $j = 1, \cdots, k$, from a fixed complex point simultaneously to all the k zeros of a given polynomial of a degree k (cf. [P87] or Appendix B of [P96b]). In particular, such approximations within the relative error bound $E(k) = c/k^e$, for two fixed constants $c > 0, e \geq 0$ [so that $d_j \leq d_j^* \leq (1 + E(k))d_j$, $j = 1, \cdots, k$], can be computed by using $O(k \log^2 k)$ arithmetic operations.

4. A Simplified Algorithm for Approximate GCDs

Next, we will simplify the computations at Stage 2 of Algorithm 3.1, at the price of relaxing some restrictions on the output polynomial $\tilde{d}(x)$. Namely, we choose $\tilde{d}(x)$ among the $(\mu\delta)$-gcds of $u(x)$ and $v(x)$, for some μ satisfying $1 \leq \mu \leq 4n - 6$, but we only require that $\deg \tilde{d}(x) \geq d_\delta$, that is, the degree of $\tilde{d}(x)$ is required to reach (or to exceed) the maximum degree, d_δ, of the δ-gcds of $u(x)$ and $v(x)$, but is not required to reach $d_{\mu\delta}$, the maximum degree among the $(\mu\delta)$-gcds. Later on in this section, we will extend this approach to computation of a maximum δ_1-gcd, for an appropriate δ_1.

Algorithm 4.1, *computation of a $(\mu\delta)$-gcd.*

Input and Stage 1 of the computation are as in Algorithm 3.1.

Output: a real μ, $1 \leq \mu \leq 4n - 6$, and the coefficients of a $(\mu\delta)$-gcd of the polynomials $u(x)$ and $v(x)$ of (3.2), denoted $\tilde{d}(x)$ and having a degree of at least d_δ.

Stage 2. Define a bipartite graph $G = G_\delta$ as at Stage 2 of Algorithm 3.1. Then, compute all connected components of G, ignoring only singletones that have less than two vertices. For each component C_q, $q = 1, \dots, s$, having at least two vertices, let the two sets Y_q and Z_q, $Y_q \subseteq Y$, $Z_q \subseteq Z$, denote the partition of the vertex set of C_q induced by the partition of the vertices of G into the two sets Y and Z. Letting $|T|$ denote the cardinality of a set T, write

$$\nu_q = |Y_q|, \quad K_q = Y_q \quad \text{if} \quad |Y_q| \le |Z_q| \ , \tag{4.1}$$

$$\nu_q = |Z_q|, \quad K_q = Z_q \quad \text{if} \quad |Y_q| > |Z_q| \ , \tag{4.2}$$

for $q = 1, \dots, s$. Compute and output $\mu = \mu(G) = 4 \max_{q=1,\dots,s} \nu_q - 2$, $K = K(\delta) = \cup_{q=1}^{s} K_q$.

Stage 3. Compute and output the coefficients of the polynomial

$$\tilde{d}(x) = \prod_{x_k \in K} (x - x_k) \ ,$$

which is a $(\mu\delta)$-gcd of $u(x)$ and $v(x)$ and has a degree of at most d_δ.

To verify *correctness of Algorithm 4.1*, observe that every simple path between Y_q and Z_q, with all edges lying in (a component C_q of) the bipartite graph G, consists of at most $2\nu_q - 1$ edges (by the definition of ν_q). On the other hand, each edge connects two vertices of G lying at a distance at most 2δ from each other. Therefore, if $x_k \in Y_q$ (if $x_k \in Z_q$, respectively), then any point of Z_q (respectively, of Y_q) lies at a distance at most $2(2\nu_q - 1)\delta \le \mu\delta$ from x_k. Finally, note that $\max_q \nu_q \le \sum_q \mu_q = n - 1$, by the definition of ν_q, and that, consequently, $\mu = 4 \max_q \nu_q - 2 \le 4n - 6$.

Clearly, the estimates for the *computational cost* of performing Stages 1 and 3 of Algorithm 3.1 also apply to the case of Algorithm 4.1, but Stage 2 (and, particularly, its parallel version) is substantially simplified, which implies respective simplification of Algorithm 3.1, particularly, of its parallel version. The computational cost of performing Stage 2 of Algorithm 4.1 is dominated by the cost of computing the connected components of the bipartite graph G. This computation only requires $O(\log^2 n)$ parallel comparison steps, which perform $O(n^2)$ comparisons of integers ranging from 1 to $m + n$, $m < n$ [J92], so the Boolean (bit) complexity of performing this stage is bounded by $O((\log^2 n) \log\log n, (n^2 \log n) \log\log n)$.

Next, we will extend Algorithm 4.1 to the computation of a maximum δ_1-gcd, for an appropriate δ_1. We may assume that the graph G has $s_0 < m+n-2$ components, for otherwise, every component of G would have had at most three vertices, and then we would have easily computed a maximum δ-gcd of $u(x)$ and $v(x)$. Now, suppose that we have performed Algorithm 4.1. Then, we write $\mu_1 = \mu$ and compute the graph $G^{(1)} = G_{\delta\mu_1}$, that is, the graph G of Stage 2 of Algorithm 4.1 for δ replaced by $\delta\mu_1$. In this case, for every q, every pair of vertices $y \in Y_q$, $z \in Z_q$ of the component C_q of the original graph $G = G^{(0)} = G_\delta$ is connected by an edge in the new graph $G^{(1)} = G_{\delta\mu_1}$, that is, the component C_q turns into a bipartite clique. If the graph $G^{(1)}$ has as many connected components as $G^{(0)}$ has, then, clearly, the output polynomial $\tilde{d}(x)$ of Algorithm 4.1 is a maximum $(\delta\mu_1)$-gcd of $u(x)$ and $v(x)$. Otherwise, the graph $G^{(1)}$ has fewer components than G. In this case, we may recursively compute the graphs $G^{(j)} = G_{\delta\mu_j}$ and the values $\mu_j = 4\max_q \nu_q(G^{(j-1)}) - 2$, where the values $\nu_q(G^{(j-1)})$ are defined by the relations (4.1) and (4.2) extended to the graph $G^{(j-1)}$ and where the maximum is over all connected components of $G^{(j-1)}$, $j = 2, 3, \ldots$, until (in at most $J \le s_0 < m+n-2$ steps) we arrive at a pair of graphs $G^{(J-1)}$ and $G^{(J)}$ having the same number of components. Then, we perform Stage 3 of Algorithm 4.1 for $K = K_{\delta\mu_J}$ and output $\tilde{d}(x) = \tilde{d}_{\delta\mu_J}(x)$, a maximum $(\delta\mu_J)$-gcd of $u(x)$ and $v(x)$. We will refer to the resulting algorithm as to **Algorithm 4.2.** The deterministic complexity of its performance is bounded by $O(J\log^2 n, n^2)$, where $J \le m+n-3$.

Since $\mu(G^{(j)}) \le 4n-6$ for all j, $j = 1, 2, \ldots, J$, $J \le m+n-3$, we may alternatively set $\mu(G^{(j-1)}) = 4n-6$, $\delta_j = (4n-6)^j\delta$, $j = 1, 2, \ldots, m+n-3$, and compute the graphs $G^{(j)} = G_{\delta\mu_j}$ for all j concurrently. Then, we choose J as the minimum j for which the graphs $G^{(j)}$ and $G^{(j-1)}$ have the same number of connected components, apply Stage 3 of Algorithm 4.1 for $K = K_{\delta\mu_J}$, and output $\tilde{d}(x) = \tilde{d}_{\delta\mu_J}(x)$, $\mu_J = (4n-6)^J$, a maximum δ_1-gcd of $u(x)$ and $v(x)$, for $\delta_1 = \delta\mu_J \le (4n-6)^J\delta \le (4n-6)^{m+n-3}\delta$.

We will cite the resulting algorithm as **Algorithm 4.3**, for computation of a maximum δ_1-gcd. Its *correctness* follows from the preceding argument. The *complexity* of its performance is the same as for Algorithm 4.1 plus the cost of construction of the additional bipartite graphs $G^{(1)}, \ldots, G^{(J)}$ and of computing their connected components. Therefore, the processor bounds of performing Stages 1 and 2 are multiplied by $m+n-3$, versus Algorithm 4.1,

whereas the time bounds at all Stages 1, 2 and 3 and the processor bound at Stage 3 do not change, that is, we have the next deterministic bounds on the overall cost, in terms of the number of arithmetic operations and comparisons involved: $O(1, n^3)$ at Stage 1, $O(\log^2 n, n^3)$ at Stage 2, and $O(\log^2 m, m)$ at Stage 3 [we keep assuming that $m < n$].

Next, let us modify Algorithm 4.3 by applying binary search at Stage 2 in order to decrease its cost bound to the levels $O(\log^3 n, n^2)$ (deterministic) and $O(\log^2 n, n^2)$ (randomized). Let $E^{(h)}$ denote the edge set of the bipartite graph $G^{(h)}$ and let s_h denote the number of connected components of this graph, for $h = 0, 1, \cdots, n-2$. Observe that $E^{(h)}$ is a subset of $E^{(h+1)}$ and that $1 \le s_{h+1} \le s_h \le m+n-3$ for all h. At the first substage of Stage 2, which we will cite as Substage 2.1 of the entire algorithm, for every pair $(i,j), i = 1, \cdots, m;\ j = 1, \cdots, n$, we apply binary search in order to compute the integer $l \ge 0$ such that

$$2(2n-6)^l\delta < |y_i - z_i| \le 2(2n-6)^{l+1}\delta$$

and include the edge (y_i, z_j) into the edge set of the bipartite graph $G^{(h)}$ if and only if $h > l$.

Then, at the second substage of Stage 2, which we will cite as Substage 2.2 of the entire algorithm, we compute s_0 and s_{m+n-3}, observe that $0 \le s_0 - s_{m+n-3} < m+n-3$, and apply binary search in order to compute the minimum k such that for some h, we have $0 \le s_h - s_{k+h} < k$. Clearly, $k \ge 1$, and if $k = 1$, then, for the corresponding h, we have $s_h = s_{h+1}$, so that $\widetilde{d}(x) = \widetilde{d}_{\delta\mu_h}(x)$ is a maximum $(\delta\mu_h)$-gcd of $u(x))$ and $v(x)$, for $\delta\mu_h = (4n-6)^h\delta$. We will cite such an entire modification of Algorithm 4.3, for computing a maximum $(\delta\mu_h)$-gcd of $u(x))$ and $v(x)$, based on the binary search, as **Algorithm 4.4.** Its cost (in terms of the number of arithmetic operations and comparisons involed) is $O(\log n, n^2)$ at Stage 1, $O(\log^3 n, n^2)$ at Stage 2 (including Substages 2.1 and 2.2), and $O(\log^2 m, m)$ at Stage 3, where $m < n$.

Finally, we may replace computing the connected components of $G^{(h)}$ by computing a maximal matching $M^{(h)}$ in $G^{(h)}$. The cardinality $|M^{(h)}|$ of $M^{(h)}$ never decreases in the transition from $G^{(h)}$ to $G^{(h+1)}$ (since all the components of $G^{(h)}$ turn into bipartite cliques in this transition). Furthermore, $M^{(h+1)}$ is a maximum matching in $G^{(h+1)}$ if $|M^{(h)}| = |M^{(h+1)}|$, and then, a $[(4n-6)^{h+1}\delta]$-gcd is computed immediately. Therefore, we may use $|M^{(h)}|$ as the binary search

parameter, instead of the number of connected components, s_h. In this case, we may apply the randomized algorithm of [II86] in order to decrease (by factor $\log n$, versus the case of Algorithm 4.4) our parallel and sequential bounds on the time-complexity of Stage 2, assuming randomized CRCW PRAM model in the case of parallel computing.

5. Computation of Extended GCDs

Let us next show the algorithms and complexity estimates for the computation of the coefficients of the cofactors $f(x)$ and $g(x)$ of (1.3)–(1.6). By equating the coefficients of the largest $m+n$ powers of x on both sides of (1.3), we obtain a linear system of $m+n$ equations, with an $(m+n)\times(m+n)$ Sylvester coefficient matrix $S=S(u,v)$, having rank $r=m+n-2d$, where $n=$ deg $u(x)$, $m=$ deg $v(x)$, $d=$ deg $d(x)$, $d(x)=$ gcd $(u(x),v(x))$. To compute the coefficients of $f(x)$ and $g(x)$ for given $u(x)$ and $v(x)$, we may first compute r (and d), which enables us to obtain the $r\times r$ nonsingular subresultant submatrix R of S. Then, it will remain to solve a nonsingular linear system of r equations with the coefficient matrix R (compare [BP94], Section 8 of Chapter 2, or [G84]).

Of course, if the degree d of the gcd is available, we may skip the first, most costly, stage, of rank computation.

If, besides, the gcd itself is available, then we may also obtain a little simpler linear system. Indeed, if we know $d(x)$, we may compute the polynomials

$$q(x)=u(x)/d(x) \quad \text{and} \quad t(x)=v(x)/d(x)$$

and then shift from (1.3) to the polynomial equation

$$f(x)q(x)+g(x)t(x)=1\ . \tag{5.1}$$

(To compute $q(x)$ and $t(x)$, we may evaluate $u(x)$, $v(x)$, and $d(x)$ at all 2^k-th roots of 1, $k=1+\lfloor\log_2\max(m-d,n-d)\rfloor$, via FFT, then compute $q(x)$ and $t(x)$ at these roots of 1 [via divisions], and finally obtain $q(x)$ and $t(x)$ by means of interpolation, performed via inverse FFT.) By equating the coefficients of the powers of x on both sides of (5.1), we will arrive at a nonsingular Sylvester linear system of $r=m+n-2d$ equations, which is slightly simpler than a subresultant linear system.

The known algorithms enable us to solve any nonsingular Sylvester or subresultant system of $O(n)$ equations by using $O(n \log^2 n)$ arithmetic operations or at the parallel cost $O(\log^2 n, n^2/\log n)$ [P92a] or [BP94]. If all the input values are integers bounded from above by some fixed value A and if rounding-off a real value to the closest integer can be performed (by any processor) in unit time, then the parallel cost bound $O((\log n)\log(n \log A), n \log n)$ can be achieved [P96a], by using computations with $O(n \log(nA))$-bit precision.

On the other hand, if the zeros of one of the input polynomials $q(x)$ and $t(x)$ are available and pairwise distinct, then we may compute the cofactors $f(x)$ and $g(x)$ at the cost $O(\log^2 n \log^* n, n/\log^* n)$, where $\log^* n = \max\{h, \log^{(h)} n \geq 0\}$, $\log^{(h)} n = \log\log^{(h-1)} n, h = 1, \ldots, \log^* n, \log^{(0)} n = n$.

Indeed, let, the coefficients of $q(x)$ and $t(x)$ be given and also let the zeros $z_1, \ldots, z_m$ of $t(x)$ be available and pairwise distinct. Then we may apply the following algorithm for computing the coefficients of $f(x)$ and $g(x)$:

Algorithm 5.1. Successively evaluate:

1. $q(z_i)$ for $i = 1, \ldots, m$,
2. $f(z_i) = 1/q(z_i)$ for $i = 1, \ldots, m$,
3. the coefficients of $f(x)$,
4. $f(a\omega^i)$, $q(a\omega^i)$, $t(a\omega^i)$, $i = 0, \ldots, H-1$, where $\omega = \exp(2\pi\sqrt{-1}/H), \ell < H = 2^h \leq 2\ell$, $\ell = \deg f(x)$, a is a fixed (large) constant such that $t(a\omega^i) \neq 0$ for all i.
5. the values $g(a\omega^i) = (1 - f(a\omega^i)q(a\omega^i))/t(a\omega^i)$, $i = 0, \ldots, H-1$,
6. the coefficients of $g(x)$.

The known algorithms (see [BP94]) enable us to perform Stages 1–6 at the claimed computational cost. (Note that Stage 1 amounts to multipoint polynomial evaluation, Stage 3 is interpolation, Stages 4 and 6 are reduced to applying forward and inverse FFTs on a set of H points.)

If, say, the zeros $z_1, \ldots, z_{m-d}$ of $t(x)$ are available but some of them are multiple (with multiplicities ranging from 1 to ℓ), then, for each $j, 1 \leq j \leq m-d$, we may successively, $\ell - 1$ times, differentiate both sides of (5.1) and obtain a

triangular linear system of ℓ equations in the values at $x = z_j$ of $f(x)$ and its derivatives $f^{(i)}(x)$ of orders ranging from 1 to $\ell-1$ (with the coefficients of the k-th equations of the form $h_{ik}q^{(i)}(z_j)$, $i = 0, 1, \ldots, \ell-1$, for some fixed integers h_{ik}). As soon as we obtain the values of all $f^{(i)}(x)$ for x equal to the respective zeros of $t(x)$ (this takes $O(n\ell)$ arithmetic operations), we may extend Algorithm 5.1 without changing its overall complexity estimates, since at Stages 1 and 3, we will arrive at the problems of computing a modular representation of a polynomial and of Hermite interpolation, respectively, which extend the problems of multipoint polynomial evaluation and interpolation and have the same overall asymptotic estimates for the complexity of their solution (see the solution of Problems 1.4.1 and 1.4.2b in [BP94], Chapter 1).

Finally, we will consider the extension of Algorithms 3.1 and 4.1–4.3 to computing the extended gcds under a model of approximate computing. Under such a model, the extended algorithms compute a δ-gcd or a $(\mu\delta)$-gcd of $u(x)$ and $v(x)$, respectively. The input to these algorithms includes approximations to the zeros of $u(x)$ and $v(x)$, among which we select approximations to the zeros of $q(x)$ and $t(x)$, respectively, as a by-product of computing $\tilde{d}(x)$. Let, for notational convenience, the latter approximations to the zeros be denoted $\tilde{y}_1, \ldots, \tilde{y}_k$, and $\tilde{z}_1, \ldots, \tilde{z}_s$, respectively, $s < k$. We may easily compute the coefficients of the two polynomials $\tilde{q}(x) = \prod_{i=1}^{k}(x - \tilde{y}_i)$, $\tilde{t}(x) = \prod_{j=1}^{s}(x - \tilde{z}_j)$ and then apply the algorithms of this section to compute the polynomials $\tilde{f}(x)$ and $\tilde{g}(x)$ satisfying the equation $\tilde{f}(x)\tilde{q}(x) + \tilde{g}(x)\tilde{z}(x) = 1$. This equation has the same form as (5.1). To see its relation to (1.3), multiply its both sides by $\tilde{d}(x)$ and obtain that

$$\tilde{f}(x)\tilde{u}(x) + \tilde{g}(x)\tilde{v}(x) = \tilde{d}(x) \ ,$$

where

$$\tilde{u}(x) = \tilde{q}(x)\tilde{d}(x) \ ,$$
$$\tilde{v}(x) = \tilde{t}(x)\tilde{d}(x) \ ,$$
$$\tilde{d}(x) = \gcd\ (\tilde{u}(x), \tilde{v}(x)) \ .$$

Due to Lemma 3.1, we have

$$\| \ \tilde{u}(x) - u(x) \ \| \leq \| \ u(x) \ \| \ ((1+\delta)^n - 1) \ ,$$

$$\| \tilde{v}(x) - v(x) \| \leq \| v(x) \| ((1-\delta)^n - 1)$$

provided that $\tilde{d}(x)$ is a δ-gcd of $u(x)$ and $v(x)$.

6. Padé Approximation and Structured Matrix Computations for Computing the GCDs.

In this section, we will recall a distinct approach to computing polynomial gcds, and in the next section, we will comment on its extension to approximating the gcds. We will start with the following basic algorithm.

Algorithm 6.1.

Input: coefficients of two nonconstant polynomials $u(x)$ and $v(x)$, of degrees m and n, respectively.

Output: $d(x) = \gcd(u(x), v(x))$.

Computation:

1. Compute two nonzero polynomials $w(x)$ and $z(x)$ of the minimum degrees satisfying the polynomial equation

$$w(x)v(x) = u(x)z(x) \ . \tag{6.1}$$

2. Compute and output $d(x) = u(x)/w(x)$.

Correctness of the algorithm is immediately verified.

Let us comment on the *computational cost* of its performance.

Stage 2 is the division of two polynomials with no remainder, which can be reduced to performing 2^ℓ concurrent divisions of pairs of scalars and $O((m+n)/d)$ FFTs at 2^ℓ-th roots of 1, for $\ell = \lceil \log_2(1+d) \rceil$, $d = \deg d(x)$. The overall cost of performing these operations is bounded by $O(\log(m+n), m+n)$ and is dominated by the cost of performing Stage 1, which, of course, may depend on the implementation of this stage (see Subalgorithm 6.2).

Hereafter, let us assume, with no loss of generality, that

$$\deg v(x) = n = \deg v(x) + 1 \tag{6.2}$$

and consider the two reverse polynomials $U(x) = x^{n-1}u(1/x)$, $V(x) = x^n v(1/x)$. Then $V(0) \neq 0$, and we may define the analytic function

$$h(y) = \frac{U(x)}{V(x)} = \sum_{i=0}^{\infty} h_i x^{-1-i} = \sum_{i=0}^{\infty} h_i y^{i+1} \ , \quad y = 1/x \ . \tag{6.3}$$

Remark 6.1 We could have alternatively associated the pair of $u(x)$ and $v(x)$ with the analytic function or formal power series $g(x) = u(x)/v(x) = \sum_{i=0}^{\infty} g_i x^i$ if $v(0) \neq 0$. If $u(x) = x^r \hat{u}(x)$, $v(x) = x^s \hat{v}(x)$, $rs > 0$, we could have written $g(x) = \hat{u}(x)/\hat{v}(x)$; this would not have preserved (6.2) unless $r = s$. Alternatively, we could have shifted the variable x so as to define an analytic function $f(x) = \frac{u(x-s)}{v(x-s)} = \sum_{i=0}^{\infty} f_i x^i$ for some scalar s, such that $v(s) \neq 0$. Then, in the subsequent presentation, we could have replaced $h(y)$ by $g(x)$ or by $f(x)$.

We will use the following definition [Gr72], [BGY80], [BP94].

Definition 6.1. For an analytic function (or even for any formal power series) $a(x) = \sum_{i=0}^{\infty} a_i x^i$ and for two nonnegative integers k and ℓ, a pair of polynomials $q(x)$ and $t(x)$ is a (k, ℓ) *Padé approximation* of $a(x)$ if $\deg q(x) \leq k$, $\deg t(x) \leq \ell$, and $q(x) - a(x)t(x) = 0 \bmod x^{N+1}$, $N = k + \ell$.

Proposition 6.1 [Gr72]. *The pair of polynomials $q(x)$ and $t(x)$ of Definition 6.1 is defined uniquely, up to its scaling by common factors or common divisors.*

Now, we may perform Stage 1 of Algorithm 6.1 by applying the following subalgorithm.

Subalgorithm 6.2.

a) Compute the first $N+1 = 2n$ coefficients $h_0, \ldots, h_N$ of the Taylor expansion (6.3) of the analytic function $h(y)$ [cf. (6.2)].

b) Compute the $(n-1-r, n-r)$ Padé approximation $W(y)$, $Z(y)$ of the analytic function $h(y)$ of (6.3), where r is the maximum integer for which (6.1) holds, provided that $w(x) = x^{n-1-r} W(1/x)$ and $z(x) = x^{n-r} Z(1/x)$.

c) Output the two polynomials $w(x)$ and $z(x)$.

Correctness of this subalgorithm immediately follows from Definition 6.1 and Proposition 6.1.

The *computational cost* of performing Stage b) dominates the cost of the entire computation.

Indeed, Stage a) is essentially reduced to computing the reciprocal of $V(x)$ modulo x^{N+1} and to multiplication modulo x^{N+1} of two polynomials and, therefore, can be performed at the cost $O((\log n) \log^* n, n/\log^* n)$, where

$\log^* n = \max\{i,\ \log^{(i)} n \geq 0\}$, $\log^{(i)} n = \log\log^{(i-1)} n$, $i = 1, 2, \ldots$; $\log^{(0)} n = n$ [BP93].

Stage c) is the cost-free reversion of the order of the coefficients of the two polynomials $W(y)$ and $Z(y)$.

Stage b) can be performed at the cost $O(n \log^2 n, 1)$ by means of applying a fast version of Euclidean algorithm [BGY88]. Alternatively, this stage can be reduced essentially to computation of the rank r of the $n \times n$ Hankel matrix $H = (h_{i,j}) = H(u, v)$, associated with the expansion (6.3) and such that $h_{i,j} = h_{i+j}$, $i, j = 0, 1, \ldots, n-1$, and to solving a Hankel linear system of r equations with the coefficients forming the $r \times r$ leading principal submatrix H_r of H (cf. Algorithm 2.5.1 of [BP94]). These two computations can be performed with $O(n \log \|H\|)$-bit precision at arithmetic cost $O((\log n) \log(n \log \|H\|), n \log n)$, provided that the matrix H is filled with integers and rounding-off a rational number to the closest integer is allowed as a unit cost operation ([BP94], section 9 of chapter 4, and [P,b]); the arithmetic cost increases to $O(\log^2 n, n^2/\log n)$ if H is filled with real or complex numbers [P92a]. Here and hereafter, we use the following definition.

Definition 6.2. $\|A\| = \|A\|_1 = \max_j \sum_i |a_{i,j}|$ denotes the *1-norm* of a matrix $A = (a_{i,j})$, and $\|A\|_2$ denotes the *2-norm* of A [and it is known that for an $n \times n$ matrix A, we have $\|A\|/\sqrt{n} \leq \|A\|_2 \leq \sqrt{n}\|A\|$] (cf. [GL89] or [BP94], pp. 90–91).

For practical numerical computation, one should take into account the requirement of numerical stability in these computations. For this reason, it is preferable to replace the solution of the Hankel linear system with the coefficient matrix H_r by the solution of the linear system whose coefficient matrix is the $r \times r$ leading principal submatrix $(JB(u, v)J)_r$ of the matrix $JB(u, v)J$, where $B(u, v) = B(u, 1)H(u, v)B(u, 1)$ is the Bezout matrix associated with the polynomials $u(x)$ and $v(x)$, where

$$B(u,1) = \begin{pmatrix} u_1 & . & . & . & u_n \\ . & & & . & \\ . & & . & & \\ . & . & & & \\ u_n & & & & O \end{pmatrix}$$

and where J is the reversion matrix, $J\vec{w} = (w_n, \dots, w_1)^T$ for any vector $(w_1, \dots, w_n)^T$,

$$J = \begin{pmatrix} O & & 1 \\ & \iddots & \\ 1 & & O \end{pmatrix}, \quad J^2 = I$$

(cf. [BP94], Algorithm 2.9.1).

The asymptotic (arithmetic) cost of the solution of this system is the same as for the original Hankel system [and, in fact, the inversion of the matrix $B(u, 1)$ is a simple operation, essentially equivalent to computing the reciprocal polynomial $(1/u(x)) \bmod x^{n+1}$], but the numerical stability of the solution algorithm is improved versus the Hankel case (see Remark 2.9.4 in [BP94]).

Remark 6.2. Computation of any fixed intermediate entry of the extended Euclidean algorithm for two polynomials $u(x)$ and $v(x)$ is equivalent to the computation of a (k, ℓ) Padé approximation of the associated analytic function (6.3) [Gr72], [BGY80], [BP94]; the algorithms of this section and their extension in the next section apply.

7. Approximating the GCDs via the Padé Approximation Approach.

Suppose that we have fixed a class C of polynomial pairs $u^*(x)$ and $v^*(x)$ approximating a given pair $u(x)$ and $v(x)$ of polynomials and that in this class we seek a pair maximizing the degree of $\gcd(u^*(x)v^*(x))$. Let $u^*_+(x)$ and $v^*_+(x)$ denote such a pair, let $d^*(x)$ denote such a gcd, and let d^* denote its degree. (We may define the class C by (1.1) and (1.2) for a fixed b or by the assumptions of Lemma 3.1 for a fixed δ.) Let us assume, with no loss of generality, that $\deg u^*(x) = n - 1$, $\deg v^*(x) = n$ [cf. (6.2)] and associate the analytic function $h^*(y) = \frac{u^*(1/x)}{xv^*(1/x)} = \sum_{i=0}^{\infty} h^*_i y^{i+1}$, $y = 1/x$, to every pair $u^*(x)$ and $v^*(x)$ [cf. (6.3)]. Let $H^* = H(u^*, v^*)$ denote the $n \times n$ Hankel matrix $(h^*_{i,j}, \ i, j = 0, 1, \dots, n-1)$, where $h^*_{i,j} = h^*_{i+j}$ for all i and j. Then

$$\deg \ \gcd\ (u^*(x), v^*(x)) = n - \operatorname{rank} H^* , \tag{7.1}$$

and our problem is reduced to the following one:

Problem 7.1. For a class C of polynomial pairs defined above, compute a pair of polynomials $(u_+^*(x), v_+^*(x)) \in C$ and a nonnegative integer r^* such that $r^* = \text{rank } H(u_+^*, v_+^*)$ is the minimum value of rank H^* over all Hankel matrices $H^* = H(u^*, v^*)$ associated with the pairs $(u^*(x), v^*(x)) \in C$.

The only effective solution algorithm that we know reduces Problem 7.1 to computing an approximate gcd of $u(x)$ and $v(x)$ (see Algorithm 8.1 in the next section), but here is an algorithm that computes a lower bound r_- on r^* and, consequently, an upper bound $d_+ = n - r_-$ on $d^* = n - r^*$.

Algorithm 7.1.

Input: a pair of polynomials $u(x)$ and $v(x)$ of degrees $n-1$ and n, respectively, and a class C of polynomial pairs $u^*(x)$, $v^*(x)$ of degrees $n-1$ and n, respectively.

Output: a lower bound r_- on the minimum rank r^* of the Hankel matrices $H^* = H(u^*, v^*)$, for $(u^*(x), v^*(x)) \in C$.

Computation:

1. Estimate an upper bound ϵ on the pertubation norm $\|H^* - H\|$ over all Hankel matrices $H^* = H(u^*, v^*)$ associated with pairs of polynomials $(u^*(x), v^*(x)) \in C$ that approximate the polynomials $u(x)$ and $v(x)$.

2. Compute the number s_ϵ of the singular values of the matrix H that exceed ϵ. Output $r_- = s_\epsilon$.

Correctness of this algorithm follows from Corollary 8.3.2 of [GL89], according to which $r_- = s_\epsilon$ is exactly the minimum rank of all the matrices A lying in the ϵ-neighborhood of H. (This minimum, however, does not have to be attained at the Hankel matrices approximating H, so that r_- does not generally give us the solution value r^* for Problem 7.1.)

Computational cost of Stage 1 may grow as we require tighter upper bounds ϵ. A reduction of Stage 1 to polynomial division leads to computing a reasonably good bound ϵ at the cost $O((\log n) \log^* n, n/\log^* n)$ [BP93].

To perform Stage 2, we may apply the following subalgorithm, where we write H^H to denote the Hermitian transpose of a a Hankel matrix H, so that each entry (i, j) of H^H is the complex conjugate of the entry (j, i) of H, and $H^H = H$ if H is a real Hankel matrix.

Subalgorithm 7.2 (cf. Remark 7.2 at the end of this section).

a) Reduce the matrix $H^H H$ to the real symmetric tridiagonal form T by means of a similarity transformation (cf. [BP94], Algorithm 2.3.1).

b) Compute the values $c_k(\epsilon)$ of the characteristic polynomials $c(\lambda) = \det(\lambda I_k - (H^H H)_k) = \det(\lambda I_k - T_k)$ for $\lambda = \epsilon$ and for $k = 0, 1, \ldots, n$, where I_k is the $k \times k$ identity matrix and W_k denotes the $k \times k$ leading principal submatrix of a matrix W for $W = H^H H$ and $W = T$.

c) Compute and output $s_\epsilon = n - SC_\epsilon$, SC_ϵ denoting the number of the sign changes in the sequence $\{c_k(\epsilon), \quad k = 0, 1, \ldots, n\}$.

To show *correctness* of this subalgorithm, recall that its Stage c) outputs the number of the eigenvalues of the matrix $H^H H$ that exceed ϵ (see Theorem 8.4.1 of [GL89]). On the other hand, the nonzero eigenvalues of $H^H H$ are the singular values of H, and therefore, correctness of Subalgorithm 7.2 follows from Corollary 8.3.2 of [GL89].

Let us comment on the *computational cost* of performing Subalgorithm 7.2.

Stage a) can be performed at the cost $O(\log^2 n, n^2/\log n)$ since the most costly stage of Algorithm 2.3.1 of [BP94] has this cost bound for a Hankel input matrix H, because the matrix $H^H H$ is a Toeplitz-like matrix whose displacement rank is at most 4 (cf. the definitions in section 11 of chapter 2 of [BP94]). To compute the sequence of the values $c_k(\epsilon)$, $k = 0, 1, \ldots, n$, at Stage b), we apply three-term recurrence relations for these values, in the matrix form (cf. [BP91], Appendix C), and this enables us to reduce the computation to the prefix product computation. By using the prefix sum/prefix product algorithm ([KR90] or [BP94], Algorithm 4.2.1), we reach the cost bounds $O(\log n, n/\log n)$, which also dominate the cost of performing Stage c). The overall cost of performing Subalgorithm 7.2 is, therefore, dominated by the cost bound $O(\log^2 n, n^2/\log n)$ of performing its Stage a).

Now, suppose that Algorithm 7.1 (with its Stage 2 performed by means of Subalgorithm 7.2) has output a value r_-. In this case we may compute an upper bound $d_+ = n - r_-$ on $d^* = n - r^*$. Suppose that we also know some lower bound d_- on d^*. (Clearly, $d^* \geq 0$, but we may obtain a better lower bound by applying, say, a numerical version of the Euclidean algorithm, cf. [EGL96].) Then, we may apply our algorithms of Sections 5 or 6 (or any black box algorithm) in order to approximate (for every d in the range from d_- to

d_+) a common divisor of a degree at most d for the polynomials $u(x)$ and $v(x)$. In the case where $r_- = r^*$ and $d = d_+ = d^*$, our algorithms of Sections 5 and 6 output a pair of polynomials $(u^*(x), v^*(x)) \in C$ and their gcd, $d^*(x)$, of degree d^*. The latter computation is reduced to solving a resultant, Hankel, or Bezout linear system, which is nonsingular if $r_- = r^*$ and whose solution can be obtained at the cost $O(n \log^2 n, 1)$ or, alternatively, $O(\log^2 n, n^2/\log n)$ (cf. [BP94]). If $d < d^*$, then our algorithms output a common divisor of $u(x)$ and $v(x)$ having a degree at most d. If $d > d^*$, we may allow our algorithms to fail or to output a wrong answer, since this will be detected at the subsequent certification stage. Due to such a certification, we may apply binary search so as to compute and to test only $\lceil \log_2(d_+ - d_-) \rceil + 2$ common divisors of $u(x)$ and $v(x)$, rather than $d_+ - d_- + 1$ divisors. An effective certification algorithm can be based on the approximation of the zeros of the available approximate common divisor and on the techniques proposed in Remark 3.3. This computation is much simpler than the approximation of all the zeros of $u(x)$ and $v(x)$ if d_+ is much less than m, which is most frequently the case in the computer algebra applications.

Remark 7.1. Instead of Hankel matrices H and H^*, one may similarly associate with $u(x)$, $v(x)$ and with $u^*(x)$, $v^*(x)$ the Sylvester matrices $S = S(u, v)$ and $S^* = S(u^*, v^*)$ and compute a lower bound on rank S^* and an upper bound on $d^* = n - \text{rank } S^*$ by counting the number of singular values of S that exceed a fixed value ϵ. The paper [CGTW95] has proposed a more complicated and more costly approach, based on computation of the entire Singular Value Decomposition of S. Furthermore, unlike the computation of the SVD, which involves irrational values even where the input is rational or integer, Algorithm 7.1 with Subalgoritm 7.2 only involve rational arithmetic computations (which can be performed with no roundoff errors) and a few comparisons.

Remark 7.2. In the most interesting case where H is a real Hankel matrix, we may obtain a substantial further simplification. Namely, we may replace the matrix $H^H H$ by H throughout Subalgorithm 7.2, compute the values $c_k(-\epsilon^*\sqrt{2})$ and $c_k(-\epsilon\sqrt{2})$, $k = 0, 1, \ldots, n$, at Stage b), for any ϵ^* slightly exceeding ϵ, and then compute $s_\epsilon = n - C_{\epsilon\sqrt{2}} + C_{-\epsilon^*\sqrt{2}}$ at Stage c), where C_α (for $\alpha = \epsilon\sqrt{2}$ and for $\alpha = -\epsilon^*\sqrt{2}$) denotes the number of sign changes in the sequence $\{c_k(\alpha)\}$. Due to Theorem 8.4.1 of [GL89], $C_{\epsilon\sqrt{2}} - C_{-\epsilon^*\sqrt{2}}$ is the

number of the eigenvalues of H in the real line interval from $-\epsilon\sqrt{2}$ to $\epsilon^*\sqrt{2}$, and this number is not less than $n-$ rank H, due to Theorem 8.1.8 of [GL89].

8. Computing the Hankel Numerical Rank of a Hankel Matrix

Let us next revisit Problem 7.1 and solve it by applying the following algorithm, which reverses our approach of section 7.

Algorithm 8.1.

Input and output as for Problem 7.1.

Computations.

1. Compute an approximate gcd, $d^*(x)$, of $u(x)$ and $v(x)$ having the maximum degree d^* over all polynomial pairs $(u^*(x), v^*(x)) \in C$. Output the associated pair $(u^*_+(x), v^*_+(x)) \in C$ such that $d^*(x) = \gcd(u^*_+(x), v^*_+(x))$.

2. Output $r^* = n - d^*$.

Correctness of this algorithm immediately follows from the definition of class C and from (7.1).

For the class C defined according to the assumptions of Lemma 3.1, the *computational cost* of performing Algorithm 8.1 has been estimated in Sections 3 and 4.

Let us now state a related problem, having independent interest too.

Problem 8.1. Given a positive ϵ and an $n \times n$ Hankel matrix H, compute an $n \times n$ Hankel matrix H^* satisfying

$$\|H^* - H\| \leq \epsilon \tag{8.1}$$

and having the minimum rank $r_\epsilon(H)$, which we will call *Hankel ϵ-rank* of H or *Hankel numerical rank* of H.

By applying Subalgorithm 7.2 (or the algorithm of Remark 7.2), we obtain a lower bound r^-_ϵ on $r_\epsilon(H)$. To compute an upper bound, r^+_ϵ, we first compute a pair of polynomials $u(x)$, $v(x)$ associated with the matrix H and then apply Algorithm 8.1 to the class C of sufficiently tight approximations $(u^*(x), v^*(x))$ to the pair $(u(x), v(x))$. Suppose that the class C is defined according to (1.1) and (1.2) or according to the assumptions of Lemma 3.1. Then, clearly, for sufficiently large b or sufficiently small positive δ, we shall arrive at some upper bounds r^+_b or r^+_δ on $r^+_\epsilon(H)$. By repeating the computation for dynamically

decreasing b or increasing δ, we may obtain tighter upper bounds on $r_\epsilon^+(H)$. For a large class of Hankel matrices H, such a process gives us sharp bounds $r_b^+ = r_\epsilon^- = r_\epsilon^+(H)$ or $r_\delta^+ = r_\epsilon^- = r_\epsilon^+(H)$. Generally, however, we may end up with a gap between r_b^+ or r_δ^+ and r_ϵ^-, so that the proposed approach generally remains heuristic, although we may also try our luck for various ϵ, thus increasing the chances for obtaining a solution to Problem 8.1.

Finally, let us examine the computation of the associated polynomial pair $u(x)$, $v(x)$. If the matrix H is nonsingular, then the associated pair $u(x)$, $v(x)$ is unique and can be computed at the cost $O(n \log^2 n, 1)$ or, alternatively, at the cost $O(\log^2 n, n^2/\log n)$ (cf. [BP94], Proposition 2.9.1, Algorithm 2.11.2, and Theorem 2.13.1).

Formally, we may ensure nonsingularity with a high probability if we shift from the input matrix H to a nearby matrix $H^{(\sigma)} = H + \sigma H_0$, where H_0 is a fixed Hankel matrix, $\|H_0\| = 1$, say, $H_0 = J$, and σ is a random value chosen from a sufficiently large set S. (Note that $\det H^{(\sigma)}$ is a polynomial in σ of a degree at most n and recall Corollary 1.5.1 of [BP94].) We may require that S consist of values σ having small magnitudes $|\sigma|$ relative to ϵ; then, the transition from H to $H^{(\sigma)}$ would little affect the solution of Problem 8.1.

Let us conclude this section by showing that the pairs of polynomials $u_\sigma(x)$, $v_\sigma(x)$ associated with $H^{(\sigma)}$ converge to the pair $u(x)$, $v(x)$ associated with H as $\sigma \longrightarrow 0$, even if H is a singular matrix.

Indeed, let us normalize the Padé approximation associated with $H^{(\sigma)}$, so as to have $\| u_\sigma(x) \| = 1$, instead of having the polynomial $u_\sigma(x)$ monic. Furthermore, the set of the coefficient vectors of all polynomials $u_{\sigma_k}(x)$ for any sequence $\{\sigma_0, \sigma_1, \ldots\}$ is compact, and the sequence of the associated coefficient vectors $\vec{u}^{\,(\sigma_k)}$ has a subsequence converging to some vector $\vec{u}$. Furthermore, the associated subsequence of the coefficient vectors $\vec{v}^{\,(\sigma_k)}$ of the polynomials $v_{\sigma_k}(x)$ also converges to some vector $\vec{v}$. [Alternatively, we could have required that $\|u_\sigma(x)\| + \|v_\sigma(x)\| = 1$, to ensure compactness of the set of the pairs of the coefficient vectors $(\vec{u}^{\,(\sigma)}, \vec{v}^{\,(\sigma)})$.]

The ratios of the associated polynomials satisfy the equations $u_{\sigma_k}(1/x)/(x v_{\sigma_k}(1/x)) = y u_{\sigma_k}(y)/v_{\sigma_k}(y) = \sum_{j=0}^{2n-1} h_j^{(\sigma)} y^{j+1} \bmod y^{2n+1}$ for $y = 1/x$ and for all k [compare (6.3)], and therefore, the polynomials $u(x)$ and $v(x)$ associated with

$\vec{u}$ and $\vec{v}$ satisfy the equation:

$$yu(y)/v(y) = u(1/x)/(xv(1/x)) = \sum_{j=0}^{2n-1} h_j y^{j+1} \bmod x^{2n+1} ,$$

that is, the pair of the polynomials $u(y)$, $v(y)$ is an $(n-1,n)$ Padé approximation of the latter power series associated with the matrix H. Such an approximation is unique [Gr72] (up to dividing $u(y)$ and $v(y)$ by their gcd) assuming that $u(x)$ and $v(x)$ are relatively prime. Due to this uniqueness, the sequence $\vec{u}^{\,(\sigma_k)}$ itself (rather than its subsequence) converges to $\vec{u}$, and consequently, $\vec{v}^{\,(\sigma_k)}$ converges to $\vec{v}$.

Remark 8.1. The algorithms of this section can be immediately extended to computation of the *Toeplitz ϵ-rank* or *numerical Toeplitz rank* of a square Toeplitz matrix T since JT is a Hankel matrix and since rank T = rank (JT). They can be also extended to the computation of the *Sylvester ϵ-rank* or *numerical Sylvester rank* of a square Sylvester matrix $S = S(u,v)$. In the case of a Sylvester input matrix, the coefficients of the associated pair of polynomials $u(x)$ and $v(x)$ are explicitly given by two columns of the matrix S, which a little simplifies the computation of the numerical Sylvester rank and the analysis of the complexity of this computation, versus the Hankel and Toeplitz cases. One may easily obtain a similar extension to computation of the *Bezout ϵ-rank* or *Bezout numerical rank* of a square Bezout matrix.

9. Summary and Concluding Remarks

We have started with recalling the known definitions and algorithms for approximate polynomial gcds from [Sc85], [KM94], [CGTW95], and [HS,a], with demonstrating a certain major deficiency of these definitions (see our Example 1.1), and with discussing some problems with these algorithms. Then we gave a new, more restrictive definition, which avoided such a deficiency, and we presented two algorithms for the computation of approximate gcds under such a more restrictive definition. In spite of an additional restriction imposed on the approximate gcds, our algorithms do not require to increase (versus the previous record bound) the overall bit-complexity of the computations (even if we include the computational cost of performing the auxiliary stage of approximating the zeros of the input polynomials); moreover, we achieve some

substantial advantages (versus the known algorithms) in overcoming the heuristic character of most of the proposed approaches and substantially improving their numerical stability and computational cost estimates; in particular, unlike all the previously known algorithms, the new algorithms can be performed in RNC or NC, that is, they can be accelerated so as to run in polylogarithmic parallel time by using $n^{O(1)}$ processors.

In Section 5, we have shown how one can compute two approximate cofactors of the gcd. The computation involves operations with Sylvester matrices. The major practical advantage, versus the previous approach, is in avoiding the computation of the ranks of such matrices. Theoretically, we have also achieved parallel acceleration, so as to perform this computation in NC.

In Section 6, we have showed how to compute polynomial gcds based on computing Padé approximation, and in Section 7, we have extended this study to approximate gcds. We have reduced the solution to computing the number of those singular values or eigenvalues of a Hankel (or Sylvester) matrix that lie in a fixed small interval about 0 and to the solution of a Bezout nonsingular linear system, and this has substantially improved the SVD approach of [CGTW95], in terms of both its computational cost (since, unlike [CGTW95], we avoid computing the SVD and also exploit the symmetry of the Hankel and Bezout matrices, involved in our computations) and numerical stability (due to using Bezout matrices instead of Sylvester matrices). Furthermore, unlike the SVD computation, involving irrational values even for a rational or integer input, our modification enables us to perform the computations by using rational arithmetic (with no roundoff errors) and a few comparisons. The latter improvements of the approach of [CGTW95] have been obtained, in our Section 7, assuming the same customary definition of approximate gcd that was used in [CGTW95]. This has facilitated the comparison with [CGTW95] but has also implied that, unlike the approach of our Sections 3 and 4 and like one of [CGTW95], the solutions proposed in Section 7 are generally heuristic because they give us an upper bound d_+ on the maximum degree d^* of an approximate gcd, but do not necessarily give us the degree itself. Unlike [CGTW95], however, we extend our computations to obtaining an approximate gcd of degree d^*, by applying our techniques of Sections 3 and 4. In this case, the computations are simplified, versus ones of Sections 3 and 4, since we

only need to approximate the zeros of the candidate approximate gcd, whose degree can be much smaller than m.

Finally, in Section 8, we have shown a partly heuristic extension of the presented approach, where approximating the gcd is a basic step of computing the Hankel numerical rank of a square Hankel matrix, which in turn is the crucial step of computing Padé approximations and of Berlekamp-Massey computations. These or similar algorithms apply to computing Bezout, Toeplitz, and Sylvester numerical ranks.

There are various natural directions for further study.

For example, one may relate approximating polynomial zeros to their magnitudes, so as to approximate (within a fixed error bound δ) the zeros z_k for which $|z_k| \leq 1$ and to approximate within δ the reciprocals $1/z_k$ of all other zeros. As another example, one may revisit the algorithms of [P95], [P96], used at a preconditioning stage of the algorithms of sections 3-5 and 8 and try to simplify them in the case where the objective is the application to approximating gcds, rather than to approximating polynomial zeros. For instance, if an algorithm of [P95], [P96] splits a polynomial $u(x)$ into two factors one of which is relatively prime with $v(x)$, then we may immediately discard this factor, instead of proceeding with approximating its zeros, as the algorithms of [P95], [P96] would normally do. Furthermore, if, say, $u(x)$ is the worst case polynomial for algorithms of [P95], [P96] (say, if the zeros of $u(x)$ form various clusters), whereas $v(x)$ is an easy case polynomial, then one may replace $u(x)$ by $u(x) + av(x)$ for some scalar a and apply the algorithms of [P95], [P96] to the polynomials $u(x) + av(x)$ and $v(x)$, rather than to $u(x)$ and $v(x)$.

Another major subject of further study could be the determination of conditions on ϵ and on the matrices H under which the lower and upper bounds r_ϵ^- and r_ϵ^+ on $r_\epsilon(H)$ of Section 8 meet each other. And, of course, a major further step should be numerical tests, in particular, for our algorithms of Section 8 and for comparision of our approach of Sections 2–5 with ones of Section 7 and with the cited alternative ways to the solution, from [Sc85], [NS91], [KM94], [HS,a], and [CGTW95].

REFERENCES

AS88 W. Auzinger, H.J. Stetter, An Elimination Algorithm for the Computation of All Zeros of a System of Multivariate Polynomial Equations, In *Proc. Conf. in Numerical Analysis*, *ISNM*, vol. 86, pp. 11-30, Birkhaeuser, 1988.

Be68 E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

BGY80 R.B. Brent, F.G. Gustavson, D.Y.Y. Yun, Fast Solution of Toeplitz Systems of Equations and Computation of Padé Approximations, *J. of Algorithms*, 1, 259–295, 1980.

BP86 D. Bini, V.Y. Pan, Polynomial Division and Its Computational Complexity, *J. of Complexity,* 2, 179–203, 1986.

BP91 D. Bini, V. Y. Pan, Parallel Complexity of Tridiagonal Symmetric Eigenvalue Problem, *Proc. 2nd Ann. ACM-SIAM Symposium on Discrete Algorithms,* 384–393, ACM Press, New York, and SIAM Publications, Philadelphia, Pennsylvania, 1991.

BP93 D. Bini, V. Y. Pan, Improved Parallel Polynomial Division, *SIAM J. of Computing,* 22, 3, 617–627, 1993.

BP94 D. Bini, V.Y. Pan, *Polynomial and Matrix Computations, vol. 1: Fundamental Algorithms*, Birkhaeuser, Boston, Massachusetts, 1994.

C88 J.F. Canny, *The Complexity of Robot Motion Planning*, ACM Doctoral Dissertation Series, MIT Press, Cambridge, Massachusetts, 1988.

C90 J.F. Canny, Generalized Characteristic Polynomials, *J. of Symbolic Computation*, pp. 241-250, 1990.

CW90 D. Coppersmith, S. Winograd, Matrix Multiplication via Arithmetic Progressions, *J. of Symbolic Computation*, 9, 3, pp. 251-280, 1990.

CGTW95 R.M. Corless, P.M. Gianni, B.M. Trager, S.M. Watt, The Singular Value Decomposition for Polynomial Systems, *Proc. Intern. Symp. on Symbolic and Algebraic Comp. (ISSAC '95)*, pp. 195-207, ACM Press, New York, 1995.

EGL96 I. Z. Emiris, A. Galligo, H. Lombardi, Certified Approximate Polynomial GCDs, to appear in Proc. of MEGA'96.

G84 J. von zur Gathen, Parallel Algorithms for Algebraic Problems, *SIAM J. on Computing*, 13, 4, pp. 802-824, 1984.

G95 A. Galligo, On Some Difficulties of Approximating Polynomial GCD, *Lecture at AMS-SIAM Conference: Mathematics of Numerical Analysis: Real Number Algorithms,* Park City, Utah, July-August 1995.

GL89 G.H. Golub, C.F. Van Loan, *Matrix Computations,* Johns Hopkins Univ. Press, Baltimore, Maryland, 1989.

GP88 Z. Galil, V. Pan, Improved Processor Bounds for Combinatorial Problems in RNC, *Combinatorica*, 8, 2, pp. 189-200, 1988.

Gr72 W.B. Gragg, The Padé Table and Its Relation to Certain Algorithms of Numerical Analysis, *SIAM Review*, 14, 1, 1-62, 1972.

H90 C.M. Hoffman, Algebraic and Numeric Techniques for Offsets and Blends, In *Computations of Curves and Surfaces* (W. Dahmen, M. Gasca, and C.M. Micchelli editors), pp. 499-528, Kluwer Academic Publishers, 1990.

HK73 J.E. Hopcroft, R.M. Karp, An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs, *SIAM J. on Computing*, 2, pp. 225-231, 1973.

HS,a V. Hribernig, H.J. Stetter, Detection and Validation of Clusters of Polynomial Zeros, to appear in *J. Symb. Comp.*, 1995.

II86 A. Israeli, A. Itai, A Fast and Simple Randomized Parallel Algorithm for Maximal Matching, *Information Proc. Letters*, 22, 77-80, 1986.

IS86 A. Israeli, Y. Shiloach, An Improved Parallel Algorithm for Maximal Matching, *Information Proc. Letters*, 22, 57-60, 1986.

J92 J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, Massachusetts, 1992.

KL96 N. Karmarkar, Y. N. Lakshman, Approximate Polynomial Greatest Common Divisor and Nearest Singular Polynomials, *Proc. Ann. ACM-SIGZAM Intern.Symp. on Symb. and Algebraic Comp.,* 1996, to appear.

KM94 N. Karcanias, M. Mitrouli, A Matrix Pencil Based Numerical Method for the Computation of GCD of Polynomials, *IEEE Trans. on Automatic Control*, 39, 5, 977-981, 1994.

KM95 S. Krishnan, D. Manocha, Numeric-Symbolic Algorithms for Evaluating One-Dimensional Algebraic Sets, *Proc. Intern. Symp. on Symbolic and Algebraic Computing (ISSAC '95)*, ACM Press, New York, 1995.

KR90 R.M. Karp, V. Ramachandran, A Survey of Parallel Algorithms for Shared Memory Machines, *Handbook for Theoretical Computer Science* (J. van Leeuwen editor), pp. 869-941, North-Holland, Amsterdam, 1990.

L81 D. Lazard, Resolution des Systems d'Equations Algebrique, *Theoretical Computer Science,* 15, 77–110, 1981.

Le92 F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes,* Morgan Kaufmann, San Mateo, California, 1992.

M94 D. Manocha, Computing Selected Solutions of Polynomial Equations, *Proc. Intern. Symp. on Symbolic and Algebraic Computing (ISSAC '94)*, pp. 1-8, ACM Press, New York, 1994.

M94a D. Manocha, Solving Systems of Polynomial Equations, *IEEE Trans. on Computer Graphics and Applications*, pp. 46-55, March 1994.

MC93 D. Manocha, J.F. Canny, Multipolynomial Resultant Algorithms, *J. of Symbolic Computation*, 15, pp. 99-122, 1993.

MD92 D. Manocha, J. Demmel, Algorithms for Intersecting Parametric and Algebraic Curves, *Graphic Interface '92*, pp. 232-241, 1992.

MD94 D. Manocha, J. Demmel, Algorithms for Intersection Parametric and Algebraic Curves I: Simple Intersections, *ACM Trans. Graphics*, 13, 1, pp. 73-10, 1994.

MD95 D. Manocha, J. Demmel, Algorithms for Intersecting Parametric and Algebraic Curves II: Multiple Intersections, *Graphical Models and Image Processing*, 57, 2, pp. 81-100, 1995.

MS95 H. Möller, H.J. Stetter, Multivariate Polynomial Equations with Multiple Zeros Solved by Matrix Eigenproblems, to appear in *Numerische Mathematik, 1995.*

NS91 M.-T. Noda, T. Sasaki, Approximate GCD and Its Application to Ill-Conditioned Algebraic Equations, *J. Computational and Applied Mathematics*, 38, pp. 335-351, 1991.

P87 V.Y. Pan, Complexity of Parallel Matrix Computations, *Theoretical Computer Science*, 54, pp. 65-85, 1987.

P92 V.Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, 34, 2, pp. 225-262, 1992.

P92a V. Y. Pan, Parametrization of Newton's Iteration for Computations with Structured Matrices and Applications, *Computers & Mathematics (with Applications),* 24, 3, 61–75, 1992.

P94 V.Y. Pan, Algebraic Improvement of Numerical Algorithms: Interpolation and Economization of Taylor Series, *Mathematical and Computer Modelling*, 20, 1, pp. 23-26, 1994.

P95 V.Y. Pan, An Algebraic Approach to Approximate Evaluation of a Polynomial on a Set of Real Points, *Advances in Computational Mathematics*, 3, pp. 41-58, 1995.

P95a V.Y. Pan, Optimal (up to Polylog Factors) Sequential and Parallel Algorithms for Approximating Complex Polynomial Zeros, *Proc. 27th Ann. ACM Symposium on Theory of Computing*, pp. 741-750, ACM Press, New York, 1995.

P96 V.Y. Pan, Optimal and Nearly Optimal Algorithms for Approximating Polynomial Zeros, *Computers and Mathematics (with Applications)*, 31, 12, pp. 97-138, 1996.

P96a V.Y. Pan, Effective Parallel Computations with Toeplitz and Toeplitz-like Matrices, to appear in *Proc. of the* AMS-SIAM *Workshop on Mathematics of Numerical Analysis: Real Number Algorithms*, Park City, Utah, July–August 1995 (M. Shub, S. Smale, J. Renegar editors), *Lectures in Applied Math.,* 32, pp.593-643, Amer. Math. Soc. Press, Providence, R.I., 1996.

P96b V. Y. Pan, On Approximating Complex Polynomial Zeros: Modified Quadtree (Weyl's) Construction and Improved Newton's Iteration, Techn. Report 2894, *INRIA, Sophia-Antipolis, France*, May 1996.

Par80 B. N. Parlett, *The Symmetric Eigenvalue Problem,* Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

PRT92 V.Y. Pan, J.H. Reif, S.T. Tate, The Power of Combining the Techniques of Algebraic and Numerical Computing: Improved Approximate Multipoint Polynomial Evaluation and Improved Multipole Algorithms, *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pp. 703-713, IEEE Computer Society Press, 1992.

PSLT93 V.Y. Pan, A. Sadikou, E. Landowne, O. Tiga, A New Approach to Fast Polynomial Interpolation and Multipoint Evaluation, *Computers and Mathematics (with Applications)*, 25, 9, pp. 25-30, 1993.

PZDH,a V.Y. Pan, A. Zheng, X. Huang, Y. Yu., Fast Multipoint Polynomial Evaluation and Interpolation via Computation with Structured Matrices, *Annals of Numerical Math.*, to appear in Sept. 1996.

Q94 M.J. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, New York, 1994.

R94 K. Roach, Symbolic-Numeric Nonlinear Equation Solving, *Intern. Symp. on Symbolic and Algebraic Computation (ISSAC' 94),* pp. 278–282, ACM Press, New York, 1995.

Sc85 A. Schönhage, Quasi-GCD Computations, *J. of Complexity*, 1, pp. 118-137, 1985.

S93 H.J. Stetter, Multivariate Polynomial Equations as Matrix Eigenproblems, *WSSIA* 2, *World Scientific*, pp. 355-371, 1993.

S93a H.J. Stetter, Verification in Computer Algebra Systems, in *Validation Numerica* (R. Albrecht, G. Alefeld, H.J. Stetter editors), *Computing Suppl.*, 9, pp. 247-263, 1993.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399