MM Research Preprints, 23–38
MMRC, AMSS, Academia, Sinica, Beijing
No. 21, December 2002

QR Factoring to compute the GCD of Univariate Approximate Polynomials [1)]

Robert M. Corless[2)] Stephen M. Watt [3)] and Lihong Zhi [4)]

Abstract. An algorithm based on QR factoring of the Sylvester matrix, using its structure, is presented for computing the GCD of univariate approximate polynomials over $\mathbb{R}[x]$ or $\mathbb{C}[x]$. An approximate polynomial is a polynomial with coefficients that are not known with certainty.

Keywords 1 *Greatest common divisor, QR-factoring, Sylvester matrix.*

1. Introduction

For an introduction to and motivation for the problem studied in this paper, and a brief survey of recent results and methods, see [?]. In short, the problem of finding a sensible greatest common divisor (GCD), when the coefficients of the polynomials are not known exactly, is of great practical importance (for example in avoiding spurious near pole-zero combinations in certain adaptive control applications) and some mathematical difficulty, owing to the potential discontinuity (of the degree of the GCD) as the coefficients are varied. Discontinuity is difficult to deal with, both symbolically (with parameters) and numerically, where problems that are near to points of discontinuity are ill-conditioned [?]. Some form of regularization must therefore be used, and most of the work in this area can be considered to be examining the effects of different regularizations. The most successful regularization seems to be to phrase the problem as an optimization problem, as in [?] and [?, ?].

1.1. Notation: Approximate Polynomials and Approximate GCD

Several papers use distinct wording and notation to denote the objects under study in this paper. We follow [?] and say that an *approximate polynomial* is a polynomial with coefficients that are not known exactly. We say that $d(x)$ is an *approximate GCD* of approximate polynomials $f(x)$ and $g(x)$ if there exist perturbations Δf and Δg, which are small in a sense to be specified later, such that $d(x)$ is a (true) GCD of $f + \Delta f$ and $g + \Delta g$. This can be contrasted with the notion of "quasi-GCD" of [?], in which the input polynomials f and g are known at any time only to a finite accuracy, but by some 'oracle' more digits of accuracy for any coefficient can be obtained on demand. The notion of "quasi-GCD" thus fits in with mathematical and computational studies of computable real numbers, but does not fit in with engineering or empirical models where the input polynomials are known only

[1)]The Ontario Research Centre for Computer Algebra University of Western Ontario London ON, N6A 5B7 Canada.

[2)]ORCCA,The University of Western Ontario, Canada

[3)]ORCCA,The University of Western Ontario, Canada

[4)]MMRC, Institute of Systems Science, AMS, Beijing, China to a limited accuracy once and for all. The paper [?] uses "quasi-GCD" and "ϵ-GCD" to distinguish two technical notions of approximate GCD.

1.2. QR factoring of the Sylvester matrix to find an approximate GCD

The paper [?] describes an efficient method to use the fast QR factoring to compute an approximate GCD, and their paper contains several important ideas and advances. Like the contemporary paper [?], it uses the structure of the Sylvester matrix to speed up computation, lowering the cost from $O((n+m)^3)$ to $O(n^2)$, but the stability is not guaranteed.

The paper [?] is perhaps not as easily available to the audience of this paper as the paper [?], and so we summarize it briefly here.

It is well known that Householder transformations and Givens rotations give stable methods to compute the QR factoring of a matrix. Householder transformations are powerful tools for introducing zeros into vectors, whereas Givens rotations introduce zeros into a vector one at a time. Therefore, Givens rotations are useful for operating on structured matrices. The Sylvester matrix (equation 2 below) consists of two Toeplitz matrices ($\mathbb{R}^{m\times(n+m)}, \mathbb{R}^{n\times(n+m)}$), formed by the coefficient vectors from f and g in equation (1) below[5)]. If we apply Givens rotations to the 1st row and $m+1$st row to eliminate g_m, then the rows i and $m+i$ for i from 2 to m can be changed in the same way. The near Toeplitz structure will not be changed until we obtain the following block matrix:

$$\begin{bmatrix} U & V \\ 0 & W \end{bmatrix}$$

where U is an $m\times m$ upper triangular matrix, $\mathsf{0}$ is a matrix with all elements zero and W is an $n\times n$ matrix. Since U is now an upper triangular matrix, Householder transformations can then be applied to the submatrix W. The complexity advantage of combining Given rotations with Householder transformations can be seen clearly. The cost of a general QR decomposition is $\frac{4}{3}(m+n)^3$ flops. Using the above special strategy, taking advantage of the structure, the flop count drops to $6nm+\frac{4}{3}n^3$.

This approach is similar in complexity to the non-orthogonal methods used in [?] to condense the Sylvester matrix into a smaller matrix, but [?] uses (as this present paper does) orthogonal reductions at all stages for stability reasons.

The paper [?] then goes on to use this method as a base method for multivariate GCD computations by Hensel lifting.

2. QR Factoring for a Sylvester Matrix

Let given polynomials f, g have degree $n \geq m$ respectively, where

$$\begin{aligned} f &= f_n x^n + f_{n-1}x^{n-1} + \cdots + f_1 x + f_0, \\ g &= g_m x^m + g_{m-1}x^{m-1} + \cdots + g_1 x + g_0. \end{aligned} \tag{1}$$

5)All the results of this paper go through immediately in the case of complex coefficients, i.e. polynomials in $\mathbb{C}[x]$, if we replace orthogonal matrices by unitary matrices.

The Sylvester matrix of f and g is

$$S(f,g) = \begin{bmatrix} f_n & f_{n-1} & \cdots & f_1 & f_0 & & & \\ & f_n & f_{n-1} & \cdots & \ddots & \ddots & & \\ & & \ddots & \ddots & & \ddots & \ddots & \\ & & & f_n & f_{n-1} & \cdots & f_1 & f_0 \\ g_m & g_{m-1} & \cdots & g_1 & g_0 & & & \\ & g_m & g_{m-1} & & \ddots & \ddots & & \\ & & \ddots & \ddots & & \ddots & \ddots & \\ & & & g_m & g_{m-1} & \cdots & g_1 & g_0 \end{bmatrix} \qquad (2)$$

Row Equilibration. We will henceforth assume that the input polynomials f and g have been scaled to have unit 2-norm, and thus the rows of $S(f,g)$ will also have unit 2-norm. This is known as row-equilibration, and to have beneficial effects on the conditioning of the matrix in certain circumstances. Here, it will simplify our error analysis somewhat, and increases the stability of the numerical computations, in essence replacing the condition numbers that come up in the analysis with an equivalent componentwise condition number [**?**]. This also makes the unit circle special, i.e, that $\int_C f^*(z)f(z)dz = 1$ where C is the unit circle.

Theorem 1 *[**?**] Suppose the QR factoring of (2) is $S(f,g) = QR$ where $Q \in \mathbb{R}^{(m+n)\times(m+n)}$ is orthogonal and R is upper triangular. Then, the last nonzero row of R gives the coefficients of a GCD of f and g.*

Proof. This theorem is proved in many places. See, for example, [**?**]. We include the following proof here because it helps motivate the proof for the approximate polynomial case.

From the construction of the Sylvester matrix, we have

$$\begin{bmatrix} x^{m-1}f \\ \vdots \\ f \\ x^{n-1}g \\ \vdots \\ g \end{bmatrix} = Q\cdot R \begin{bmatrix} x^{n+m-1} \\ \vdots \\ x^{n-1} \\ \vdots \\ x \\ 1 \end{bmatrix} = Q \begin{bmatrix} r_{n+m-1}(x) \\ \vdots \\ r_d(x) \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \qquad (3)$$

The polynomial r_i of degree i is formed from the $(n+m-i)$-th nonzero row of R for $i = d, d+1, \ldots, n+m-1$.

Suppose that x_k is a common root of multiplicity e_k of $f(x)$ and $g(x)$. Then one can easily verify that $S\Lambda$ is the $(m+n-1)\times e_k$ zero matrix, where Λ is the $(m+n-1)\times e_k$

matrix parameterized by x_k as follows:

$$\begin{bmatrix} x_k^{n+m-1} & (n+m-1)x_k^{n+m-2} & (n+m-1)(n+m-2)x_k^{n+m-3} & \cdots & (n+m-1)^{\underline{e_k}}x_k^{n+m-e_k} \\ x_k^{n+m-2} & (n+m-2)x_k^{n+m-3} & \cdots & & \vdots \\ \vdots & & \vdots & & (e_k-1)! \\ & & & \iddots & \\ x_k^2 & 2x_k & 2 & & \\ x_k & 1 & & & \\ 1 & & & & \end{bmatrix} \tag{4}$$

From this it is obvious that $r_d(x)$ and all its derivatives up to order e_k are zero at x_k. Conversely, if $r_d(x)$ and all its derivatives up to order e_k are zero at x_k, then by using the upper triangular structure of R we may see that $R\Lambda$ is zero. $\natural$

The GCD computation of f and g is equivalent to finding the null space of $S(f,g)$, i.e,

$$f^{(\ell)}(x) = g^{(\ell)}(x) = 0 \Longleftrightarrow S \cdot \mathbf{x}^{(\ell)} = \mathbf{0},\ \mathbf{x} = \frac{\mathbf{d}^{\ell}}{\mathbf{dx}^{\ell}}[\mathbf{x^{n+m-1}}, \cdots, \mathbf{x}, \mathbf{1}]^{\mathbf{T}} \tag{5}$$

Corollary 1 *x is a common zero of f and g if and only if x is a zero of r_d, which is the polynomial formed by multiplying the last nonzero row of R by* $\mathbf{x}$.

It is well known (see e.g. [**?**]) that QR factoring using Givens rotations or Householder transformations is numerically stable, in the following sense. Let $\hat{R}$ be a computed upper triangular factor of S obtained via Given rotations or Householder transformations. Then there exists an orthogonal $\hat{Q}$ such that

$$S + \Delta S = \hat{Q}\hat{R}, \tag{6}$$

with $\|\Delta S\|_F \le \eta \|S\|_F$, $\eta = O(\mu)$, μ is the unit roundoff, and $\|\cdot\|_F$ is the Frobenius norm. Nevertheless, the small residual $\Delta S = QR - S$ for the factoring does not guarantee a small forward error $\Delta R = \hat{R} - R$. Consider the following example.

Example 1

$$f = (x-5)(x-1/2)(56x^8 + 83x^7 + 91x^4 - 92x^2 + 93x - 91) \tag{7}$$
$$g = (x-5)(x-1/2)(32x^8 - 37x^6 + 93x^5 + 58x^4 + 90x^2 + 53) \tag{8}$$

Computing the QR factoring of $S(f/\|f\|_2, g/\|g\|_2)$ numerically for Digits=10 in Maple 7, and comparing with the exact solution (being careful about the possible nonunique orderings of factors), we obtain that

$$\|\Delta S\|_F = 0.106 \cdot 10^{-8},$$
$$\|\Delta R\|_F \ge 0.11.$$

This shows that the forward error (ΔR) may be many orders of magnitude larger than the backward error (ΔS). Due to the sensitivity of $\hat{R}$, Theorem 1 and Corollary 1 seem useless for numerically computing the GCD. For Example 1, the symbolic (exact) QR factoring gives us $r_{20} = r_{19} = 0$ and the GCD $(x-5)(x-1/2)$ can be discovered from the polynomial r_{18}. On the other hand, the numerical QR factoring gives us

$$\begin{aligned} &\vdots \\ r_{19} &= -0.1133648381x + 0.05668241903 \\ r_{20} &= -0.262 \cdot 10^{-10} \end{aligned}$$

We see that the size of r_{19} is too large to be neglected. From the equation r_{19}, the common factor $x - 1/2$ can easily be found, but the other common factor $x - 5$ is lost. The reason is shown by the following analysis.

Theorem 2 *Let f and g be given univariate approximate polynomials, with common roots x_i, $1 \le i \le k$, all lying inside the unit circle, $|x_i| \le \rho_1 < 1$. There may be other common roots not inside the unit circle. Then the QR factoring of the Sylvester matrix reveals (in the last nonzero row of R) a factor of the approximate GCD of f and g that contains the zeros x_i, $1 \le i \le k$.*

Proof.

If $S = QR$ and $S + E = \hat{Q}\hat{R}$, and S is the Sylvester matrix of $f + \Delta f$ and $g + \Delta g$ such that the null space N of S is parameterized by the zeros x_i of the GCD of $f+\Delta f$ and $g+\Delta g$, then we have that

$$(S + E)N = EN = \hat{Q}\hat{R}N$$

and so

$$\hat{R}N = \hat{Q}^* EN \,.$$

Interpreting this matrix equation as polynomial evaluation at the common zeros of $f + \Delta f$ and $g + \Delta g$, then we see that in particular the polynomial arising from the last nonzero row of $\hat{R}$, when evaluated at the common zeros, will be bounded in value by

$$|\hat{R}_d(x_j)| \le \|E\|\|N\|$$

If the roots x_j are less than 1 in magnitude, then the corresponding columns of N form a subspace N_k that also satisfies the above equation, and therefore $\|N_k\| \le c_m$ a constant that depends on the dimension of the problem, and the multiplicity of the zeros, and we see that each x_j is a pseudozero of $\hat{R}_d(x)$. By the results of [**?**], this polynomial is therefore close (in a dual norm) to the common divisor $R_d(x)$. ♮

Remark. A short calculation using the known structure of the null space shows that c_m may be taken as $(n+m)^{\underline{e}}$, where e is the maximum order of multiplicity of any zero inside the unit circle. If ρ_1 is very close to 1, then this bound may be nearly attained in practice; and if the root is of maximum possible multiplicity, namely m, then we see that this constant may grow exponentially with m, in these special circumstances. Thus the only problem here is with a highly multiple root x^* close to the unit circle (close to zero is not a problem). If there is only one such root x^*, expanding the polynomial in the basis $1, x - x^*, (x - x^*)^2, \ldots$ may improve stability.

Theorem 3 *If f and g are given as in Theorem 2, then it occurs in practice that if any x_j is outside the unit circle, it may not be detected by the QR factoring algorithm.*

Proof.
The numerical QR factoring gives us:

$$\begin{bmatrix} x^{m-1}f \\ \vdots \\ f \\ x^{n-1}g \\ \vdots \\ g \end{bmatrix} + \Delta S \begin{bmatrix} x^{n+m-1} \\ \vdots \\ x^{n-1} \\ \vdots \\ x \\ 1 \end{bmatrix} = \hat{Q}\cdot\hat{R} \begin{bmatrix} x^{n+m-1} \\ \vdots \\ x^{n-1} \\ \vdots \\ x \\ 1 \end{bmatrix} \tag{9}$$

With high probability, $\|\Delta S \cdot \mathbf{x}\| \approx \|\mathbf{\Delta S}\|\|\mathbf{x}\|$, because usually ΔS not itself a Sylvester matrix and hence $\mathbf{x}$ is not nearly in its null space. The common roots of f and g will still correspond to the null space of $\hat{Q}\hat{R}$ if and only if the perturbation term $\Delta S\cdot\mathbf{x}$ can be neglected. Suppose $\|f\|_2 = \|g\|_2 = 1$, then $\|S\|_F = \sqrt{n+m}$. If $|x| > 1$, $\|\Delta S \cdot \mathbf{x}\|_\mathbf{2}$ may increase quickly along with $m+n$. ♮

Let us check Example 1 again. For the common root $x = 1/2$,

$$\|\Delta S \cdot \mathbf{x}\|_2 \approx 0.122 \cdot 10^{-8}.$$

In contrast, for the common factor $x - 5$,

$$\|\Delta S \cdot \mathbf{x}\|_2 \approx 0.206 \cdot 10^{6}.$$

The perturbation is large enough to disrupt the null space. Therefore, it is not a surprise that the root $x = 1/2$ can be recovered from r_{19} while the other common root $x = 5$ is missing. If we compute the QR factoring for Digits=20 in Maple, $\|\Delta S\|_F = 0.1526 \cdot 10^{-18}$ and

$$\begin{aligned} &\vdots \\ r_{18} &= 0.62543 - 0.13759x + 0.025017x^2, \\ r_{19} &= 0.25195 \cdot 10^{-11} - 0.5039 \cdot 10^{-11}x, \\ r_{20} &= -0.12970 \cdot 10^{-20}. \end{aligned}$$

Both common roots can be recovered from r_{18} since $\|\Delta S \cdot \mathbf{x}\|_\mathbf{2} \leq \mathbf{10^{-10}}$ for $x = 1/2$, and also for $x = 5$, because we worked to higher precision here (and the input was in fact exact).

In the special case where all common roots of f and g lie inside the unit disc, the last "non-zero" row of R as in Theorem 1 will give us a good candidate for the GCD.

2.1. Reversals

Similarly, if all common roots of f, g lie outside the unit disc, the QR factoring of $S(\underline{f}, \underline{g})$, where $\underline{f} = x^{\deg f} f(1/x)$, $\underline{g} = x^{\deg g} g(1/x)$, are the reversals (reciprocals) of f and g, will provide us the reversal (reciprocal) of the GCD of f and g. Consequently, we can detect relatively prime numerical polynomials from the QR factors of $S(f, g)$ and $S(\underline{f}, \underline{g})$.

2.2. Relative Primality

It has been proved in [?] that a lower bound for perturbations Δf and Δg such that $f + \Delta f$ and $g + \Delta g$ have a common root is $\frac{1}{\kappa}$, where

$$\kappa = \left\| \begin{bmatrix} v & \underline{v} \\ u & \underline{u} \end{bmatrix} \right\| \tag{10}$$

can be found from $u, v, \underline{u}, \underline{v}$, which are polynomials solving the Diophantine equations:

$$f \cdot v + g \cdot u = 1, \deg u < \deg f, \deg v < \deg g \tag{11}$$

$$f \cdot \underline{v} + g \cdot \underline{u} = x^{n+m-1}, \deg \underline{u} < \deg f, \deg \underline{v} < \deg g\ , \tag{12}$$

which arise from imposing relative primality on f with g and $\underline{f}$ with $\underline{g}$. We can find this bound from the QR factoring used here, as follows. Suppose $S(f, g) = Q \cdot R$ and $S(\underline{f}, \underline{g}) = \underline{Q} \cdot \underline{R}$, $u, \underline{u}, v, \underline{v}$ are obtained from the last rows of $Q^T, R, \underline{Q}^T, \underline{R}$. Since $Q, \underline{Q}$ are orthogonal, κ is determined by the last rows of R and $\underline{R}$.

We note that the complexity of [?] is typically $O((m+n)^2)$ which is therefore "fast". Here, since the Sylvester matrix consists of two Toeplitz blocks, we can apply selected Given rotations to take advantage of the special structure of S and obtain a more efficient way for QR factoring, as in [?], the complexity is $O(n^3)$. This corresponds in complexity to the approach in [?] of first reducing to a smaller block of the Sylvester matrix by using a fast Sylvester solver to condense the system. Here we use orthogonal transformations in an effort to delay the accumulation of rounding errors.

Now that we have a stable method, we may look for ways to make it as fast as the weakly stable methods.

2.3. Common roots outside the unit circle

Since the common roots of f and g inside the unit circle are easily identified by using QR factoring, we can find an approximate common factor d_1 of f and g by QR factoring of $S(f, g)$ and another common factor d_2 by applying the QR factoring to $S(f^*, g^*)$ where $f^* = \underline{\frac{f}{d_1}}, g^* = \underline{\frac{g}{d_1}}$ are the reversals of f and g, after having divided out the common factor already found. For Example 1, after dividing out the factor $x - 1/2$, the QR factoring of f^*, g^* for Digits=10 in Maple 7 returns:

$$\begin{aligned} &\vdots \\ r_{17} &= 0.005438 - 0.02719x \\ r_{18} &= -0.267 \cdot 10^{-11} \end{aligned}$$

The common root $x = 5$ can be easily identified from r_{17}.

From our experiments with thousands of examples, of degrees up to approximately 1000, we find that about 90 percent of all problems can be solved in the above way. The algorithm even works for polynomials of high degree. See the last several examples in Table 1. This can be explained by the rapid increase of $\|\Delta S \cdot \mathbf{x}\|$ for $|x| > 1$ when the degrees of f and g are large. Therefore, there will be a clear separation of common roots inside the unit circle from common roots outside the unit circle.

Example 2 *(Random polynomials of large degree)*

$$f = -0.011637 + 0.011604x + \cdots + 0.0035539x^{1019} + 0.0044980x^{1020},$$
$$g = 0.0060163 - 0.0023432 - \cdots - 0.0067346x^{1019} + 0.012149x^{1020}.$$

Suppose $S(f,g) = QR$. We observe that the norms of the right-bottommost submatrices of R have a big jump in norm between the 16th and 15th last rows:

$$0.622 \cdot 10^{-11}, 0.146 \cdot 10^{-10}, \cdots, 0.646 \cdot 10^{-9}, 0.182 \cdot 10^{-8}, 0.376 \cdot 10^{-8}, 0.00677 \cdots$$

The 15th last row of R gives a common factor d_1 of f, g with backward error of the order 10^{-7}. The roots of d_1 are all inside the unit circle.

The QR factoring of $S(\underline{f/d_1}, \underline{g/d_1})$ gives us another common factor d_2 of degree 6 as the norm of R also has a big jump between the 7th and 6th last rows:

$$0.165 \cdot 10^{-13}, 0.439 \cdot 10^{-12}, 0.133 \cdot 10^{-11}, 0.749 \cdot 10^{-11}, 0.148 \cdot 10^{-10}, 0.320 \cdot 10^{-10}, 0.05 \cdots$$

d_2 has all its roots outside the unit circle. The details of the backward errors are given in the second last row in Table 1.

2.4. Graeffe's root-squaring to improve separation from the unit circle

Graeffe's root-squaring technique is a classical technique to transform one polynomial problem to another, hopefully simpler, problem. The basic idea is this: Suppose $f(x) = f_n(x-\alpha_1)(x-\alpha_2)\cdots(x-\alpha_n)$ is the polynomial whose zeros α_k we wish to approximate.

Consider

$$\begin{aligned} f_2(x) &= (-1)^n f(-\sqrt{x}) f(\sqrt{x}) \\ &= (-1)^n f_n^2 (x-\alpha_1^2)(x-\alpha_2^2)\cdots(x-\alpha_n^2)\,. \end{aligned} \tag{13}$$

The first equation gives us a rational means to compute the coefficients of f_2, while the second equation shows that the roots of f_2 are the squares of the roots of f. The coefficients of f_2 can be computed using the FFT in time $O(n \log n)$.

Roots less than 1 in magnitude become smaller, therefore, while roots larger than 1 become larger. Thus, each step of the root-squaring process taken improves the separation of the roots from the unit circle. One drawback is that initially close complex zeros may become more separated (in angle and in magnitude) by this process, so we do not wish to use too many root-squaring operations.

In practice, we find that only a few root-squaring steps are needed to give useful improvements to the QR factoring. However, it is not a panacea, and for difficult cases, a further refinement is needed.

3. Splitting Polynomials over the Unit Circle

In our experiments, we noticed some difficult cases where $\|\Delta S \cdot \mathbf{x}\|$ is of moderate size because there were common roots very close to the unit circle. Without a refinement of the above technique, it is hard to compute the GCD correctly in such a case. We now present one such refinement.

It is well-known that, for high degree polynomials with coefficients randomly chosen from a normal or uniform distribution, the roots cluster about the unit circle [?]. However, for many cases occurring in practice, the common roots of f and g are distributed randomly inside or outside the unit circle. If the common roots are too close to the unit circle, the numerical QR factoring may not give us correct information about the GCD.

One possible solution is to split (factor) f or g over the unit circle. For example, $f(x) = f_1(x) \cdot A(x) \cdot f_2$ where all zeros of f_1 lie inside the circle of radius $\rho_1 < 1$, all zeros of f_2 lie outside the circle of radius $\rho_2 > 1$, and $A(x)$ has all its zeros in the "ambiguous" annulus $\rho_1 \le |z| \le \rho_2$. Then GCD$(f_1, g)$, GCD$(\underline{f_2}, \underline{g})$ can be obtained by the QR factoring correctly. We will discuss this approach in more detail in a subsequent paper. For the rest of this paper, we assume that $A(x)$ can always be taken to be 1, i.e. that there are no common roots in the ambiguous annulus. This can be made more nearly true always by using Graeffe's root-squaring process. We give a brief overview of this process in a later section, but for details see any older numerical analysis text, e.g. [?]. The splitting helps the stability of the algorithm considerably.

Example 3 *(A difficult example.) Let $f = f_1 \cdot h_1$, $g = g_1 \cdot h_1$, f_1, g_1 are relatively prime and*

$$\begin{aligned} f_1 &= -26 + 31x + 18x^2 + 56x^3 + \cdots + 65x^{22} - 48x^{23} + 3x^{24} + 64x^{25}, \\ g_1 &= 37 - 31x - 45x^2 - 80x^3 + \cdots + 85x^{25} - 54x^{26} + 49x^{27} + 79x^{28}, \\ h_1 &= 1 + 14x + 90x^2 - 4x^3 + \cdots - 8x^{11} - 55x^{12} - 62x^{13} - 10x^{14}. \end{aligned}$$

The norms of the right bottommost submatrices of R increase steadily as:

$$.164 \cdot 10^{-14}, \cdots, .703 \cdot 10^{-8}, .168 \cdot 10^{-7}, .156 \cdot 10^{-5}, .157 \cdot 10^{-5}, .758 \cdot 10^{-4}, .139 \cdot 10^{-3}, .0111, \cdots.$$

It is therefore hard to obtain a good approximation to h_1.

However, if we split the polynomial $f = f_1 \cdot f_2$ over the unit circle and $S(f_1, g) = Q_1 \cdot R_1$, $S(\overline{f_2}, \overline{g}) = Q_2 \cdot R_2$, the norms of the right-bottom submatrices of R_1, R_2 have a big jump in norm:

$$0.669 \cdot 10^{-10}, 0.645 \cdot 10^{-9}, 0.140 \cdot 10^{-8}, 0.225 \cdot 10^{-8}, 0.413 \cdot 10^{-8}, 0.00152, \cdots$$

$$0.580 \cdot 10^{-13}, 0.601 \cdot 10^{-12}, 0.411 \cdot 10^{-11}, \cdots, 0.336 \cdot 10^{-9}, 0.479 \cdot 10^{-9}, 0.00201, \cdots$$

It follows that h_1 can be retrieved from the 6th last row of R_1 and the 10th last row of R_2. In this case, there are no roots too close to the unit circle for this refinement, and h_1 is correctly recovered by this refined technique.

The splitting can be performed in a classical way, using contour integrals. This method has been discussed by many authors [?, ?, ?, ?] as a tool for finding all roots of polynomials. The main steps are given in the following algorithm. It is time-consuming to evaluate the contour integrals to high accuracy. Therefore, the algorithm first splits the polynomial to a relatively low accuracy and then refines the factoring by an iterative method. The first step of root-squaring is used to push the roots away from the unit circle. The FFT is used to accelerate the computation in steps 1 and 2.1. Unlike the algorithm in [?], which uses the fast methods available for Padé approximation in Step 3, we recursively make use of QR

factoring for GCD computation. The reason is that polynomials $p_{k-j}(x)$ and $G_{k-j+1}(x^2)$ only have common zeros outside the unit circle since the all roots of G_{k-j+1} lie outside the unit circle. Step 2.2.1 can also use QR factoring for the same reason because the roots of F_k and G_k are well separated by the unit circle.

Step 2.2.2 needs a good algorithm for approximate division. See [**?**] for a description of the algorithm that we have used.

Algorithm Split

Input: One monic univariate polynomial $p(x)$ with degree n, radius $r = 1$, tolerance ϵ.

Output: Univariate polynomials F, G, such that $\|p - F \cdot G\| < \epsilon$, split by the unit circle and center $(0, 0)$.

Step 1. [recursive lifting] Apply k root-squaring Graeffe's steps (usually k is 1, 2 or 3)

$$p_{i+1}(x) = (-1)^n p_i(-\sqrt{x})p_i(\sqrt{x}), \quad i = 0, 1, ..., k-1.$$

Step 2. [splitting p_k]

2.1 [rough approximation of F and G]

2.1.1 Compute

$$s_N = \frac{1}{2\pi i}\int_C x^N \frac{p_k'(x)}{p_k(x)} dx = \sum_{i=1}^{v} {z_i}^N$$

where C is the unit circle; z_i are all the roots of p_k inside C, i.e.,the roots of F_k; v is the integer closest to s_0, i.e., the number of zeros of p_k inside C.

2.1.2 From $s_1, s_2, ..., s_v$, compute the coefficients of the polynomial F_k.

2.1.3 Compute $G_k = \text{quo}(p_k,\, F_k)$.

2.2 [Newton's iteration, one step]

2.2.1 Compute u and v such that

$$u \cdot F_k + v \cdot G_k = 1.$$

2.2.2 Compute ΔF_k and ΔG_k in higher precision as

$$\begin{aligned} \Delta F_k &= \text{polyrem}(p_k \cdot v,\, F_k) \\ \Delta G_k &= \text{polydiv}(p_k - F_k \cdot G_k - \Delta F \cdot G_k,\, F_k) \end{aligned}$$

2.2.3 Set $F_k = F_k + \Delta F_k, \quad G_k = G_k + \Delta G_k$.

Step 3. [recursive descending] For i from 1 to k do

$$\begin{aligned} G_{k-j} &= \text{gcd}(p_{k-j}(x),\, G_{k-j+1}(x^2)) \\ F_{k-j} &= \text{polydiv}(p_{k-j},\, G_{k-j}) \end{aligned}$$

4. The Algorithm for GCD Computation

Algorithm GCD

Input: Two univariate polynomials $f(x)$ and $g(x)$, tolerance ϵ.

Output: Univariate polynomials u, v, d such that $\|f - d \cdot f_1\| < \epsilon$, $\|g - d \cdot g_1\| < \epsilon$, $\|uf + vg - d\| < \epsilon$ and $\deg(u) < \deg(g) - \deg(d)$, $\deg(v) < \deg(f) - \deg(d)$.

Note that $\|uf + vg - d\| < \epsilon$ can be rewritten as $\|u\Delta f + v\Delta g\| < \epsilon$, where $\Delta f = f - d \cdot f_1$ and $\Delta g = g - d \cdot g_1$. This represents an extra constraint on u and v, and thus disallows them from growing to be too large.

```
Step 1. [Initialization]
    1.1 Make the input $f$ and $g$ to be unit 2-norm with positive leading coefficients.
Step 2. [$QR$-factoring]
    2.1 Form the Sylvester matrix $S$ of $f$ and $g$.
    2.2 Compute the $QR$–factoring for $S = Q \cdot R$.
    2.3 Suppose $R_{22}^{(k)}$ are the last $(k+1) \times (k+1)$ submatrices of $R$ such that $\|R_{22}^{(k)}\| > \epsilon$
        but $\|R_{22}^{(k-1)}\| < \epsilon$.
    case 0. [ $\|R_{22}^{0}\| > \epsilon$]: $d_1 = 1$, $u$ and $v$ are formed by the last row of $Q^T$.
    case 1. [ $\frac{\|R_{22}^{(k)}\|}{\|R_{22}^{(k-1)}\|} > 0.1/\epsilon$]: $d_1$'s coefficients are given by the first row of $R_{22}^{(k)}$.
    case 2. [ $\exists\, k_1$(biggest) such that $\frac{\|R_{22}^{(k_1)}\|}{\|R_{22}^{(k_1-1)}\|} > 0.1/\epsilon$]: $d_1$'s coefficients are given by the
            first row of $R_{22}^{(k_1)}$.
    case 3. [Difficult case]: Use the algorithm **Split** to find the common roots of $f$ and
            $g$ inside the unit circle and form the divisor $d_1$.
Step 3. [Coprime check]
    3.1 Compute cofactors $f_1$ and $g_1$:
            $f_1 = \mathrm{polydiv}(f, d_1)$,
            $g_1 = \mathrm{polydiv}(g, d_1)$.
    3.2 Apply Step 2 to $x^{\deg(f_1)} \cdot f_1(x^{-1})$, $x^{\deg(g_1)} \cdot g_1(x^{-1})$ to obtain $d_2$.
    3.3 Apply Step 2 to cofactors of $f, g$ w.r.t. $d = d_1 \cdot d_2$ to obtain $u$, $v$(case 0).
Step 4. Return $u, v, d$.
```

5. Multiple Common Roots

The method given in this paper has no difficulty finding accurate common factors of problems that have multiple approximate common roots; however, it is the coefficients of the factors with multiple roots that are recovered, not the multiple roots themselves. To accurately find the multiple roots from these approximate common factors requires a separate analysis.

6. Theoretical Complexity Analysis

Suppose that the degree of f is n and the degree of g is m, and $n \geq m$. The complexity of the main steps of Algorithm GCD (page 33) are:

1. Step 2.1: $O(n^3)$ (see Section 1.2.)
2. Step 2.3: see below (Algorithm Split)
3. Step 3.1: $O(n^2)$, because the matrix involved in the polynomial division is a Toeplitz matrix. The complexity can be achieved by using the algorithm in [?].

Supposing that the degree of the polynomial to be split is n, the complexity of the main steps in Algorithm Split (page 32) are:

1. Step 1. $O(kn \log n)$, k is usually 1,2, or 3.
2. Step 2.1 $O(n \log^2 n)$.
3. Step 2.2.1 $O(n^2)$. Since the roots of two factors are well separated (with respect to the unit circle), the Sylvester matrix is well conditioned. Moreover, Sylvester matrix is quasi-Toeplitz matrix, the method in [?] gives a fast stable way to find u and v.
4. Step 2.2.2. $O(n^2)$.
5. Step 3. $O(n^3)$. We apply QR factoring to the reciprocal of the two polynomials since the two polynomials only have common roots outside the unit circle.

7. Test Results

7.1. Comparison with Example 2 of [1]

The Example 2 of [?] is as follows. Let $A(z) = d(z)A_1(z)$ and $B(z) = d(z)B_1(z)$, where

$$\begin{aligned} d(z) &= z^5 - 0.6z^4 - 0.05z^3 - 0.05z^2 - 1.05z + 0.55 \qquad (14)\\ B_1(z) &= z^9 + 1.95z^8 + 0.6699z^7 + 0.1978z^6 + 0.2271z^5 \\ &\quad - 1.5652z^4 - 1.99118z^3 - 0.7413z^2 - 0.0801z + 0.0634 \qquad (15)\\ A_1(z) &= z^{10} - 1.6z^9 + 2.43z^8 - 1.148z^7 + 1.2248z^6 \\ &\quad + 1.3875z^5 - 0.9895z^4 + 0.9751z^3 - 0.7813z^2 - 0.623z + 0.0692\,. \qquad (16) \end{aligned}$$

We note that $d(z)$ has one root inside the unit circle, and four outside. Using the technique of this paper, the root inside the unit circle is easily found by a QR factoring of the Sylvester matrix of A and B, and the four roots outside are found by a QR factoring of the reversals of A and B.

The paper [?] reported a failure of the condition estimator of the method of that paper. We believe that this failure was, in essence, caused by the fact that some of the common roots were inside and some were outside of the unit circle. The improvement of this present paper is sufficient to allow this example to be solved in a straightforward way, even without the contour integral splitting refinement.

Assume $A(z), B(z)$ are perturbed by the noise uniformly distributed over the interval $[-10^{-4}, 10^{-4}]$, for example:

$$p_1 = -0.000045\,z - 0.000026\,z^{14} + 0.000027\,z^{13} + 0.000007\,z^7 + 0.000031\,z^2 + 0.000097\,z^5,$$
$$p_2 = 0.000067\,z^{10} + 0.000077\,z^{11} + 0.000009\,z^9 - 0.00009\,z^8 + 0.000017\,z^7 - 0.000007\,z^2.$$

```
u,v,G:=GCD(A,B,z,10^(-4));
GCDAux1:   "the norm of last row"   .284656437560687029e-7
GCDAux1:   "the norm of row i"   .8316275671e-5
GCDAux1:   "the norm of row i"   .1481225254e-4
GCDAux1:   "the norm of row i"   .7056767811e-4
GCDAux1:   "the norm of row i"   .2704901034e-3
GCDAux1:   "difficult case"   3.833059421
evalpower:   "the number of evaluation points"   128
newtoncorr:   "backward error before Newton correction"
.354778554787317338890e-8
newtoncorr:   "backward error after  Newton correction"
.494576642755856003l8e-14
liftsplit:   "the recursive lifting"   2
gcd:   "the norm of last row"   .22953160554339941600e-19
gcd:   "the norm of row i"   .466052253477438643 45e-19
gcd:   "the norm of row i"   .337929733532576706 83e-18
gcd:   "the norm of row i"   .975512977864364667 87e-17
gcd:   "the norm of row i"   .37999540616867081304
liftsplit:   "the recursive lifting"   1
gcd:   "the norm of last row"   .106642541366968712 63e-18
gcd:   "the norm of row i"   .267301840882935266 29e-18
gcd:   "the norm of row i"   .470710960065758619 55e-18
gcd:   "the norm of row i"   .725944217953661048 27e-18
gcd:   "the norm of row i"   .49635707831783407565
GCDAux1:   "Degree of GCD and backward error for f,g"   1
.2775811604e-5   .5520080356e-5
GCDAux2:   "the norm of last row"   .687906778348371770e-6
GCDAux2:   "the norm of row i"   .1446564243e-5
GCDAux2:   "the norm of row i"   .3282314241e-5
GCDAux2:   "the norm of row i"   .1167654451e-4
GCDAux2:   "the norm of row i"   .1251891406
GCDAux2:   "quick decrease"   10721.42024
```

```
GCDAux2:   "Degree of GCD and backward error for f,g"   4
.1128148472e-4   .8291147487e-5
```

$$\begin{aligned}
u &= -2.535456495 - 13.23934493\,z - 25.83016828\,z^2 - 17.92465245\,z^3 + 3.008400870\,z^4\\
&\quad +4.855307551\,z^5 + 11.93468274z^6 + 24.99521476\,z^7 + 12.24668103\,z^8,\\
v &= 10.77657120 + 3.167861608\,z - 3.131239168\,z^2 - .3687451268\,z^3 - 9.197080343\,z^4\\
&\quad -21.26844693z^5 + 15.19657543\,z^6 - 29.53533751\,z^7 + 18.48136619\,z^8 - 12.24726791z^9,\\
G &= .279076256 - .7608296186\,z - .0253504943\,z^2 - .02532861052\,z^3 - .3044336272\,z^4\\
&\quad +.5072422832\,z^5.
\end{aligned}$$

Suppose we start with the QR-factorization of reciprocal of A and B, then no splitting is needed. Since the final results are quite similar to the u, v, G above, we omit them here.

```
GCDAux2:   "the norm of last row"   .827778337414056300e-6
GCDAux2:   "the norm of row i"   .1718129308e-5
GCDAux2:   "the norm of row i"   .2883180742e-5
GCDAux2:   "the norm of row i"   .1320829125e-4
GCDAux2:   "the norm of row i"   .1195369248
GCDAux2:   "quick decrease"   9050.143015
GCDAux2:   "Degree of GCD and backward error for f,g"   4
.7347826910e-5   .7980726203e-5
GCDAux1:   "the norm of last row"   .102517511356116598e-6
GCDAux1:   "the norm of row i"   .1774173932e-2
GCDAux1:   "quick decrease"   17306.05735
GCDAux1:   "Degree of GCD and backward error for f,g"   1
 .7987969771e-5   .1561375946e-4
```

7.2. Summary of tests with high degree random polynomials

Table 1: backward errors for algorithm GCD

$(\deg(f), \deg(g))$	$\deg(d_1)$	$\deg(d_2)$	$\|u \cdot f + v \cdot g - d\|_2$	$\|f - d \cdot f_1\|_2$	$\|g - d \cdot g_1\|_2$
(30, 25)	8	2	0.326e-6	0.718e-7	0.926e-7
(42, 32)	11	1	0.174e-6	0.981e-7	0.780e-7
(55, 50)	18	2	0.553e-5	0.404e-5	0.356e-5
(55, 50)*	10	10	0.154e-7	0.651e-7	0.336e-7
(65, 45)	3	12	0.208e-6	0.151e-6	0.523e-6
(65, 55)*	14	11	0.156e-5	0.470e-6	0.467e-6
(65, 55)	21	4	0.820e-8	0.136e-7	0.129e-7
(75, 65)	14	1	0.890e-6	0.389e-6	0.168e-6
(80, 60)	19	1	0.371e-4	0.642e-5	0.290e-5
(80, 60)*	10	10	0.349e-7	0.719e-7	0.862e-7
(100, 80)	36	4	0.216e-3	0.210e-4	0.193e-4
(100, 80)*	18	22	0.123e-7	0.388e-7	0.409e-7
(110, 90)	5	5	0.270e-7	0.321e-8	0.684e-8
(155, 155)	97	6	0.628e-4	0.367e-4	0.478e-4
(159, 159)	99	5	0.203e-4	0.389e-6	0.714e-6
(168, 101)*	33	26	0.632e-7	0.233e-7	0.198e-7
(166, 79)*	28	33	0.411e-7	0.217e-6	0.213e-6
(156, 78)*	28	26	0.702e-5	0.631e-6	0.740e-6
(138, 83)*	30	20	0.7335e-6	0.213e-6	0.131e-6
(132, 128)*	19	29	0.193e-5	0.259e-6	0.239e-6
(208, 108)	4	4	0.192e-4	0.293e-5	0.208e-5
(220, 120)	9	11	0.277e-4	0.258e-4	0.156e-4
(320, 220)	12	8	0.381e-6	0.266e-7	0.214e-7
(510, 210)	4	6	0.322e-6	0.108e-8	0.912e-8
(530, 230)	18	12	0.327e-5	0.112e-5	0.127e-5
(1020,1020)	14	6	0.192e-6	0.183e-7	0.187e-7
(1024,1022)	10	10	0.282e-5	0.421e-8	0.227e-8
(1020,1020)	8	12	0.121e-3	0.247e-5	0.877e-6

The examples with (*) mean difficult cases and the splitting algorithm is needed.

8. Concluding Remarks

This paper identifies a difficulty with previous attempts at practical methods for the computation of approximate GCD, and presents an improved alternative together with an error analysis, theoretical complexity analysis, and experimental results on several thousand examples. The method used in this paper seems to be of potential use in practice, for polynomials of moderately large degree (up to about 1000).

One open problem is what to do about common roots in the ambiguous annulus $\rho_1 < |x| < \rho_2$, and we will pursue this in a future paper. Another open problem is whether fast $O(n^2)$ QR factoring [?] can be stably used in this context.

Acknowledgments

Lihong Zhi would like to thank Prof. Matu-Tarow Noda for useful suggestions.