Seeking Better Algorithms for Approximate GCD

Kosaku Nagasaka
Kobe University
Kobe, JAPAN, 657-8501
`nagasaka@main.h.kobe-u.ac.jp`

1 Introduction

We are interested in seeking better algorithms by just slightly updating and combining the well-known algorithms for the following well-known problem in Symbolic-Numeric Computations.

Definition 1 (Approximate Polynomial GCD)
For the given polynomials $f(x), g(x) \in \mathbb{C}[x]$, we compute the polynomial $d(x) \in \mathbb{C}[x]$ called "approximate polynomial GCD" of tolerance $\varepsilon \in \mathbb{R}_{\geq 0}$, which has the maximum degree[1] *and satisfies*

$$f(x) + \Delta_f(x) = f_1(x)d(x),\ g(x) + \Delta_g(x) = g_1(x)d(x)$$

for some polynomials $\Delta_f(x), \Delta_g(x), f_1(x), g_1(x) \in \mathbb{C}[x]$ such that $\deg(\Delta_f) \leq \deg(f)$, $\deg(\Delta_g) \leq \deg(g)$, $\|\Delta_f\|_2 < \varepsilon \|f\|_2$ and $\|\Delta_g\|_2 < \varepsilon \|g\|_2$ where $\|\cdot\|_2$ denotes the 2-norm. $\triangleleft$

For this problem, we focus on recent and/or famous algorithms: QRGCD[3] (and its extended version: ExQRGCD[4]), UVGCD[5] and Fastgcd[1]. Actually we update their original algorithms slightly and combine their sub-algorithms used in their similar frameworks (Table 1) to seek better algorithms. The result indicates that our approach improves their performance drastically (especially ExQRGCD which becomes competitive with UVGCD and Fastgcd, and possibly better).

2 Performance Tests

We implemented the following combinations that are represented by the regular expressions where the symbols `q`, `e`, `u` and `f` denote the subroutines of QRGCD, ExQRGCD, UVGCD and Fastgcd, respectively. Moreover, the program is linked to the single thread ATLAS and LAPACK libraries if the prefix is `s`, and the threaded (parallel thread) ATLAS and LAPACK libraries if it is `t`.
`(s|t)-qrgcd-(q|u|f)`: QRGCD with the refinement method specified by `q`,`u`,`f`.
`(s|t)-exqrgcd-(e|u|f)`: ExQRGCD with the refinement method specified by `e`,`u`,`f`.
`(s|t)-fastgcd-(f|r)(f|u|s|t)(f|u)(f|u)`: The 1st grouping defines the method of candidate degree where `r` denotes the method of PivQR[2]. The 2nd defines the method of initial cofactors where `s` denotes UVGCD with column pivoting and `t` denotes UVGCD with UVGCD's pivoting strategy. The 3rd defines the method of initial approximate GCD. The last defines the refinement.
others: `(s|t)-qrgcd-p`, `(s|t)-exqrgcd-p` and `(s|t)-uvgcd-p` denote QRGCD, ExQRGCD and UVGCD with pivoting (we extended the algorithms to be compatible with pivoting).

[1]The algorithms appeared here do not certify this property but try to find a factor of nearly maximum degree.

	candidate degree	cofactors	approximate GCD	refinement	loop structure
QRGCD	QR	Least Square	row of R	-	finding inside, outside roots
ExQRGCD	QR	Least Square	row of R	-	repeatedly inside, outside roots
UVGCD	smallest singular value	right singular vector	Least Square	Gauss-Newton with QR	unidirectional degree search
Fastgcd	LU with GKO	Ker(U)	FFT	Gauss-Newton with GKO	bidirectional degree search

Table 1: Framework (with mainly used approaches)

We have prepared the following tolerance sensitive problems where the 2-norm of perturbed polynomials (as noisy error) in each pair is 10^{-8}. These tests with the input tolerance $\varepsilon = 10^{-8}$ are not easy task since the expected tolerance of the expected degree is mostly in $[10^{-8}, 10^{-7}]$.
half degree: For $k \in \mathbb{Z}_{>0}$, we have generated 100 pairs of polynomials of degree $10k$, of unit 2-norm. Each pair has GCD of degree $5k$. Each factor has random integer coefficients in $[-99, 99]$ before normalization. We added the same degree polynomials whose norm is 10^{-8}, and re-normalized.
low degree: Similar to the half degree except that each pair has the GCD of degree k.
asymmetric: Similar to the low degree except that degrees of each polynomials are $2k$ and $18k$.

We proceeded a set of tests with GNU C Compiler 4.8.4 (optimized with -O3 -march=corei7-avx), ATLAS 3.11.38 (as BLAS), LAPACK 3.6.0 (through LAPACKE) and FFTW 3.3.3 (Fastgcd requires FFT) on Ubuntu 14.04 LTS (x86_64, 3.19.0) with Intel Core i7-5960X and 64GB memory. Please note that we have not done any opimization on the code, and QRGCD, UVGCD and Fastgcd can be considered as non-official implementations though ExQRGCD is since ExQRGCD is proposed by the present author. Therefore, timing data should be considered as just a reference though the resulting tolerances are better than the official implementation of Fastgcd.

Table 2 shows a partial result (only originals and notable combinations for $k = 10$) of single thread versions since threaded versions are not faster for this size of problems according to our results. Each cell has 2 numbers that are the averages of detected degree of approximate GCD and the elapsed time (sec.) (not CPU time), respectively. Please note that we ran the programs once for each pair $f(x)$ and $g(x)$ in this order and also once for the reverse order (i.e. $g(x)$ and $f(x)$) to avoid any effect of input ordering (the tables show the averages).

Moreover, we also examined the performance for the example 8.9.2(1) introduced by Boito[2]: let $n_1 = 25k$, $n_2 = 15k$, $n_3 = 10k$ and define the input polynomials $f(x) = f_1(x)d(x)$ and $g(x) = g_1(x)d(x)$, where $f_1(x) = (x^{n_1} - 1)(x^{n_2} - 2)(x^{n_3} - 3)$, $g_1(x) = (x^{n_1} + 1)(x^{n_2} + 5)(x^{n_3} + \imath)$ and $d(x) = x^4 + 10x^3 + x - 1$ (this is the expected GCD). Figure 1 shows the timing data of some combinations. As reported by Boito[2], UVGCD is weak for polynomials with GCD of low degree.

3 Conclusion

Implementing modern algorithms for approximate GCD has various possibilities: pivoting, finding over $\mathbb{R}$ or $\mathbb{C}$, selecting the subroutines and matrix factoring methods. Our numerical experiments indicate the followings: 1) QRGCD (including ExQRGCD) and UVGCD are compatible with the QR factorization with pivoting however pivoting does not improve their performance. 2) Fastgcd with the small modification is able to find approximate GCDs over $\mathbb{R}$ and it works as well as other

	half degree		low degree		asymmetric	
`s-qrgcd`	0.00 (0.009)	0.00 (0.023)	0.64 (0.004)	0.64 (0.009)	1.02 (0.004)	1.02 (0.009)
`s-qrgcd-u`	0.00 (0.009)	0.00 (0.023)	0.64 (0.004)	0.64 (0.010)	1.02 (0.005)	1.02 (0.011)
`s-exqrgcd`	0.00 (0.033)	0.00 (0.093)	2.44 (0.029)	2.44 (0.078)	1.05 (0.014)	1.06 (0.037)
`s-exqrgcd-u`	47.84 (0.028)	47.84 (0.076)	10.00 (0.015)	10.00 (0.040)	10.00 (0.016)	10.00 (0.042)
`s-uvgcd`	50.00 (0.015)	50.00 (0.029)	10.00 (0.051)	10.00 (0.092)	10.00 (0.017)	10.00 (0.035)
`s-fastgcd`	24.48 (0.865)	34.73 (1.442)	7.64 (0.368)	7.91 (0.290)	4.36 (0.425)	5.32 (0.470)
`s-fastgcd-fffu`	47.23 (0.044)	48.33 (0.059)	9.62 (0.039)	9.89 (0.077)	6.67 (0.060)	7.49 (0.103)

Table 2: Performance Test (finding over $\mathbb{R}$ (left) and $\mathbb{C}$ (right))

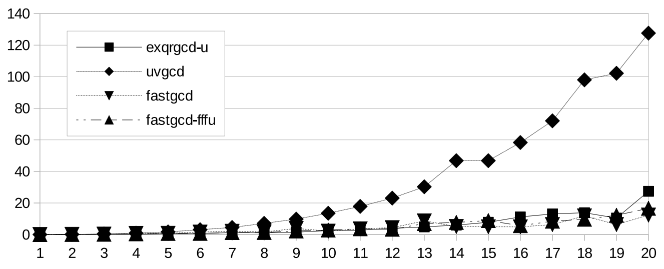

Figure 1: Boito 8.9.2(1) (elapsed time for single thread versions)

algorithms, which is originally designed for over $\mathbb{C}$ though. 3) In practise, the Gauss-Newton refinement of UVGCD is much better than that of Fastgcd, and ExQRGCD with the refinement method of UVGCD works very well though it was not expected before. 4) In theory, the computational complexity of Fastgcd is much better than others. However, the performance is not well. One of reasons may be that its special LU factorization (the modified GKO method) is not directly supported in LAPACK though we implemented it with many sub-routines of BLAS. We note that this work was supported by JSPS KAKENHI Grant Number 15K00016.

References

[1] D. A. Bini and P. Boito. Structured matrix-based methods for polynomial ϵ-gcd: Analysis and comparisons. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 9–16, New York, NY, USA, 2007. ACM.

[2] P. Boito. *Structured Matrix Based Methods for Approximate GCD.* Ph.D. Thesis. Department of Mathematics, University of Pisa, Italia, 2007.

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

[4] K. Nagasaka and T. Masui. Extended qrgcd algorithm. In *Proceedings of the 15th International Workshop on Computer Algebra in Scientific Computing - Volume 8136*, CASC 2013, pages 257–272, New York, NY, USA, 2013. Springer-Verlag New York, Inc.

[5] Z. Zeng. The numerical greatest common divisor of univariate polynomials. In *Randomization, relaxation, and complexity in polynomial equation solving*, volume 556 of *Contemp. Math.*, pages 187–217. Amer. Math. Soc., Providence, RI, 2011.