Solving Approximate GCD of Multivariate Polynomials By Maple/Matlab/C Combination

Kai Li   Lihong Zhi   Matu-Tarow Noda
Department of Computer Science
Ehime University, Japan
{likai,lzhi,noda}@hpc.cs.ehime-u.ac.jp

Abstract

This paper introduces an approach based on EZ-GCD algorithm to compute GCD of multivariate polynomials with floating-point coefficients. We discuss various issues related to the implementation of the algorithm and present some preliminary test results.

1 Introduction

The problem of computing approximate GCD of polynomials with inexactly-known coefficients has been well studied recently [2] [5] [6] [8] [11] [13] [14] [15] . In most of the previous works, the focus was on finding the approximate GCD of univariate polynomials.

The standard algebraic algorithms for computing GCD of multivariate polynomial with integer coefficients are based on subresultant polynomial remainder sequence (PRS), or modular homomorphisms and their inverse mappings. In [5] and [2], subresultant PRS algorithm and modular algorithms based on Chinese remainder theorem are modified to compute approximate GCD of multivariate polynomials. Two methods have their own advantages and disadvantages (see [16]). In [16], an approach that modifies the modular algorithm based on Hensel construction was proposed and discussed briefly. In this paper, we detail in the implementation of the algorithm using Maple, Matlab and C.

In section 2, we introduce our notation and describe the standard EZ-GCD algorithm [3]. Substantial modifications to compute GCD of multivariate polynomials are discussed when the algorithm is extended to polynomials with complex or real number coefficients. In section 3 we describe the design of our package which combines the algorithms in Maple, Matlab and C. The results from a number of numerical experiments are presented to demonstrate the efficiency of our package and to show why it is necessary to write some of the algorithms in Matlab or C instead of using Maple routines. We give summaries and conclusions in section 4.

2 Multivariate GCD

The problem we consider here is: Given two multivariate polynomials $F(x_1, \cdots, x_n)$ and $G(x_1, \cdots, x_n)$ with coefficients of limited accuracy, are there nearby polynomials with a non-trivial GCD for a given tolerance ϵ? Using notation in [4], we give a formal definition of approximate GCD.

Definition 1 *An empiric polynomial* $P = (\bar{P}, \epsilon)$*, with* $\bar{P} = \Sigma_{j \in J} \bar{\alpha}_j x^j \in \mathbb{C}[x]$*, where* $x = (x_1, ..., x_n)$, $\bar{\alpha}_j \in \mathbb{C}$, $J \in \mathbb{N}_0^n$*, defines a family of polynomial neighborhoods*

$$N_\delta(\bar{P}, \epsilon) := \{\tilde{P} \in \mathbb{C}[x] \, : \, \frac{||\tilde{P} - \bar{P}||}{\epsilon} \leq \delta\} \, , \ \delta \geq 0 \, .$$

Where $|| \cdot ||$ *denotes the maximal norm. The polynomials in* $N_1(\bar{P}, \epsilon)$ *will be considered as indistinguishable.*

Definition 2 $C \in \mathbb{C}[x]$ *is a* δ*-GCD of the empiric polynomials* F *and* G *if there exist polynomials* $\tilde{F} \in N_\delta(\bar{F}, \epsilon)$ *and* $\tilde{G} \in N_\delta(\bar{G}, \epsilon)$ *such that* C *is a genuine GCD of* $\tilde{F}, \tilde{G}$*. A 1-GCD is simply called a valid approximate GCD of* F *and* G*.*

The EZ-GCD algorithm reduces two input polynomials to two univariate polynomials whose GCD is then lifted back to the multivariate domain using a generalized Newton's iteration. It is essential to choose a good evaluation point, discussed later, so that the homomorphic GCD is relatively prime to one of its cofactors. Now let us concentrate on Hensel construction. For simplicity, suppose we are given polynomials $F(x, y)$, $G^{(0)}(x)$, and $H^{(0)}(x)$ such that:

$$F \equiv G^{(0)} H^{(0)} \mod y. \tag{1}$$

Assume we want to lift the two univariate factors to degree n in y. At step k of Hensel lifting, we want to compute $G^{(k)}(x, y)$ and $H^{(k)}(x, y)$ such that:

$$F \equiv G^{(k)} H^{(k)} \mod y^{k+1} \tag{2}$$

and

$$G^{(k)} \equiv G^{(k-1)} \mod y^k, \quad H^{(k)} \equiv H^{(k-1)} \mod y^k. \tag{3}$$

For (3) to hold, we set:

$$\begin{aligned} G^{(k)} &:= G^{(k-1)} + \Delta_1^{(k)} y^k, \\ H^{(k)} &:= H^{(k-1)} + \Delta_2^{(k)} y^k \end{aligned} \tag{4}$$

where $\Delta_i^{(k)} (i = 1, 2)$ are univariate polynomials in $\mathbb{C}[x]$. Plugging (4) into (2), we see that lifting from y^k to y^{k+1} amounts to solving for the $\Delta_i' s$ in the following univariate Diophantine equation:

$$\frac{F - G^{(k-1)} H^{(k-1)}}{y^k} \equiv \Delta_1^{(k)} H^{(0)} + \Delta_2^{(k)} G^{(0)} \mod y. \tag{5}$$

If $G^{(0)}(x)$ and $H^{(0)}(x)$ are relatively prime, then (5) always has a unique solution which can be computed by the solutions of $\sigma_i^{(k)}$ for

$$\sigma_1^{(k)} H^{(0)} + \sigma_2^{(k)} G^{(0)} = x^k. \tag{6}$$

Suppose

$$\begin{aligned} G^{(0)}(x) &= g_s x^s + g_{s-1} x^{s-1} + \cdots + g_1 x + g_0, g_s \neq 0, \\ H^{(0)}(x) &= h_t x^t + h_{t-1} x^{t-1} + \cdots + h_1 x + h_0, h_t \neq 0. \end{aligned}$$

Then (5) is reduced to solving linear algebra equations

$$M \quad \vec{z} = \vec{b}, \tag{7}$$

where $\vec{b}$ is the coefficient vector of polynomial on the left side of (5), $\vec{z}$ is the coefficient vector of polynomial $\Delta_i^{(k)}(i = 1, 2)$; M is Sylvester matrix of polynomial $G^{(0)}$ and $H^{(0)}$, i.e.,

$$M = \begin{bmatrix} g_s & g_{s-1} & & \cdots & \cdots & & 0 & 0 \\ & g_s & g_{s-1} & & \cdots & \cdots & & 0 \\ & & \ddots & \ddots & & & \vdots & \vdots \\ & & & g_s & g_{s-1} & \cdots & g_1 & g_0 \\ h_t & h_{t-1} & & \cdots & \cdots & & 0 & 0 \\ & h_t & h_{t-1} & & \cdots & \cdots & & 0 \\ & & \ddots & \ddots & & & \vdots & \vdots \\ & & & h_t & h_{t-1} & \cdots & h_1 & h_0 \end{bmatrix}^T . \tag{8}$$

If we proceed as in symbolic computation, i.e, solving (6), it is equivalent to compute the inverse of the matrix M. From the view of numerical computation, it is unstable and also time consuming to solve linear equations by computing inverse of the matrix. From other numerical methods such as LU, QR or SVD decomposition [7], we select QR decomposition as it is easy to exploit the structure of the Sylvester matrix M for an efficient and stable algorithm. Since the Sylvester matrix M consists of two Toeplitz matrices, if we apply the Given rotation method to the row 1 and $t+1$ then the row i and $t+i$ for i from 2 to t can be changed accordingly. So we start the QR decomposition with the Given rotation to eliminate the elements below the diagonal of the first t columns, and then apply the Householder transformations to the lowest $s \times s$ submatrix (suppose $s >= t$).

The advantage of combining Given rotations with Householder transformations can be seen clearly from comparing complexity in the case $s = t = n$. The flops used in general LU, QR and SVD decomposition are $\frac{2}{3}(2n)^3$, $\frac{4}{3}(2n)^3$ and $12(2n)^3$; while using the above strategy, the flops we used are $6n^2 + \frac{4}{3}n^3$.

If $G^{(0)}(x)$ and $H^{(0)}(x)$ has an approximate GCD, then M will be near rank deficient. So it is necessary to estimate the condition number of the matrix M before we start the Hensel lifting. If M is near singular, we have to choose other evaluation points, other main variable, or even try the squarefree decomposition of the polynomials F or G.

Another important issue related to Hensel construction is when to stop the Hensel lifting. In the symbolic computation, the process will stop after $k \geq \deg_y(F)$. For example,

$$F = (x + 2 + y)(x + 151/100 + 4y - 2y^2 + y^3) + \eta(x + y)$$

when $\eta = 0$:

$$\begin{aligned} F &\equiv (x + 2)(x + 151/100) \mod y, \\ F &\equiv (x + 2 + y)(x + 151/100 + 4y) \mod y^2, \\ F &\equiv (x + 2 + y)(x + 151/100 + 4y - 2y^2) \mod y^3, \\ F &\equiv (x + 2 + y)(x + 151/100 + 4y - 2y^2 + y^3) \mod y^4, \\ F &\equiv (x + 2 + y)(x + 151/100 + 4y - 2y^2 + y^3) \mod y^5. \end{aligned}$$

The lifting from y^4 to y^5 contribute nothing new and both factors remain unchanged. Moreover, the first factor $x+2+y$ stays unchanged from the second step onwards. However, in the floating-point computation case, things are different. Suppose $\eta = \frac{1}{10^4}$(rounding to 4 digits):

$$\begin{aligned}
F &\equiv (x+2.)(x+1.51) \mod y,\\
F &\equiv (x+2.+1.002y)(x+1.51+3.998y) \mod y^2,\\
F &\equiv (x+2.+1.002y+0.01357y^2)\\
&\quad (x+1.51+3.998y-2.014y^2) \mod y^3,\\
F &\equiv (x+2.+1.002y+0.01357y^2+0.07349y^3)\\
&\quad (x+1.51+3.998y-2.014y^2+0.9265y^3) \mod y^4,\\
F &\equiv (x+2.+1.002y+0.01357y^2+0.07349y^3+0.3979y^4)\\
&\quad (x+1.51+3.998y-2.014y^2+0.9265y^3-0.3979y^4) \mod y^5.
\end{aligned}$$

Both factors continue to include terms with high degree w.r.t. y. Although the coefficients of y^2 and y^3 in the first factor are relatively small, the computation becomes more unreliable when the power of y increase to 4 and 5. So rather than lifting to the full degree w.r.t. y, we prefer to estimate the degrees of y in the factors, and stop the lifting as soon as one factor arrives at it. We try to get the full expression of the other factor by solving an overdetermined linear equation system. For the above example, we let $G = x + 2. + 1.002y$ be a candidate factor, and suppose another factor be

$$H = x + a_1 + a_2 y + a_3 y^2 + a_4 y^3.$$

From a comparison of coefficients in $F = G \cdot H$ we obtain the overdetermined linear equation system:

$$\begin{bmatrix} 1. & 0 & 0 & 0 \\ 0 & 1. & 0 & 0 \\ 0 & 0 & 1. & 0 \\ 0 & 0 & 0 & 1. \\ 2. & 0 & 0 & 0 \\ 1.002 & 2. & 0 & 0 \\ 0 & 1.002 & 2. & 0 \\ 0 & 0 & 1.002 & 2. \\ 0 & 0 & 0 & 1.002 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 1.510 \\ 3.998 \\ -2. \\ 1. \\ 3.02 \\ 9.510 \\ 0 \\ 0 \\ 1. \end{bmatrix}.$$

Usually, an overdetermined linear system can be solved using the method of least squares. But for this example, it is easy to get candidate coefficients a_i by solving the upper 4×4 submatrix:

$$H \approx x + 1.510 + 3.998y - 2y^2 + 1.0y^3.$$

For given $\epsilon = 10^{-3}$, the backward error is:

$$\eta = \frac{\|F - G \cdot H\|}{10^{-3}} = 6.659.$$

Since η is not big, we can simply accept them as candidate factors or try to improve them by solving a linearized minimization problem as in [16]:

$$\min_{\Delta G, \Delta H} \|F - GH - G\Delta H - \Delta GH\|. \tag{9}$$

Solving it using linear programming, we obtain:

$$\begin{aligned} \Delta G &= -0.0003385 - 0.002454y; \\ \Delta H &= 0.00004460 + 0.002861y + 0.001910y^2 - 0.001761y^3. \end{aligned}$$

The backward error is

$$\frac{\|F - (G + \Delta G) \cdot (H + \Delta H)\|}{10^{-3}} = 0.22.$$

3 Comprehensive Environment Construction

From the analysis of the algorithm for computing GCD of polynomials with floating-point coefficients, it is clear that we need powerful tools to handle several numerical linear algebraic problems such as solving linear systems, least squares and minimization. Maple V provides all these facilities but, as we will see from the following tables, it is much more efficient if we make use of Matlab or C programs.

3.1 Test Results

Table 1: Sylvester Matrix QR(on DEC Alpha)

g_s	h_t	Maple QR (s)	Matlab QR (s)	C routine (s)
$s = 72$	$t = 58$	7.32	0.0888	0.0234
$s = 36$	$t = 18$	0.576	0.0039	0.00097
$s = 41$	$t = 35$	1.53	0.0088	0.0030
$s = 52$	$t = 43$	2.93	0.0146	0.0059
$s = 66$	$t = 40$	3.998	0.0205	0.0068
$s = 88$	$t = 45$	7.749	0.040	0.0137
$s = 124$	$t = 101$	37.004	0.1688	0.0810
$s = 168$	$t = 110$	72.295	0.3156	0.1318
$s = 192$	$t = 135$	119.783	0.5067	0.2362
$s = 206$	$t = 153$	160.635	0.6507	0.3205

Table 2: Overdetermined linear system solving(on DEC Alpha)

Matrix $A(m \times n)$	Maple (second)	Matlab (second)
9×4	0.027	0.00097
35×28	3.206	0.0029
56×42	11.747	0.0059
65×43	13.520	0.0068
30×20	16.653	0.0098
82×45	1.073	0.0020
115×90	157.683	0.0348
98×64	49.329	0.0176
154×101	398.048	0.0507
143×112	519.882	0.0566

Table 3: Linearized minimization solving(on DEC Alpha)

Matrix $A(m \times n)$	Maple (second)	Matlab (second)
9×4	1.259	0.1817
30×20	72.671	0.5251
35×28	114.254	0.6958
43×31	384.704	0.9533
56×42	1207.040	2.1803
82×45	1794.680	4.8994
98×64	4471.431	9.1208
116×90	18627.066	15.8375
143×112	51925.986	29.2890
154×101	60720.916	34.2238

So now our problem is how to combine programs in Maple, Matlab and C. There is an easy-to-use interface between Maple V Release 5 and Matlab. Matlab includes a Maple kernel to do symbolic processing, and provides a top-level Matlab command (`maple()`) to execute Maple function calls. On the other hand, Matlab library can also be invoked within Maple by entering the command `with(Matlab)`, which let users access all Matlab package function freely, and can also invoke an individual function using the long form `Matlab[function]`. Since most computations in our algorithm are done in symbolic way, we implement the package in Maple and call some functions from Matlab within Maple. For the C program, since Maple V Release 5 [9] does not provide a direct way to call C in Maple, we link the C program to Matlab and call it as built-in function of Matlab.

3.2 C Routines Integration

Our implementation creates MEX-files [10], which are dynamically linked subroutines that the Matlab interpreter treats like Matlab's own built-in functions. Thus one can call MEX-files exactly as calling its common function.

The source code for a MEX-file consists of two distinct parts:

1. A C computational routine containing the code for performing the computations that one wants to implement in the MEX-file.

2. A gateway routine that interfaces the computational routine with Matlab by the entry point `mexFunction` and its parameters `prhs, nrhs, plhs, nlhs`, where `prhs` is an array of right-hand input arguments, `nrhs` is the number of right-hand input arguments, `plhs` is an array of left-hand output arguments, and `nlhs` is the number of left-hand output arguments, The gateway calls the computational routine as subroutine.

It is necessary to point out that Matlab works with only a single object type: the Matlab array. In C programming, it is defined as a structure named `mxArray`. All Matlab variables, including scalars, vectors, matrices, strings and cell arrays are stored as `mxArray` type format. Parameters `prhs[]` and `plhs[]` in gateway routine `mexFunction` are pointers to `mxArray`, so variables can be passed between Matlab and C computational routines by calling library functions such as `mxGetPr(prhs[])`.

In our computation, the C routine named `qr_sylvester.c` contains both the computational routine `qr_sylvester()` (QR decomposition for Sylvester matrix [12]) and the gateway routine `mexFunction()`. It requires 2 parameters as input(`nrhs=2`), and 2 parameters as its computing output(`nlhs=2`). After compiling, the routine can be executed directly within Matlab just like its build-in function with the command form as:

```
[q,r]=qr_sylvester(v1,v2)
```

In this command, the parameters `v1,v2` are declared as Matlab vectors, corresponding to the coefficients of two given polynomials, and `q,r` are its output in Matlab data format, showing the result of QR decomposition.

3.3 Maple/Matlab/C combination

Now it is clear that our comprehensive system for approximate GCD computation of multivariate polynomials can be described as:

- Use Maple to perform symbolic computation;

- Invoke Matlab functions from within Maple to perform numerical computations such as linear programming and least squares.

- Merge C programming application routines `qr_sylvester.c`, acting as the Sylvester matrix QR solver, into Matlab being as a plug-in function, so as to be available from Maple.

4 Conclusion

In this paper, we briefly discuss using Hensel lifting algorithm to compute approximate GCD of multivariate polynomials in $\mathbb{C}$. A comprehensive environment has been constructed by Maple/Matlab/C combination for a more efficient algorithm. The method to combine Maple and Matlab with C routines provides a powerful approach for complicated computation. The work presented here indicates how to develop more powerful problem solving environments(PSE) [1] in the future.

References

[1] Elias N. Houstis and John R. Rice.(2000). On the Future of Problem Solving Environments. http://www.cs.purdue.edu/people/jrr.

[2] Corless, R. M., Gianni, P. M., Trager, G. M. and Watt, S. M.: The singular value decomposition for polynomial systems, *Proc. ISSAC '95*, ACM Press, New York, 1995, 195-207.

[3] Geddes, K.O., Czapor, S.R. and Labahn, G.: Algorithms for Computer Algebra, Boston, Kluwer, 1992.

[4] Huang, Y., Stetter, H. J., Wu, W. and Zhi, L. H.: Pseudofactors of multivariate polynomials, accepted by ISSAC'00.

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

[6] Chin, P., Corless, R. M. and Corliss, G. F.: Optimization strategies for approximate GCD problem, *Proc. ISSAC'98,* ACM Press, New York, 1998, 228-235.

[7] Gene H. Golub, Charles F. Van Loan: Matrix Computations, Second Edition. The John Hopkins University Press, 1989.

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

[9] K. M. Heal, M. L. Hansen, K. M. Rockard.: Maple V Programming Guide. Waterloo Maple Inc.

[10] Matlab Application Program Interface Guide, The MathWorks, Inc.

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

[12] Press, W., Flannery, B., Teukolsky, S. and Vetterling, W.: Numerical Recipes: The Art of Scientific Computation, Cambridge U. Press, Cambridge, 1990.

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

[14] Emiris, I.Z., Galligo, A. and Lombaradi, H.: Certified approximate univariate GCDs, *J. Pure and Applied Algebra*, **117** (1997), 229-251.

[15] Karmarkar, N. and Lakshman Y.N.: Approximate polynomial greatest common divisors and nearest singular polynomials, *Proc. ISSAC '96*, ACM Press, New York, 1996, 35-39.

[16] L.H.Zhi and M.-T. Noda: Approximate GCD of Multivariate Polynomials. Submitted to ASCM '00.