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
http://www.hpc.cs.ehime-u.ac.jp

August 12, 2005

Abstract

The problem of solving approximate GCD of multivariate polynomials has been well studied in the computational literature because of its importance particularly in engineering application[2, 6, 7]. Numbers of algorithms have been proposed to give the approach such as approximate subresultant PRS and modular algorithm using SVD. Here we focus on EZ-GCD[3], another method based on Hensel Lifting. In this computation, QR decomposition of Sylvester Matrix is the key operation. Generally, Computer Algebra Systems such as Maple[9] and Asir/Risa[12] are only applicable in small sized matrix problem. But in multivariate polynomial case, matrix size becomes very large. Obviously it could be more effective if numeric method is adopted. So we must address it in symbolic-numeric combined computation. Moreover, noticing the specificity of Sylvester Matrix data construction[8], more efficient method could be applied if we invoke a C routine acting as Sylvester Matrix QR solver. Hence it is clear that a comprehensive toolkit which can offer faster and more accurate solution on this term is needed. In this paper, Maple, a computer algebra system; Matlab[10], a high-performance numeric library; and C routines, are combined to implement our computation, showing a stable and faster result.

1 Introduction

Algorithms for approximate GCD computation have been significant advances in the past decade. Many papers have mentioned their proposal on field of univariate or multivariate polynomials. The problem we consider here is: Given two multivariate polynomials $F(x_1, ..., x_n)$ and $G(x_1, ..., x_n)$ with floating-point coefficients, how to compute the 'satisfactory' approximate GCD for a given error tolerance ϵ?

$$F = C(x_1, ..., x_n)\widetilde{F}(x_1, ..., x_n) + \mathsf{O}(\epsilon(x_1, ..., x_n))$$
$$G = C(x_1, ..., x_n)\widetilde{G}(x_1, ..., x_n) + \mathsf{O}(\epsilon(x_1, ..., x_n))$$

for example:

$$\begin{aligned} F &= (x^2 + y^2 - 1)(xy - 0.25) - 10^{-5}xy, \\ G &= (x^2 + y^2 - 1)(x - y) - 10^{-5}(x + 1) \end{aligned}$$

For $\epsilon = 10^{-5}$, the approx-GCD may be

$$x^2 + y^2 - 1$$

Of course, for such multivariate problem like this, we can consider our solution based on univariate GCD approach. But it is nontrivial to modify the techniques used in symbolic computation such as interpolation and Hensel lifting to compute the approximate GCD of multivariate polynomials. There are three main methods for this domain: Subresultant PRS, Modular Algorithm and EZ-GCD using Hensel lifting algorithm[3]. In this paper, we discuss our solution with EZ-GCD, and focus on extending the Hensel lifting to polynomials with floating-point coefficients. This computation involves the QR-decomposition of a Sylvester matrix as the key operation. Considering its special data structure, we invoke a C routine acting as an efficient Sylvester matrix QR solver. Moreover, a linear local optimization problem is also proposed to improve the candidate approximate factors obtained form Hensel lifting. Hence it is clear that a comprehensive environment which provides fast and stable symbolic and numeric computation is needed. Here we combine Maple/Matlab/C routine to meet our requirement. Showing a satisfied result.

2 Related Works

As we have just mentioned, Subresultant PRS and Modular Algorithm are also available to multivariate approximate GCD problems.

The Subresultant PRS algorithm is the Euclidean algorithm using pseudo-division in a polynomial ring.

$$\{P_1 = F, P_2 = G, \ldots, P_k \neq 0, P_{k+1} = 0.\}$$

Where

$$\beta_i P_{i+1} = \text{remainder}(\alpha_i P_{i-1}, P_i), i = 2, 3, \ldots.$$

here,

$$\begin{aligned} \alpha_i &= \text{lc}(P_i)^{d_i+1}, \quad d_i = \deg(P_{i-1}) - \deg(P_i); \\ \beta_2 &= 1, \quad \gamma_2 = 1; \\ \beta_i &= \text{lc}(P_{i-1}), \quad \gamma_i = \text{lc}(P_{i-1})^{d_i-1} \gamma_{i-1}^{1-d_{i-1}}, i \geq 3. \end{aligned}$$

lc(P) denotes the leading coefficient of P. The above method has been extended to polynomials with floating-point coefficients by Prof.Noda and Sasaki[6].

In Modular algorithm, multivariate polynomial GCD problems are reduced to univariate problems by using evaluation homomorphisms to eliminate variables, and reconstruct to GCD of the original inputs from these "images" using interpolations. There are two kinds of modular methods: One is dense and the other is sparse. For the dense modular algorithm, the number of homomorphisms is exponential in the number of variables, It is usually very large. If the multivariate polynomials and their nontrivial GCD are sparse, then sparse modular method needs much less number of homomorphisms. It computes the degree and sparsity of the GCD by random evaluation homomorphisms and only interpolates nonzero coefficients. Corless et al.[2] modified the sparse modular algorithm to compute approximate GCD in the case of bivariate polynomials.

1. Choose random evaluations α, β_i of x and y to compute T_x nonzero terms in $GCD(F(x, \beta_i), G(x, \beta_i))$ and T_y nonzero term in $GCD(F(\alpha, y), G(\alpha, y))$.

2. Solve M monic $GCD(F(x, \beta_i), G(x, \beta_i)$ for random choose β_i, where $M \geq T_y T_x/(T_x - 1)$.

3. Interpolate all coefficients simultaneously.

Both of the two methods have advantages and disadvantages[5]. For example: sparse modular method can find the "small" GCD very fast but may be unlucky or need a large number of evaluation homomorphisms in the case the GCD is dense; Subresultant algorithm can deal with the polynomials with complicated leading coefficients but it is very inefficient to find that two polynomials are actually prime to each other.

3 Hensel Algorithm

This section briefly describes the EZ-GCD using Hensel algorithm to compute approximate GCD of multivariate polynomials.

When we apply Hensel algorithm for computing GCD, the two input polynomials are reduced to two univariate polynomials whose GCD is then lifted back to the multivariate domain using a generalized Newton's iteration.

$$\begin{array}{ccccc} \mathsf{R}[x_1,\ldots,x_n] & \times & \mathsf{R}[x_1,\ldots,x_n] & \overset{\text{GCD}}{\rightarrow} & \mathsf{R}[x_1,\ldots,x_n] \\ \text{mod I} & \downarrow & & & \downarrow \text{ mod I} \\ \mathsf{R}[x_1] & \times & \mathsf{R}[x_1] & \overset{\text{GCD}}{\rightarrow} & \mathsf{R}[x_1] \end{array}$$

Where $I = (x_2 - a_2, \ldots, x_n - a_n)$.

For polynomials with exact coefficients, the main steps compute GCD using Hensel lifting can be illustrated as:

1. Choose a main variable, suppose x_1, find lucky evaluation homomorphism $I = (x_2 - a_2, \ldots, x_n - a_n)$.
2. $F_I = F \mod I, G_I = G \mod I$. Compute C_I =GCD(F_I, G_I) and cofactors $\widetilde{F}_I, \widetilde{G}_I$.
3. If there is one of the cofactors which is prime to C_I, then use multivariate Hensel construction to lift C_I and the cofactor. Otherwise perform a square-free decomposition for either F or G.

Among the steps of above, the last one is critical. Suppose P is a polynomial in $x_1, \ldots, x_n$ with leading coefficient $p_m(a_2, \ldots, a_n) \neq 0$. Let $x = x_1, \mathsf{u} = x_2, \ldots, x_n, G^{(0)}$ and $H^{(0)}$ be relatively prime polynomials satisfying

$$P(x, \mathsf{u}) = G^0(x)H^{(0)}(x). \tag{1}$$

The multivariate Hensel construction is to calculate polynomials $G^{(k)}(x, \mathsf{u})$ and $H^{(k)}(x, \mathsf{u})$ satisfying

$$P(x, \mathsf{u}) = G^k(x)H^{(k)}(x) \mod I^{k+1}. \tag{2}$$

Now we face the challenge of solving polynomial Diophantine equations for the fixed polynomials $G^{(0)}(x)$ and $H^{(0)}(x)$. If $G^{(0)}(x)$ and $H^{())}(x)$ are relatively prime, then for any polynomial $R(x)$ with deg(R(x)) $<$ deg(G$^{(0)}$(x)) + deg(H$^{(0)}$(x)), there exist unique polynomials $A(x), B(x)$ such that

$$A(x) \cdot G^{(0)}(x) + B(x) \cdot H^{(0)}(x) = R(x)$$

and

$$\deg(\mathrm{A(x)}) < \deg(\mathrm{H^{(0)}(x)}), \quad \deg(\mathrm{B(x)}) < \deg(\mathrm{G^{(0)}(x)}).$$

Suppose

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

We are going to solve the linear equations:

$$M\ x = r, \tag{3}$$

where r is the coefficient vector of polynomial $R(x)$, M is Sylvester matrix of polynomial $G^{(0)}(x)$ and $H^{(0)}(x)$, i.e.,

$$\begin{bmatrix} g_s & g_{s-1} & & \cdots & \cdots & 0 \\ & \ddots & \ddots & & & \vdots \\ & & g_s & g_{s-1} & \cdots & g_0 \\ h_t & h_{t-1} & & \cdots & \cdots & 0 \\ & \ddots & \ddots & & & \vdots \\ & & h_t & h_{t-1} & \cdots & h_0 \end{bmatrix}^T \tag{4}$$

Several ways can be applied to solve the linear equations(3). LU decomposition which is the fastest but may be unstable in some cases, especially when M is close to singular; SVD method can detect nearly rank-deficiency in the presence of roundoff error and give satisfactory results while LU decomposition fails, but it is notoriously computational intensive; QR decomposition with Householder transformations or Given rotations are very stable although it is about double cost than LU decomposition. We prefer QR decomposition as it is easy to exploit the structure of the Sylvester matrix M for the purpose of finding a computationally efficient and stable algorithm.

In general case, we can use Maple command or invoke Matlab's build-in command `qr()` to calculate the decomposition. But in case of Sylvester matrix, we find it would be more efficient if we apply a C routine[11] which is a Sylvester matrix QR solver. The reason is simple: the specificity of Sylvester matrix determines a high-speed algorithm to update its computation.

Notice that Sylvester matrix is composed of two submatrice G and H, and each submatrix is a special one being called Toeplitz whose entries are constant along each diagonal[8]. So, each time when we zero an element selectively applying Given rotations algorithm, we can change other following rows directly without necessary of calculating it again. The step can be going

on and on and finally we apply Householder reflection to zero remaining part as a submatrix.

For example, suppose we have a Sylvester matrix that $s = t = 3$, the matrix can be written as:

$$\begin{bmatrix} g_3 & g_2 & g_1 & g_0 & 0 & 0 \\ 0 & g_3 & g_2 & g_1 & g_0 & 0 \\ 0 & 0 & g_3 & g_2 & g_1 & g_0 \\ h_3 & h_2 & h_1 & h_0 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & h_0 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 & h_0 \end{bmatrix}$$

1. Apply Given rotation[8] to row 1 and 4 to zero H_3, rewrite other corresponding rows(row 2,3 from row 1, and row 5,6 from row 4) accordingly.

$$\begin{bmatrix} g_3^{(1)} & g_2^{(1)} & g_1^{(1)} & g_0^{(1)} & 0 & 0 \\ 0 & g_3^{(1)} & g_2^{(1)} & g_1^{(1)} & g_0^{(1)} & 0 \\ 0 & 0 & g_3^{(1)} & g_2^{(1)} & g_1^{(1)} & g_0^{(1)} \\ 0 & h_2^{(1)} & h_1^{(1)} & h_0^{(1)} & 0 & 0 \\ 0 & 0 & h_2^{(1)} & h_1^{(1)} & h_0^{(1)} & 0 \\ 0 & 0 & 0 & h_2^{(1)} & h_1^{(1)} & h_0^{(1)} \end{bmatrix}$$

2. Apply Given rotation to row 2 and 4, to zero element $H_2^{(1)}$, and rewrite row 3 and 5 accordingly. And so on, one more step, we get:

$$\begin{bmatrix} g_3^{(1)} & g_2^{(1)} & g_1^{(1)} & g_0^{(1)} & 0 & 0 \\ 0 & g_3^{(2)} & g_2^{(2)} & g_1^{(2)} & g_0^{(2)} & 0 \\ 0 & 0 & g_3^{(3)} & g_2^{(3)} & g_1^{(3)} & g_0^{(3)} \\ 0 & 0 & 0 & h_0^{(3)} & v^{(2)} & v^{(3)} \\ 0 & 0 & 0 & h_1^{(2)} & h_0^{(2)} & v^{(1)} \\ 0 & 0 & 0 & h_2^{(1)} & h_1^{(1)} & h_0^{(1)} \end{bmatrix}$$

3. Apply Householder reflection[8] to the low 3×3 submatrix and estimate the condition of the upper triangular factor R.

$$\begin{bmatrix} g_3^{(1)} & g_2^{(1)} & g_1^{(1)} & g_0^{(1)} & 0 & 0 \\ 0 & g_3^{(2)} & g_2^{(2)} & g_1^{(2)} & g_0^{(2)} & 0 \\ 0 & 0 & g_3^{(3)} & g_2^{(3)} & g_1^{(3)} & g_0^{(3)} \\ 0 & 0 & 0 & v_1 & v_2 & v_3 \\ 0 & 0 & 0 & 0 & v_4 & v_5 \\ 0 & 0 & 0 & 0 & 0 & v_6 \end{bmatrix}$$

The efficiency of combining Given rotations with Householder transformations is clear. For example, in case of $s = t = n$, the flops used in general LU, QR and SVD decomposition are $\frac{2}{3}(2n)^3$, $\frac{4}{3}(n^3) + 6n^2$, and $12(2n)^3$.

Our C routine, `qr_sylvester.c` runs well in the computation. Although one can use Matlab or Maple build-in function to get the result, but our experience have verified the optimization of using plug-in C function as a Sylvester matrix QR solver.

Running time (seconds) for 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
$s = 58$	$t = 35$	3.00	0.0156	0.0049
$s = 36$	$t = 34$	1.29	0.0058	0.0020
$s = 36$	$t = 18$	0.57	0.0039	0.00096

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 star the Hensel lifting. If M is near singular, we have to choose other evaluation points or try the squarefree decomposition of polynomials F or G.

For Hensel construction, it is also very important to decide when to stop the lifting. Obviously, the procedure will stop as soon as $\|\Delta P^{(k)}\| = \mathsf{O}(\varepsilon)$. However, it is also possible that $\Delta P^{(k)}$ still has some coefficients whose absolute values are not $\mathsf{O}(\varepsilon)$ when k is larger than the total degree of the variables u in P. For example, the given polynomial:

$$P = (x + 2 + y)(s + 1.51 + 4y - 2y^2 + y^3) + \eta(x + y)$$

when $\eta = 0$:

$$\begin{aligned}
P &\equiv (x+2)(x+1.51) \mod y \\
P &\equiv (x+2+y)(x+1.51+4y) \mod y^2 \\
P &\equiv (x+2+y)(x+1.51+4y-2y^2) \mod y^3 \\
P &\equiv (x+2+y)(x+1.51+4y-2y^2+y^3) \mod y^4 \\
P &\equiv (x+2+y)(x+1.51+4y-2y^2+y^3) \mod y^5
\end{aligned}$$

The significant character make it easy to determine when to stop the lifting because the coefficients will not change anymore after certain lifting steps.

However, in approximate case, we are not so lucky:
when $\eta = \frac{1}{10^4}$(rounding to 4 digits):

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

In this case, we have to decide if it is caused by error accumulation or P has no approximate factors indeed. If many coefficients of $\Delta P^{(k)}$ are relatively much larger than ε, it is believed that F and G have no approximate GCD and no correction is necessary. Otherwise, we can use the optimization method to improve the initial approximate univariate GCD and it cofactor $G^{(0)}$ and $H^{(0)}$, or increase the number of digits carried in floating-point computations. But it is possible that the above two techniques are very inefficient. A correction method on this issue was discussed by Lihong Zhi and her colleagues[4].

When $\Delta P^{(0)}$ is reasonable small, we round off $G^{(k-1)}$ and $H^{(k-1)}$ to G and H respectively. G and H are supposed to be candidate factors of P.
Refer to the same example of above, when $\eta = \frac{1}{10^4}$, we suppose

$$\begin{aligned}
G &= x+2.+1.002y, \\
H &= x+a_1+a_2y+a_3y^3+a_4y^3.
\end{aligned}$$

Thinking about the sequence $[x, xy, xy^2, xy^3, 1, y, y^2, y^3, y^4]$, we get the linear equations:

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

so we get:

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

The backward error $\|P - G \cdot H\| = 0.006659$.
For given $\epsilon = 10^{-3}$,

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

Clearly it is not very large, thus we come to the conclusion that G and H are possibly valid factors of P with the necessary of correction.

Now we are facing the challenge of linearized minimization problem:

$$\min_{\Delta G, \Delta H} \|P - GH - G\Delta H - \Delta GH\|. \tag{5}$$

Notice that it is not necessary to actually solve the minimization problem. We are only interested in finding $\Delta H, \Delta G$ which modify H and G to achieve a sufficient small backward error. Here

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

So we can see that:

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

it is quite satisfying.

On the other hand, notice that the minimization problem is essentially linear programming solving

$$\min \|A^T x - b\|_\infty$$

$$\min\{y : A^T - ye \leq b, \quad A^T x + ye \geq b, y \in R\}$$

Even though we can apply complex search method or primal-dual interior-point method to solve it, but it is necessary to point out that the matrix A is usually very big. For example, Let $x = (x_1, x_2, x_3), deg(P) = 6, deg(G) = 2, deg(H) = 4$, then A is an 84×45 matrix. In this case, numeric computation package such as Matlab is necessary for a more efficient calculation.

4 Comprehensive Environment Construction

The fact that we could not find one stand-alone mathematical package which simultaneously provided high performance for both symbolic processing and numerical computation impels us continuously to think about a useful comprehensive environment on this domain. Fortunately we already have relatively sophisticated packages are provided separately which are standing on their own field, such as Maple to perform symbolic computations and Matlab to perform linear programming solving. We also have C for programming applications such as Sylvester matrix QR solver. That's why we are focusing on the combination of these three part to work together. Another reason we selected this particular combination is that Maple and Matlab already provide an easy-to-use interface between the two system.

4.1 Two methods

The simplest way to enable both Maple and Matlab to be available in solving one problem is to transfer the data from one format to the other. Each system has its own inter form for data representation which is not usable directly to the other. However, both of them have provided an input-output interface for data transfer. Due to this, we wrote a routine in C to perform this manipulation. But the off-line mode solution is of inconvenient and not satisfactory–people should do some job within one system such as Maple, save the data, run the C routine to perform the transfer, and then call the other software such as Matlab as a UNIX external program, read in the transfered data to continue the mission, and vice versa. Clearly, the more direct and more efficient implementation is needed.

From Maple V Release 5 on, an easy-to-use interface between Maple and Matlab has been provided. Matlab currently includes a Maple kernel to do symbolic processing, and also provides a top-level Matlab command (`maple()`) to execute Maple function calls. On the other hand, Matlab library can also be invoked successfully within Maple system 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]`.

4.2 C Routines Integration

Our implementation has been done via creating MEX-fines that are dynamically linked subroutines that the Matlab interpreter can automatically behave

just like Matlab's own built-in functions, thus one can call MEX-files exactly as calling its built-in function.

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

1. A C computational routine that contains the code for performing the computations that one wants implemented 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` which contains both computational routine(QR decomposition for Sylvester matrix) `qr_sylvester()` and 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 the command, parameters `v1,v2` are declared 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.

4.3 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 function from within Maple system to perform number computation such as linear programming solving, by using `with(Matlab)` and other corresponding command;

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

5 Conclusion

In this paper, we briefly discuss using Hensel lifting algorithm to solve approximate GCD of multivariate polynomials. A comprehensive environment has been implemented by Maple/Matlab/C combination for a more efficient computation. As both Maple and Matlab are very popular system to be used in its corresponding field, the method to combine them with C routines shows a powerful approach for complicated problem solving.

PSE(Problem Solving Environments)[1] is now a well established methodology for computational science, and therefore we have reason to predicate a more powerful and standardized PSE-building architecture should be available for robust, flexible, effective PSEs construction.

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, submitted to ISSAC'00.

[5] Liao, H. C. and Fateman, R. J.: Evaluation of the heuristic polynomial GCD, *Proc. ISSAC '95,* ACM Press, New York, 1995,240-247.

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

[7] 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.

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

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

[12] Noro, M. A Computer Algebra System Risa/Asir, 2000. ftp://archives.cs.ehime-u.ac.jp/put/asir2000/