Cancellation Errors in Multivariate Resultant Computation with Floating-point Numbers *

Tateaki Sasaki †) and Tomoyuki Sato ‡)

†) Institute of Mathematics and ‡) Master's Program in Education
University of Tsukuba
Tsukuba-shi, Ibaraki 305, Japan
{sasaki,tsato}@math.tsukuba.ac.jp

Abstract

We test five methods of computing resultant of multivariate polynomials with floating-point number coefficients, mostly from the viewpoint of numerical errors. The methods tested are a) Brown-Traub's subresultant PRS algorithm, b) Collins' algorithm based on polynomial interpolation, c) minor expansion method of Bezout's determinant, d) Gaussian elimination method (Bareiss' algorithm) of Bezout's determinant, and e) efficient Gaussian elimination method (Sasaki-Murao's algorithm) of Bezout's determinant. We find that method a) causes so large cancellation errors that it is almost useless practically. The errors caused by methods b) and d) are not so large as those by a) but still large. The methods c) and e) are quite safe against the cancellation error and they are efficient, too. Mechanisms of occurrence of large cancellation errors in methods a), b) and d) are clarified. By these, we show that approximate algebraic computation often causes large cancellation errors hence the error analysis and development of error-safe algorithms are very important.

*Work supported in part by Japanese Ministry of Education, Science and Culture under Grants 09308008.
Key words: approximate algebra, approximate algebraic computation, cancellation errors, error analysis, multivariate polynomial resultant, polynomial resultant

1 Introduction

As for computation of resultant of multivariate polynomials, we have now four major methods, 1) calculating Sylvester's determinant, 2) calculating Bezout's determinant, 3) calculating polynomial remainder sequence, or PRS in short, by Brown-Traub's subresultant PRS algorithm [BT71] (we call this method *subresultant method* in this paper), and 4) *Collins' algorithm* based on polynomial interpolation [Col71]. The methods 1) and 2) are essentially the same, hence we consider only 2) because Bezout's determinant is faster to evaluate than Sylvester's. There are several algorithms to calculate symbolic determinants, *minor expansion method*, *Gaussian elimination method* (Bareiss' algorithm [Bar68]), and *efficient Gaussian elimination method* (Sasaki-Murao's algorithm [SM82]).

So far, researchers were interested mostly in the polynomials with exact coefficients, such as rational numbers or algebraic numbers. In this paper, we consider polynomials with floating-point number coefficients and investigate the numerical errors caused by the algorithms. There are two kinds of numerical errors of floating-point numbers, round-off error and cancellation error. The latter is due to the cancellation of almost same numbers. The cancellation error is very dangerous because $5 \sim 10$ digits may be lost by a single cancellation, and the computed result may often become very inaccurate if cancellation happens. Although the round-off error cannot be negligible, we focus our attention on the cancellation error in this paper.

In 1969, Ku and Adler [KA69] tested two methods of resultant computation, the subresultant method (precisely, they tested the reduced PRS algorithm [Col67]) and the minor expansion method, and found that the minor expansion method is quite safe against numerical errors while the subresultant method causes large cancellation errors. Ku and Adler concluded that large errors in the subresultant method were caused by polynomial division. This is correct but their reasoning is too naive. In this paper, we test five methods of resultant computation, from the viewpoint of not only numerical errors but also computation time. The methods tested are a) the subresultant method, b) Collins' algorithm, c) minor expansion method of Bezout's determinant, d) Gaussian elimination method of Bezout's determinant, and e) efficient Gaussian elimination method of Bezout's determinant. Then, we clarify the mechanisms of occurrence of large errors. We also emphasize that the univariate polynomial division by a divisor of small leading coefficient causes large errors.

In **2**, we explain necessary algorithms briefly. In **3**, we show the results of empirical tests of five methods of resultant computation for polynomials in $2 \sim 4$ variables. In **4**, we clarify why the subresultant method, Collins' algorithm and the Gaussian elimination method cause large cancellation errors.

2 Brief explanation of algorithms

In order to make this paper self-contained and for the sake of later analysis, we briefly explain Brown-Traub's subresultant PRS algorithm, Collins' algorithm, Bareiss' algorithm, and Sasaki-Murao's algorithm.

By $\deg(P)$ and $\mathrm{lc}(P)$, with P a polynomial in $\mathbf{C}[x, y, \ldots, z]$, we denote respectively the degree and the leading coefficient of P, w.r.t. variable x.

2.1 Brown-Traub's algorithm (the subresultant method)

Let F and G be polynomials in $\mathbf{C}[x, y, \ldots, z]$ and represented as

$$\begin{cases} F = f_m x^m + f_{m-1}x^{m-1} + \cdots + f_0, \\ G = g_n x^n + g_{n-1}x^{n-1} + \cdots + g_0, \quad n \le m. \end{cases} \tag{2.1}$$

Consider calculating polynomial remainder sequence

$$(P_1 = F,\ P_2 = G,\ P_3,\ \cdots, P_i,\ \cdots), \quad \deg(P_i) < \deg(P_{i-1}) \quad (i = 3, 4, \ldots) \tag{2.2}$$

by the formula

$$\beta_i P_{i+1} = \alpha_i P_{i-1} - Q_i P_i, \quad \alpha_i, \beta_i \in \mathbf{C}[y, \ldots, z]. \tag{2.3}$$

We usually set α_i as

$$\alpha_i = \mathrm{lc}(P_i)^{\deg(P_{i-1}) - \deg(P_i) + 1}. \tag{2.4}$$

The elements P_i $(i = 3, 4, \ldots)$ can be expressed as

$$P_i(x, y, \ldots, z) = \lambda_i(y, \ldots, z) S_j(x, y, \ldots, z), \quad \deg(S_j) = j = \deg(P_i), \tag{2.5}$$

where S_j is called the jth subresultant and represented by determinant as

$$S_j = \begin{vmatrix} f_m & f_{m-1} & \cdots & \cdots & f_{2j+2-n} & x^{n-j-1}F \\ & \ddots & \ddots & \ddots & \vdots & \vdots \\ & & f_m & \cdots & f_{j+1} & x^0 F \\ g_n & g_{n-1} & \cdots & \cdots & g_{2j+2-m} & x^{m-j-1}G \\ & \ddots & \ddots & \ddots & \vdots & \vdots \\ & & g_n & \cdots & g_{j+1} & x^0 G \end{vmatrix}. \tag{2.6}$$

Note that S_0 is nothing but Sylvester's determinant.

The PRS such that $P_i = S_j$ for any i is called subresultant PRS. Collins [Col67] found an algorithm with which we have $P_i = S_j$ for "normal" sequence, and Brown and Traub [BT71] devised an algorithm with which we have $P_i = S_j$ for any case. In Brown-Traub's algorithm, β_i is set as follows.

$$\beta_2 = 1, \quad \beta_i = c_{i-1}\zeta_i^{d_{i-1}} \quad (i = 3, 4, \ldots), \tag{2.7}$$

where $c_i = \mathrm{lc}(P_i)$, $d_i = \deg(P_i) - \deg(P_{i+1})$ and ζ_i is calculated as follows.

$$\zeta_3 = c_2^{d_1}, \quad \zeta_i = c_{i-1}^{d_{i-2}} \zeta_{i-1}^{(1-d_{i-2})} \quad (i = 4, 5, \ldots). \tag{2.8}$$

2.2 Collins' algorithm

Computation of subresultant PRS is considerably time-consuming: $\tilde{P}_{i+1} \overset{\text{def}}{=} \alpha_i P_{i-1} - Q_i P_i$ is usually of much larger size than P_{i+1} when input polynomials are multivariate, hence we must compute polynomials of much larger sizes than the resultant itself.

Collins [Col71] proposed to interpolate resultant$_x(F,G)$ by calculating

$$\text{resultant}_x(F(x,y_i,\ldots,z_i),G(x,y_i,\ldots,z_i)), \quad i=1,2,\ldots,N, \tag{2.9}$$

where $y_i,\ldots,z_i$ are suitably chosen numbers such that $f_m(y_i,\ldots,z_i)\neq 0$, $g_n(y_i,\ldots,z_i)\neq 0$ and N is a positive integer. The interpolation can be done by either Lagrange's formula or Newton's formula, see [Lip71] for example. The determinant representation of resultant, Sylvester's or Bezout's, allows us to know upper bounds of degrees of resultant w.r.t. $y,\ldots,z$:

$$\deg_y(\text{resultant}) \le N_y, \ \cdots, \deg_z(\text{resultant}) \le N_z. \tag{2.10}$$

Then, we have to choose N_y+1, ..., N_z+1 different values for $y,\ldots,z$, respectively, hence $N=(N_y+1)\times\cdots\times(N_z+1)$.

With computational-time analysis, Collins insists that his algorithm is much superior to the subresultant method or calculating Bezout's determinant.

2.3 Bareiss' algorithm (Gaussian elimination)

Let M be an $n\times n$ matrix of elements $a_{i,j}$, where $1\le i\le n$ and $1\le j\le n$, and let $M^{(k)}$, with $0\le k\le n-1$, be an $(n-k)\times(n-k)$ matrix of elements $M^{(k)}_{i,j}$, where we set i and j as $k+1\le i\le n$ and $k+1\le j\le n$ for convenience:

$$\begin{aligned} M &= (a_{i,j}), \qquad 1\le i,j\le n,\\ M^{(k)} &= (M^{(k)}_{i,j}), \quad k+1\le i,j\le n. \end{aligned} \tag{2.11}$$

We put $M^{(k)}_{i,j}$ as follows.

$$M^{(0)}_{i,j}=a_{i,j}, \qquad M^{(k)}_{i,j}=\begin{vmatrix} a_{1,1} & \cdots & a_{1,k} & a_{1,j}\\ \vdots & \ddots & \vdots & \vdots\\ a_{k,1} & \cdots & a_{k,k} & a_{k,j}\\ a_{i,1} & \cdots & a_{i,k} & a_{i,j} \end{vmatrix} \quad (k=1,\ldots,n-1). \tag{2.12}$$

Then, we have the following lemma [Bar68].

Lemma 1 *For $1\le k\le n-2$ and $k+2\le i,j\le n$, we have*

$$\tilde{M}^{(k+1)}_{i,j} \stackrel{\text{def}}{=} \begin{vmatrix} M^{(k)}_{k+1,k+1} & M^{(k)}_{k+1,j}\\ M^{(k)}_{i,k+1} & M^{(k)}_{i,j} \end{vmatrix} = M^{(k-1)}_{k,k}M^{(k+1)}_{i,j}. \quad \square \tag{2.13}$$

Therefore, putting $M^{(-1)}_{0,0}=1$, $M^{(0)}=M$ and calculating $M^{(k+1)}_{i,j}$ iteratively as

$$M^{(k+1)}_{i,j}=\tilde{M}^{(k+1)}_{i,j}/M^{(k-1)}_{k,k}, \quad k=0,1,\ldots,n-2. \tag{2.14}$$

the $(n-1)$-st step (i.e., $k=n-2$) gives $M^{(n-1)}_{n,n}=D$. Thus, Bareiss obtained the following algorithm.

Algorithm Bareiss
input: $M(n \times n),\ n \geq 2$;
output: $D = |M|$;
  for $k = 0$ to $n-2$ **do begin**
    if $k = 0$ **then** $M_{0,0}^{(-1)} := 1$ **else** make pivoting so that $M_{k,k}^{(k-1)} \neq 0$;
    for $i = k+2$ to n **do**
      for $j = k+2$ to n **do**

$$M_{i,j}^{(k+1)} := (M_{k+1,k+1}^{(k)} M_{i,j}^{(k)} - M_{i,k+1}^{(k)} M_{k+1,j}^{(k)}) / M_{k,k}^{(k-1)}; \tag{2.15}$$

    end ;
  return $D := M_{n,n}^{(n-1)}$ □

2.4 Sasaki-Murao's algorithm (efficient Gaussian elimination)

The most time-consuming step in Bareiss' algorithm is Eq. (2.15): $\tilde{M}_{i,j}^{(k+1)}$ is usually of much larger size than $M_{i,j}^{(k+1)}$ when input polynomials are multivariate, hence we must compute polynomials of much larger sizes than $|M|$. On the other hand, $M_{i,j}^{(k+1)}$ is given by the quotient of $\tilde{M}_{i,j}^{(k+1)}$ and $M_{k,k}^{(k-1)}$, hence we can calculate $M_{i,j}^{(k+1)}$ by a part of the expression of $\tilde{M}_{i,j}^{(k+1)}$. Utilizing this fact, Sasaki and Murao devised the following algorithm [SM82].

Let us first consider the case that the diagonal elements of M are variables independent of other elements:

$$M \;=\; \begin{pmatrix} X_1 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & X_2 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & X_n \end{pmatrix}. \tag{2.16}$$

Then, $M_{k,k}^{(k-1)}$ can be expressed as follows.

$$\begin{aligned} & M_{k,k}^{(k-1)} = X_1 X_2 \cdots X_k + N^{(k-1)}, \\ & \text{total-degree of } N^{(k-1)} \text{ w.r.t. } X_1, \ldots, X_k \text{ is } k-2. \end{aligned} \tag{2.17}$$

Therefore, in order to calculate $M_{i,j}^{(k+1)}$, it is enough to calculate only the terms proportional to $X_1 \cdots X_k$ in $\tilde{M}_{i,j}^{(k+1)}$. In order to calculate the necessary terms only, Sasaki and Murao introduced a special multiplication operator $\otimes$. Let P and P' be polynomials in main variables $X_l, X_{l+1}, \ldots, X_n$ and be expressed as $P = P_1 X_l + P_0$ and $P' = P'_1 X_l + P'_0$, where P_0 and P'_0 are independent of X_l. Then, operation of $\otimes_l^k$ is defined recursively as

$$P \otimes_l^k P' \;=\; \begin{cases} (P_1 \otimes_{l+1}^k P'_1) X_l + (P_1 \otimes_{l+1}^k P'_0 + P_0 \otimes_{l+1}^k P'_1) & \text{if } \; l \leq k, \\ P \times P' & \text{if } \; l > k. \end{cases} \tag{2.18}$$

That is, the operator $\otimes_l^k$ takes out only the terms proportional to $X_l \cdots X_k$ and divides them by $X_l \cdots X_k$.

According to [SM82], we can calculate $M_{i,j}^{(k+1)}$ as follows. We first calculate $Q^{(k-1)} \stackrel{\text{def}}{=} \text{quotient}((X_1X_2\cdots X_k)^2, M_{k,k}^{(k-1)})$ as

$$Q^{(k-1)} := X_1X_2\cdots X_k - N^{(k-1)} + N^{(k-1)} \otimes_1^k N^{(k-1)} - N^{(k-1)} \otimes_1^k N^{(k-1)} \otimes_1^k N^{(k-1)} + \cdots. \tag{2.19}$$

Then, we calculate $M_{i,j}^{(k+1)}$ as follows.

$$M_{i,j}^{(k+1)} := \left(M_{k+1,k+1}^{(k)} \otimes_1^k M_{i,j}^{(k)} - M_{i,k+1}^{(k)} \otimes_1^k M_{k+1,j}^{(k)}\right) \otimes_1^k Q^{(k-1)}. \tag{2.20}$$

In the general case in which the diagonal elements are not independent variables, we first replace diagonal elements $a_{1,1}, \ldots, a_{n-2,n-2}$ by independent variables $X_1, \ldots, X_{n-2}$, respectively, then apply the above algorithm, and finally substitute $a_{i,i}$ for X_i $(i = 1, \ldots, n-2)$ in the resulting determinant.

3 Empirical tests

We have tested the five methods mentioned above by polynomials in $2 \sim 4$ variables and investigated the numerical errors contained in the coefficients of resultants computed.

The tests are actually done as follows. We give polynomials F_r and G_r with rational number coefficients, and convert the coefficients into double precision floating-point numbers: $F_f := \text{floatcoef}(F_r)$, $G_f := \text{floatcoef}(G_r)$. We calculate $R_r = \text{resultant}(F_r, G_r)$ and $R_f = \text{resultant}(F_f, G_f)$ by rational number arithmetic and floating-point number arithmetic, respectively. Finally, we compare each pair of the mutually corresponding coefficients of R_r and R_f, let the coefficients be c_r and c_f, and calculate the relative error $|c_f - c_r|/|c_r|$. By *maximum (minimum) relative error*, we mean the maximum (minimum) value of such relative errors. The machine used is SPARC station ELC (cpu: SPARC 33MHz), and the algebra system used is GAL [Sas91].

Below, although polynomials $\{F, G\}$ in each example are given as integer coefficients, we calculate the resultant of $F/3.0$ and $G/3.0$ or the resultant of $F/7.0$ and $G/7.0$. The reason is that some algorithms give extremely small numerical errors for $\{F, G\}$ or $\{F/2.0, G/2.0\}$, etc. and such cases are not suited for investigating the cancellation errors, while each algorithm produces cancellation errors for $\{F/3.0, G/3.0\}$ or $\{F/7.0, G/7.0\}$ as expected.

3.1 Test of Collins' algorithm

Note that there are several versions of Collins' algorithm, due to arbitrariness of the choice of interpolation points and the choice of interpolation method, and the amount of cancellation error depends on the choices largely. We first investigate this by a simple example.

Example 2.1 $R_{2,1} = \text{resultant}_x(F_{2,1}/3.0, G_{2,1}/3.0)$, where

$$\left\{\begin{array}{rcl} F_{2,1} &=& (y+1)x^4 + (2y^2-3)x^2 + (y^2+y-3)x - (y^3-2y^2+y), \\ G_{2,1} &=& (y^2+2y-1)x^3 + (y-1)^2x + (y^3-3y^2+y+5). \end{array}\right. \tag{3.1}$$

We note that $\deg(R_{2,1}) = 17$ and $0.00045 < |\text{coefficient}(R_{2,1})| < 6.17$. □

An upper bound of the degree of $R_{2,1}$, estimated by Bezout's determinant is 20, so we choose the evaluation points $(y_0, y_1, y_2, \ldots, y_{20})$ either points on the real axis as

$$(\ -10r,\ -9r,\ \cdots,\ -r,\ 0,\ +r,\ \cdots,\ +9r,\ +10r\ ),$$

or complex points on a complex square as

$$(\ -5c,\ -(4 \pm i)c,\ \cdots,\ -(1 \pm 4i)c,\ (0 \pm 5i)c,\ +(1 \pm 4i)c,\ \cdots,\ +(4 \pm i)c,\ +5c\ ),$$

where r and c are positive real numbers. Table I shows the minimum and maximum relative errors occurred in the coefficients of $R_{2,1}$ calculated by Collins' algorithm by putting $r, c = 1.0,\ 0.5,\ 0.3,\ 0.2,\ 0.1$ and adopting Lagrange's interpolation method [Lip71].

r	min-rel-error	max-rel-error
1.0	1.0×10^{-16}	3.6×10^{-07}
0.5	1.0×10^{-16}	1.1×10^{-11}
0.3	2.9×10^{-15}	2.0×10^{-06}
0.2	3.8×10^{-13}	1.3×10^{-09}
0.1	2.8×10^{-13}	3.5×10^{-06}

c	min-rel-error	max-rel-error
1.0	3.6×10^{-16}	7.1×10^{-10}
0.5	1.0×10^{-16}	8.8×10^{-14}
0.3	5.5×10^{-16}	2.9×10^{-12}
0.2	3.5×10^{-16}	8.8×10^{-11}
0.1	4.2×10^{-16}	4.7×10^{-06}

Table I : Comparison of evaluation points in Collins' method.
Left table: real evaluation points on the real axis.
Right table: complex evaluation points on a rectangle.

We see that some coefficients are quite erroneous, although some are quite accurate in each parameter setting. We observed that the lower the degree of a term is, the lower the accuracy of corresponding coefficient is.

We have also tested Newton's interpolation method [Lip71], but it gave almost nonsense results because of too large cancellation errors.

3.2 Tests on bivariate polynomials

In this subsection, we perform a simple testing of the five methods by bivariate polynomials. This simple test will reveal weak points of some methods clearly. Taking account the results of test in the previous subsection, we choose Lagrange's interpolation formula and set the evaluation points by putting $r = c = 0.5$ in Collins' algorithm.

Below, we use the following abbreviations. (See **4.3** for improved subresultant method.)

resSo	:	subresultant method (original),
resSi	:	subresultant method (improved),
resCr	:	Collins' algorithm (real evaluation points),
resCc	:	Collins' algorithm (complex evaluation points),
resM	:	minor expansion method,
resG	:	Gaussian elimination method,
resEG	:	efficient Gaussian elimination method.

Example 2.2 $R_{2,2} = \text{resultant}_x(F_{2,2}/7.0, G_{2,2}/7.0)$, where

$$\left\{ \begin{array}{rcl} F_{2,2} & = & (y+2)^3x^5 - (y-1)^3x^4 + (y+2)^3x^3 \\ & & + \; (y-2)^3x^2 - (y^2+4y-5)x - (y^3+2y^2+5), \\ G_{2,2} & = & (2y^2-1)^2x^4 + (y^3+4y^2-3)x^3 \\ & & + \; (2y^2-3)^2x^2 + (y^2+y-3)^2x - (y^3-2y^2+y). \end{array} \right. \tag{3.2}$$

We note that $\deg(R_{2,2}) = 32$ and $0.00019 < |\text{coefficient}(R_{2,2})| < 339.$. $\square$

Example 2.3 $R_{2,3} = \text{resultant}_x(F_{2,3}/3.0, G_{2,3}/3.0)$, where

$$\left\{ \begin{array}{rcl} F_{2,3} & = & (2y^2-1)^2x^6 + (2y^2-3)^2x^5 - (y^2+1)^2x^4 \\ & & + \; (y^3+y^2-1)x^2 - (2y^3-y+4)x + (y^3-3y^2+y+5), \\ G_{2,3} & = & (2y+1)^5x^6 - (y-2)^5x^5 + (2y^2+y-4)^2x^4 \\ & & - \; (y^2+2y-1)^2x^2 + (y^2-5)^2x + (y^4+3y^3-2y+7). \end{array} \right. \tag{3.3}$$

We note that $\deg(R_{2,3}) = 50$ and $0.0016 < |\text{coefficient}(R_{2,3})| < 1.4 \times 10^{12}$. $\square$

method	Ex. 2.1	time	Ex. 2.2	time	Ex. 2.3	time
resSo	1.8×10^{-04}	60	$\cdots^{(1)}$	132	$\cdots^{(5)}$	2000
resSi	1.7×10^{-11}	35	$\cdots^{(2)}$	266	$\cdots^{(6)}$	1270
resCr	1.1×10^{-11}	100	$\cdots^{(3)}$	434	$\cdots^{(7)}$	1860
resCc	8.8×10^{-14}	250	2.1×10^{-07}	1253	$\cdots^{(8)}$	5120
resM	1.1×10^{-15}	20	2.6×10^{-15}	309	3.2×10^{-15}	2010
resG	4.4×10^{-11}	33	$\cdots^{(4)}$	223	$\cdots^{(9)}$	1060
resEG	5.0×10^{-16}	26	3.8×10^{-15}	520	2.3×10^{-14}	2740

Table II : Comparison of five methods (max-rel-error & time(ms)).
" $\cdots$ " shows that some coefficients are fully erroneous.
The number of almost fully erroneous coefficients:
(1) 23, (2) 11, (3) 6, and (4) 15 among 33 coefficients,
(5) 39, (6) 36, (7) 44, (8) 30, and (9) 23 among 51 coefficients.

Table II shows the maximum relative errors and the computing times. We see that the original subresultant method causes so large errors and are completely useless unless for very small size expressions. The improved subresultant method, Collins' algorithm and the Gaussian elimination method are safer than the original subresultant method, but they are also useless for medium or large size expressions. We understand that large errors in these methods are mostly cancellation errors: the reason is that the round-off error occurs equally to every method and the errors in minor expansion method and efficient Gaussian elimination method are quite small.

3.3 Tests on polynomials in 3 and 4 variables

In this subsection, we test only c) the minor expansion method, d) Gaussian elimination method, and e) efficient Gaussian elimination method. The test polynomials are as follows.

Example 3.1 $R_{3,1} = \mathrm{resultant}_x(F_{3,1}/3.0, G_{3,1}/3.0)$, where

$$\left\{ \begin{array}{rcl} F_{3,1} & = & (y-z+1)^2x^4 + (2y^2-3z^2)x^3 \\ & & -\ (y^2-3yz)x^2 + (y^2+y-3z^2)x - (y^3-2y^2z+yz^2), \\ G_{3,1} & = & (y^2+2yz-1)x^3 + (y+z-1)^2x^2 \\ & & -\ (yz+z^2+z)x + (y^3+3y^2-yz^2+5z^3). \end{array} \right. \tag{3.4}$$

We note that #terms($R_{3,1}$) = 173 . □

Example 3.2 $R_{3,2} = \mathrm{resultant}_x(F_{3,2}/7.0, G_{3,2}/7.0)$, where

$$\left\{ \begin{array}{rcl} F_{3,2} & = & (y-z+3)^2x^6 + (2y^2-3z^2)x^5 - (y^2+3yz)x^4 \\ & & +\ (y+2x+3)^2x^2 + (yz-3y+2z)x - (y^2+z^2-5), \\ G_{3,2} & = & (y^2+2yz-3)x^4 + (y+z-5)^2x^3 - (y^2+2yz)x^2 \\ & & -\ (yz+z^2+5z)x + (2y^2-3yz^2+5z^3+8). \end{array} \right. \tag{3.5}$$

We note that #terms($R_{3,2}$) = 304 . □

Example 4.1 $R_{4,1} = \mathrm{resultant}_x(F_{4,1}/3.0, G_{4,1}/3.0)$, where

$$\left\{ \begin{array}{rcl} F_{4,1} & = & (y+z-u+1)^2x^4 + (2y^2+z^2-2u^2)x^3 \\ & & -\ (y^2-2yz+3zu)x^2 + (y^2+3z^2-2zu-2u^2)x \\ & & -\ (y^3-2y^2z+yz^2+2z^2u-3z^3+u^3), \\ G_{4,1} & = & (y^2+2yz+2zu-1)x^3 + (y+z-u-1)^2x^2 \\ & & -\ (yz+z^2-2u^2+z-u+1)x \\ & & +\ (y^3+3y^2-yz^2+2z^3-3zu^2+u^2-z+u). \end{array} \right. \tag{3.6}$$

We note that #terms($R_{4,1}$) = 1275 □

Example 4.2 $R_{4,2} = \mathrm{resultant}_x(F_{4,2}/7.0, G_{4,2}/7.0)$, where

$$\left\{ \begin{array}{rcl} F_{4,2} & = & (y-z+u+1)^2x^5 + (y^2-3z^2+2u^2)x^4 \\ & & -\ (y^2-yz+2yu-2zu)x^3 + (yz+z^2-2u^2)x^2 \\ & & -\ (3yz+2z^3-zu^2)x - (y^3-2y^2z+yz^2+2zu^2), \\ G_{4,2} & = & (y^2+2yz+yu-z^2-3)x^4 + (y-z+2u-2)^2x^3 \\ & & -\ (y^2+2yz-z^2-2zu+2z)x^2 + (yz-3y+2z-u^2)x \\ & & -\ (y^2-2yz+z^2-zu+2u^2-2z+2u-5). \end{array} \right. \tag{3.7}$$

We note that #terms($R_{4,2}$) = 2201. □

Table III shows the maximum relative errors and the computing times (in seconds). We see clearly that the minor expansion method and the efficient Gaussian elimination method are quite safe against the numerical error, even for rather large size expressions.

Example	resM	time	resG	time	resEG	time
Ex. 3.1	4.4×10^{-15}	0.32	7.3×10^{-07}	0.81	4.4×10^{-15}	0.56
Ex. 3.2	4.3×10^{-14}	6.40	$\cdots$	6.41	5.7×10^{-14}	12.70
Ex. 4.1	9.1×10^{-14}	4.01	8.8×10^{-04}	13.78	1.8×10^{-13}	9.15
Ex. 4.2	1.3×10^{-13}	21.60	$\cdots$	243.00	5.9×10^{-13}	70.00

Table III : Comparison of three methods (max-rel-error & time(sec)).
" $\cdots$ " shows that many coefficients are fully erroneous.

3.4 The case that cancellation occurs inevitably

One may think that the minor expansion method and the efficient Gaussian elimination method always give almost accurate results, but this is wrong. If the input polynomials have an approximate common factor, the computation with floating-point numbers causes some amounts of cancellation errors inevitably. We show this by two simple examples.

Example 5.1 $R_{2,4} = \mathrm{resultant}_x(F_{2,4}/3.0, G_{2,4}/3.0)$, where

$$\begin{cases} F_{2,4} &=& (x+y-1) \times G_{2,1} + F_{2,1}/1000, \\ G_{2,4} &=& (x-2y+1) \times G_{2,1}. \end{cases} \tag{3.8}$$

We note that $\deg(R_{2,4}) = 23$ and $3.6 \times 10^{-12} < |\mathrm{coefficient}(R_{2,4})| < 2.6 \times 10^{-7}$. □

Example 5.2 $R_{2,5} = \mathrm{resultant}_x(F_{2,5}/7.0, G_{2,5}/7.0)$, where

$$\begin{cases} F_{2,5} &=& (x+y-1) \times G_{2,2} + F_{2,2}/1000, \\ G_{2,5} &=& (x-2y+3) \times G_{2,2}. \end{cases} \tag{3.9}$$

We note that $\deg(R_{2,5}) = 41$ and $5.4 \times 10^{-15} < |\mathrm{coefficient}(R_{2,5})| < 3.8 \times 10^{-7}$. □

	resM	resM	resEG	resEG
Example	min-rel-error	max-rel-error	min-rel-error	max-rel-error
Ex. 5.1	4.4×10^{-13}	5.7×10^{-8}	6.1×10^{-13}	4.0×10^{-8}
Ex. 5.2	2.0×10^{-12}	4.4×10^{-9}	3.3×10^{-10}	7.7×10^{-3}

Table IV : Input polynomials have an approximate common factor.

Table IV shows the minimum and the maximum relative errors caused by the minor expansion method and the efficient Gaussian elimination method. We note that there occurred cancellation errors in all the coefficients of the output resultants, as can be seen by the minimum relative errors. We further note that the minor expansion method is better than the efficient Gaussian elimination method in this testing.

4 On occurrence of large errors

In this section, we clarify why so large errors occur frequently in the subresultant method and Collins' algorithm and improve the subresultant method simply. We also investigate the Gaussian elimination method briefly.

4.1 Term cancellation in the subresultant method

We first investigate a concrete case that $\deg_x(F) = n+1$ and $\deg_x(G) = n$:

$$F = f_{n+1}x^{n+1} + f_n x^n + \cdots + f_0, \quad G = g_n x^n + g_{n-1}x^{n-1} + \cdots + g_0. \tag{4.1}$$

The pseudo-remainder $P_3 \stackrel{\text{def}}{=} \text{remainder}(g_n^2 F, G)$ is given by

$$\begin{aligned} P_3 &= g_n^2 F - (g_n f_{n+1} x + g_n f_n - g_{n-1} f_{n+1}) G \\ &= \begin{vmatrix} g_n & g_{n-1} & g_{n-2} \\ & g_n & g_{n-1} \\ f_{n+1} & f_n & f_{n-1} \end{vmatrix} x^{n-1} + \cdots + \begin{vmatrix} g_n & g_{n-1} & g_{i-1} \\ & g_n & g_i \\ f_{n+1} & f_n & f_i \end{vmatrix} x^i + \cdots . \end{aligned} \tag{4.2}$$

We next calculate $\tilde{P}_4 \stackrel{\text{def}}{=} \text{remainder}(\text{lc}(P_3)^2 G, P_3)$:

$$\tilde{P}_4 = \sum_{i=0}^{n-2} \begin{vmatrix} \begin{vmatrix} g_n & g_{n-1} & g_{n-2} \\ & g_n & g_{n-1} \\ f_{n+1} & f_n & f_{n-1} \end{vmatrix} & \begin{vmatrix} g_n & g_{n-1} & g_{n-3} \\ & g_n & g_{n-2} \\ f_{n+1} & f_n & f_{n-2} \end{vmatrix} & \begin{vmatrix} g_n & g_{n-1} & g_{i-2} \\ & g_n & g_{i-1} \\ f_{n+1} & f_n & f_{i-1} \end{vmatrix} \\ & \begin{vmatrix} g_n & g_{n-1} & g_{n-2} \\ & g_n & g_{n-1} \\ f_{n+1} & f_n & f_{n-1} \end{vmatrix} & \begin{vmatrix} g_n & g_{n-1} & g_{i-1} \\ & g_n & g_i \\ f_{n+1} & f_n & f_i \end{vmatrix} \\ g_n & g_{n-1} & g_i \end{vmatrix} x^i. \tag{4.3}$$

The coefficient determinant of the x^i term is calculated as follows.

$$\begin{aligned} & g_n^5 \times (-\, f_{n-1}f_{i-1} + f_{n-2}f_i\ ) \\ +\ & g_n^4 \times (\ g_{n-1}f_nf_{i-1} - g_{n-1}f_{n-1}f_i + g_{n-2}f_{n+1}f_{i-1} - g_{n-2}f_nf_i - g_{n-3}f_{n+1}f_i \\ & \qquad - g_if_nf_{n-2} + g_if_{n-1}^2 - g_{i-1}f_{n+1}f_{n-2} + g_{i-1}f_nf_{n-1} + g_{i-2}f_{n+1}f_{n-1}\ ) \\ +\ & g_n^3 \times (-\, g_{n-1}^2f_{n+1}f_{i-1} + g_{n-1}^2f_nf_i + 2g_{n-1}g_{n-2}f_{n+1}f_i + g_{n-1}g_if_{n+1}f_{n-2} \\ & \qquad - g_{n-1}g_if_nf_{n-1} - g_{n-1}g_{i-1}f_n^2 - g_{n-1}g_{i-2}f_{n+1}f_n - 2g_{n-2}g_if_{n+1}f_{n-1} \\ & \qquad + g_{n-2}g_if_n^2 - g_{n-2}g_{i-2}f_{n+1}^2 + g_{n-3}g_if_{n+1}f_n + g_{n-3}g_{i-1}f_{n+1}^2\ ) \\ +\ & g_n^2 \times (-\, g_{n-1}^3f_{n+1}f_i + g_{n-1}^2g_if_{n+1}f_{n-1} + g_{n-1}^2g_{i-1}f_{n+1}f_n + g_{n-1}^2g_{i-2}f_{n+1}^2 \\ & \qquad - g_{n-1}g_{n-2}g_if_{n+1}f_n - g_{n-1}g_{n-2}g_{i-1}f_{n+1}^2 - g_{n-1}g_{n-3}g_if_{n+1}^2 + g_{n-2}^2g_if_{n+1}^2\ ) \\ +\ & g_n^1 \times (-\, \underline{g_{n-1}^3g_{i-1}f_{n+1}^2} + \underline{g_{n-1}^2g_{n-2}g_if_{n+1}^2} + \underline{2g_{n-1}^3g_if_{n+1}f_n} \\ & \qquad + \underline{g_{n-1}^3g_{i-1}f_{n+1}^2} + \underline{g_{n-1}^2g_{n-2}g_if_{n+1}^2} - \underline{2g_{n-1}^3g_if_{n+1}f_n} - \underline{2g_{n-1}^2g_{n-2}g_if_{n+1}^2}\ ) \\ +\ & g_n^0 \times (-\, \underline{g_{n-1}^4g_if_{n+1}^2} + \underline{g_{n-1}^4g_if_{n+1}^2}\ ). \end{aligned} \tag{4.4}$$

We note that terms $g_n^1 \times (\cdots)$ and $g_n^0 \times (\cdots)$ disappear completely by cancellation, although each term underlined appears during the computation.

The above kind of cancellation occurs always in the computation of P_4, P_4, $\cdots$. In order to see this, we consider formulas (2.7) with (2.8). According to [Bro78], β_i is a product of powers of $\mathrm{lc}(G)$, $\mathrm{lc}(P_3)$, $\ldots$. (If $d_2 = d_3 = \cdots = 1$ then $\beta_i = \mathrm{lc}(P_{i-1})^2$, $i = 3, 4, \cdots$.) Let $\tilde{P}_{i+1} = \mathrm{remainder}(\alpha_i P_{i-1}, P_i)$, then we have $\tilde{P}_{i+1} = \beta_i P_i$. On the other hand, $\tilde{P}_{i+1}$ can be expressed in terms of determinant, as in (4.3). Expanding the determinant, we see there appear terms which do not proportional to β_i, and formula (2.3) tells us that such terms must cancel exactly.

4.2 Error magnification in polynomial division

If the above underlined terms dominate the other terms, that is, if their numerical coefficients are much greater than others, then large cancellation errors will occur. However, at least for Example 2.1, this is not the case: we have checked the magnitudes of canceled terms and found that they are of almost the same magnitudes as uncanceled terms.

Then, why so large errors occur in the subresultant method ? The answer is error magnification in univariate polynomial division. Univariate polynomial division with floating-point arithmetic will magnify the numerical errors largely due to the cancellation, if the leading coefficient of the divisor is small and the degree of the dividend is large.

Let C and D be univariate polynomials in y :

$$C = c_l y^l + \cdots + c_0, \quad D = d_m y^m + \cdots + d_0, \quad (l > m). \tag{4.5}$$

Division of C by D is performed by eliminating the y^i terms, $i = l, l-1, \ldots, m$, of C successively as follows.

$$C' := C - (c_l/d_m) y^{l-m} \times D. \tag{4.6}$$

Note that $\mathrm{quotient}(C, D) = (c_l/d_m)y^{l-m} + \cdots$. Below, by $\|P\|$, with P a polynomial, we denote the maximum of the absolute numerical coefficients of P.

Now, putting $C = QD + R$, $\deg(R) < \deg(D)$, suppose we have $\|C\| = \|D\| = 1$ and $\|Q\| \simeq 1$, for simplicity. Applying the subresultant method to Example 2.1, we encounter a situation that $0 < |d_m| \ll 1$ and some coefficients of higher degree terms of C, let one of them be $c_{\bar{l}}$ with $\bar{l} \geq m$, are as large as 1. Let an intermediate dividend $C'_{l'}$ be $C'_{l'} = c'_{l'} y^{l'} + \cdots + c'_0$, then $c'_{l'}/d_m$ is the coefficient of $y^{l'-m}$ term of Q. Since $\|Q\| \simeq 1$, we must have $|c'_{l'}/d_m| \lesssim 1$ for all l' hence for $l' = \bar{l}$, too. This means that $|c'_{l'}| \lesssim |d_m| \ll 1$. Therefore, during the elimination of $y^l, y^{l-1}, \ldots, y^{\bar{l}}$ terms of C, the accuracy of the coefficient of $y^{\bar{l}}$ term is decreased by an amount of $|1/d_m|$ or larger due to the cancellation. This will increase relative errors in not only Q but also R, because the error of the leading coefficient propagates to the succeeding intermediate dividends. We note that the lower the degrees of terms of Q and R are, the larger the relative errors of their coefficients are.

Since the subresultant method employs the division operation, we may encounter a large numerical errors. The middle column of Table V shows the maximum relative errors contained in the remainders appearing in the computation of $\mathrm{resultant}_x(F_{2,1}/3.0, G_{2,1}/3.0)$.

remainders	original method	improved method
$P_1 = F_{2,1}/3.0$		
$P_2 = G_{2,1}/3.0$		
$P_3 = \mathrm{prem}(P_1, P_2)$	1.0×10^{-16}	1.0×10^{-16}
$\tilde{P}_4 = \mathrm{prem}(P_2, P_3)$	4.6×10^{-13}	1.0×10^{-16}
$P_4 = \tilde{P}_4 \div \mathrm{lc}(P_2)^2$	7.0×10^{-12}	1.0×10^{-16}
$\tilde{P}_5 = \mathrm{prem}(P_3, P_4)$	9.1×10^{-04}	4.0×10^{-11}
$P_5 = \tilde{P}_4 \div \mathrm{lc}(P_3)^2$	1.8×10^{-04}	1.7×10^{-11}

Table V : Maximum relative errors in subresultant methods.
"original method" uses the multivariate division. See **4.3** for "improved method".

We explain the computation of $\tilde{P}_5$ explicitly. Putting

$$P_3 = C_2(y)x^2 + C_1(y)x + C_0(y), \quad P_4 = D_1(y)x + D_0(y), \tag{4.7}$$

for P_3 and P_4 in Table V, we have

$$\begin{array}{rclrcl} C_2 &=& 0.076\cdots y^6 + \cdots, & \|C_2\| &\simeq& 0.59, \\ D_1 &=& 0.0082\cdots y^{11} + \cdots, & \|D_1\| &\simeq& 1.9, \\ D_1^2 C_2 &=& 0.0000050\cdots y^{28} + \cdots, & \|D_1^2 C_2\| &\simeq& 5.0. \end{array}$$

In the computation of $\mathrm{prem}(P_3, P_4)$, a univariate polynomial $D_1^2 C_2$ of degree 28 is divided by a univariate polynomial D_1 with a small leading coefficient $\sim 0.0082\cdots$. This division will cause large numerical errors in both the quotient and the remainder.

Of course, the subresultant method does not always cause large errors, because it does not always produce remainders with small leading coefficients. However, considering that a power of a high degree polynomial is multiplied and divided in the pseudo-remainder operation, we will often encounter univariate polynomials with small leading coefficients.

4.3 An improvement of the subresultant method

We can reduce numerical errors due to the pseudo-remainder computation largely by a simple trick. The pseudo-division is nothing but to eliminate the leading terms of the dividend by the divisor, and we can perform this elimination without using univariate and numerical divisions. For concreteness, let us consider P_3 and P_4 in (4.7). The x^2 term of P_3 is eliminated by P_4 as

$$P_3' := D_1 P_3 - x C_2 P_4 = (D_1 C_1 - C_2 D_0)\, x + D_1 C_0.$$

Eliminating x^1 term of P_3' further, we can calculate $\mathrm{prem}(P_3, P_4)$ as

$$\mathrm{prem}(P_3, P_4) := D_1 P_3' - \mathrm{lc}(P_3') P_4.$$

This computation does not employ univariate or numerical divisions, hence we expect that $\tilde{P}_5$ computed in this way is quite accurate. In fact, we have max-rel-error$(\tilde{P}_5) = 1.4 \times 10^{-11}$.

In the "improved" columns of Tables II and V, we give maximum relative errors of resultants computed in this way. We see that the above simple trick reduces the errors largely. We must note, however, that the division in calculating P_4 and P_5 from $\tilde{P}_4$ and $\tilde{P}_5$, respectively, cannot be removed by the above simple trick.

4.4 Cancellation in Collins' algorithm

Let us consider Lagrange's interpolation formula which plays an essential role in Collins' algorithm with floating-point number arithmetic. Let the evaluation points be $(y_0, y_1, \ldots, y_N)$ and the evaluation values be $(v_0, v_1, \ldots, v_N)$. Then, Lagrange's formula determines a polynomial $P(y)$ of degree $\le N$, as

$$P(y) = \sum_{i=0}^{N} v_i \cdot \frac{\prod_{j=0,\neq i}^{N} (y - y_j)}{\prod_{j=0,\neq i}^{N} (y_i - y_j)}. \tag{4.8}$$

We see $P(y_i) = v_i$ $(i = 0, 1, \ldots, N)$, easily. Putting

$$Q(y) = \prod_{j=0}^{N} (y - y_j), \quad Q'(y) = \frac{\mathrm{d}Q(y)}{\mathrm{d}y}, \tag{4.9}$$

we can express $P(y)$ as

$$P(y) = \sum_{i=0}^{N} v_i \cdot P_i(y), \quad \text{where} \quad P_i(y) = \frac{Q(y)/(y - y_i)}{Q'(y_i)}. \tag{4.10}$$

The polynomial $P_i(y)$ depends on the value of i and the choice of evaluation points largely, and the logarithms of its coefficients range very widely. For example, consider three choices by setting $N = 20$.

$$\begin{array}{ll}
\text{Choice A}: & (y_0, y_1, \cdots, y_{20}) = (-10.,\ -9.0,\ -8.0,\ \cdots,\ 0,\ \cdots,\ 9.0,\ 10.)\ , \\
\text{Choice B}: & (y_0, y_1, \cdots, y_{20}) = (-3.0,\ -2.7,\ -2.4,\ \cdots,\ 0,\ \cdots,\ 2,7,\ 3.0)\ , \\
\text{Choice C}: & (y_0, y_1, \cdots, y_{20}) = (-1.0,\ -0.9,\ -0.8,\ \cdots,\ 0,\ \cdots,\ 0.9,\ 1.0)\ .
\end{array}$$

We show several terms of P_{10}, P_{15}, and P_{20} for Choices A, B, and C.

$$\text{A} \quad \left\{ \begin{array}{lcl}
P_{10} & = & 7.59\cdots 10^{-14} y^{20} + \cdots - 6.27\cdots 10^{-4} y^{10} + \cdots + 1, \\
P_{15} & = & -6.37\cdots 10^{-15} y^{20} + \cdots + 2.56\cdots 10^{-5} y^{10} + \cdots + 0.0167\cdots y, \\
P_{20} & = & 4.11\cdots 10^{-19} y^{20} + \cdots - 5.62\cdots 10^{-10} y^{10} + \cdots - 5.41\cdots 10^{-7} y,
\end{array} \right.$$

$$\text{B} \quad \left\{ \begin{array}{lcl}
P_{10} & = & 0.00217\cdots y^{20} + \cdots - 106.0\cdots y^{10} + \cdots + 1, \\
P_{15} & = & -1.82\cdots 10^{-4} y^{20} + \cdots + 4.34\cdots y^{10} + \cdots + 0.0559\cdots y, \\
P_{20} & = & 1.17\cdots 10^{-8} y^{20} + \cdots - 9.51\cdots 10^{-5} y^{10} + \cdots - 1.80\cdots 10^{-6} y,
\end{array} \right.$$

$$\text{C} \quad \left\{ \begin{array}{lcl}
P_{10} & = & 7.59\cdots 10^{6} y^{20} + \cdots - 6.27\cdots 10^{6} y^{10} + \cdots + 1, \\
P_{15} & = & -6.37\cdots 10^{5} y^{20} + \cdots + 2.56\cdots 10^{5} y^{10} + \cdots + 0.167\cdots y, \\
P_{20} & = & 41.1\cdots y^{20} + \cdots - 5.62\cdots y^{10} + \cdots - 5.41\cdots 10^{-6} y.
\end{array} \right.$$

Furthermore, the evaluation value v_i also changes very largely as i changes. That is, Collins' algorithm constructs a target polynomial the coefficients of which are usually of moderate sizes by adding very different polynomials the coefficients of which range very widely. Hence, main parts of terms of large coefficients will cancel one another, and we will encounter large cancellation errors usually.

4.5 Term cancellation in Bareiss' algorithm

Finally, we consider Bareiss' algorithm for Gaussian elimination which also causes cancellation of terms systematically. For concreteness, let us consider the second elimination step (i.e., calculation of $\tilde{M}_{i,j}^{(2)}$).

$$\begin{aligned}
\tilde{M}_{i,j}^{(2)} &= \left| \begin{array}{cc} \left| \begin{array}{cc} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{array} \right| & \left| \begin{array}{cc} a_{1,1} & a_{1,j} \\ a_{2,1} & a_{2,j} \end{array} \right| \\ \left| \begin{array}{cc} a_{1,1} & a_{1,2} \\ a_{i,1} & a_{i,2} \end{array} \right| & \left| \begin{array}{cc} a_{1,1} & a_{1,j} \\ a_{i,1} & a_{i,j} \end{array} \right| \end{array} \right| \qquad (4.11) \\
&= a_{1,1}^2 \times (a_{2,2}a_{i,j} - a_{i,2}a_{2,j}) \\
&- a_{1,1}^1 \times (a_{2,2}a_{i,1}a_{1,j} + a_{1,2}a_{2,1}a_{i,j} - a_{i,2}a_{2,1}a_{1,j} - a_{i,1}a_{1,2}a_{2,j}) \\
&+ a_{1,1}^0 \times (\underline{a_{1,2}a_{2,1}a_{i,1}a_{1,j}} - \underline{a_{i,1}a_{1,2}a_{2,1}a_{1,j}}).
\end{aligned}$$

We see that the underlined terms cancel each other. The same kind of cancellation occurs at every elimination step except for the first. At the $(k+1)$-st step, we calculate $\tilde{M}_{i,j}^{(k+1)}$ given in (2.13). $\tilde{M}_{i,j}^{(k+1)}$ contains a factor $\tilde{M}_{k,k}^{(k-1)}$. However, in the computation of $\tilde{M}_{i,j}^{(k+1)}$, there appear terms which do not contain the factor $\tilde{M}_{k,k}^{(k-1)}$, hence such terms must cancel one another.

The above cancellation itself does not cause much errors. Large errors listed in Table II for the Gaussian elimination method are due to the error magnification by polynomial division. However, the error magnification is not so large as that in the subresultant method, because the sizes of polynomials which are multiplied and then divided out are smaller in Bareiss' method than in the subresultant method.

5 Conclusions

We have tested five methods of multivariate resultant computation with fixed-precision floating-point number arithmetic and analyzed the occurrence of large cancellation errors, which show the followings clearly.

1) There occur large cancellation errors if executed is the division of a univariate polynomial of a high degree by another polynomial with a small leading coefficient, as the subresultant method shows.

2) Interpolation of high degree polynomials will cause large cancellation errors usually, as Collins' algorithm shows.

3) Algorithms which do not employ polynomial division are safe against the numerical errors, unless the input polynomials have an approximate common factor, as the minor expansion method and the efficient Gaussian elimination method show.

4) Unless the target resultant is an extremely large polynomial, the minor expansion method and the efficient Gaussian elimination methods are recommendable to use, both in two variables and many variables cases.

Furthermore, from the viewpoint of approximate algebraic computation, we must point out the followings.

a) *Although approximate algebraic computation seems to be quite fruitful, we must be careful enough of numerical errors.*

b) *Numerical errors in approximate algebraic computation are never small; all the significant digits may be lost frequently by the cancellation. Therefore, error analysis is indispensable for approximate algebraic algorithms.*

c) *Even a simple improvement may reduce the numerical errors drastically. This shows necessity and importance of development of error-safe algorithms.*

References

[Bar68] E. H. Bareiss, Sylvester's identity and multistep integer-preserving Gaussian elimination. Math. Comput. **22** (1968), 565-578.

[BT71] W. S. Brown and J. F. Traub, On Euclid's algorithm and the theory of subresultants. J. ACM, **18** (1971), 505-514.

[Col67] J. E. Collins, Subresultant and reduced polynomial remainder sequence. J. ACM, **14** (1967), 128-142.

[Col71] J. E. Collins, The calculation of multivariate polynomial resultants. J. ACM, **18** (1971), 515-532.

[KA69] S. Y. Ku and R. J. Adler, Computing polynomial resultants: Bezout's determinant vs. Collins' reduced P.R.S. algorithm. Communi. ACM, **12** (1969), 23-30.

[Lip71] J. D. Lipson, Chinese remainder and interpolation algorithms. Proc. SYMSAC, ACM, 372-391, 1971.

[Sas91] T. Sasaki, Formula manipulation system GAL, *Proc. on Computing in High Energy Physics '91*, Universal Academic Press, Inc. (Tokyo), 381-389, 1991.

[SM82] T. Sasaki and H. Murao, Efficient Gaussian elimination method and linear systems. ACM Trans. Math. Software **8** (1982), 277-289.