Conclusion

In this note, we have outlined how we conducted our research with the help of a computer algebra system. As was shown, the use of algebraic devices (working modulo some polynomial) to compensate the weaknesses of the simplification algorithm(s) for complex expressions, gave us much more than just a convenient solution. One of the outstanding effects of this use of MAPLE was the unveiling of unexpected formulas such as (1), (2), (3), and (4). That is why this approach ought to be publicized.

Bibliography


Exact Algorithms for the Matrix-Triangularization Subresultant PRS Method

Alkiviadis G. Akritas
University of Kansas
Department of Computer Science
Lawrence, Kansas, 66045

Abstract. In [2] a new method is presented for the computation of a greatest common divisor (gcd) of two polynomials, along with their polynomial remainder sequence (prs). This method is based on our generalization of a theorem by Van Vleck (1899) [12] and uniformly treats both normal and abnormal prs's, making use of Barcisz's (1968) [4] integer-preserving transformation algorithm for Gaussian elimination; moreover, for the polynomials of the prs's, this method provides the smallest coefficients that can be expected without coefficient gcd computations. In this paper we present efficient, exact algorithms for the implementation of this new method, along with an example where bubble pivot is needed.

1. Introduction

In this note we restrict our discussion to univariate polynomials with integer coefficients and to computations in $\mathbb{Z}[x]$, a unique factorization domain. Given the polynomial $p(x) = c_nx^n + c_{n-1}x^{n-1} + \ldots + c_0$, its degree is denoted by $\deg(p(x))$ and $c_n$, its leading coefficient, by $\text{lc}(p)$; moreover, $p(x)$ is called primitive if its coefficients are relatively prime.

Consider now $p_1(x)$ and $p_2(x)$, two primitive, nonzero polynomials in $\mathbb{Z}[x]$. $\deg(p_1(x)) = n$ and $\deg(p_2(x)) = m$, $n \geq m$. Clearly, the polynomial division (with remainder) algorithm, call it P1, that works over a field, cannot be used in $\mathbb{Z}[x]$ since it requires exact divisibility by $\text{lc}(p_2)$. So we use pseudo-division, which always yields a pseudo-quotient and pseudo-remainder, in this
In general, if we have the polynomial remainder sequence \( r(x), p_2(x), p_3(x), \ldots, p_n(x) \), \( \deg(p_1(x)) = n, \deg(p_2(x)) = m, n \geq m \), we can obtain the (negated) coefficients of the \((i+1)\)th member of the PRS, \( i = 0, 1, 2, \ldots, n-1 \), as minors formed from the first \( 2i \) rows of \((S)\) by successively associating with the first \( 2i-1 \) columns of \((2i)\) by \((2n)\) matrix each succeeding column in turn.

On the other hand, we transform the matrix corresponding to the resultant \((S)\) into its upper triangular form using Bareiss's integer-preserving transformation algorithm [4]. That is:

\[
\begin{align*}
\begin{pmatrix}
0, & 1, & \ldots, & n, & \ldots, & 0
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{pmatrix}
r_{k+1,k-1} & \cdots & r_{k+1,1} & r_{k+1,0} \\
r_{k,1} & \cdots & r_{k,1} & r_{k,0} \\
\vdots & \ddots & \vdots & \vdots \\
r_{0,0} & \cdots & r_{0,1} & r_{0,0}
\end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\begin{pmatrix}
r_{k,k-1} & \cdots & r_{k,1} & r_{k,0} \\
r_{k,k-1} & \cdots & r_{k,1} & r_{k,0} \\
\vdots & \ddots & \vdots & \vdots \\
r_{k,k-1} & \cdots & r_{k,1} & r_{k,0}
\end{pmatrix}
\end{align*}
\]

Of particular importance in Bareiss's algorithm is the fact that the determinant of order \( 2 \) is divided exactly by \( r_{k-1,k-1}^{-1} \) (the proof is very short and clear and is described in Bareiss's paper [4]) and that the resulting coefficients are the smallest that can be expected without coefficient gcd computations and without introducing rationals. Notice how all the complicated expressions for \( \beta_i \) in the reduced and subresultant PRS algorithms are mapped to the simple factor \( r_{k-1,k-1} \) of this method.

It should be pointed out that using Bareiss's algorithm we have to perform pivots (interchange two rows) which will result in a change of signs. We also define the term bubble pivot as follows: if the diagonal element in row \( i \) is zero and the next nonzero element down the column is in row \( i+j \), \( j > 1 \), then row \( i+j \) will become row \( i \) after pairwise interchanging it with the rows above it. Bubble pivot preserves the symmetry of the determinant.

We have the following theorem.

**Theorem 2** ([2]). Let \( p_1(x) \) and \( p_2(x) \) be two polynomials of degrees \( n \) and \( m \) respectively, \( n \geq m \). Using Bareiss's algorithm transform the matrix corresponding to \( \text{resg}(p_1(x), p_2(x)) \) into its upper triangular form \((T)\); let \( n_1 \) be the degree of the polynomial corresponding to the \( i \)th row of \((T)\), \( i = 1, 2, \ldots, 2n \), and let \( p_{k-1}(x) \), \( k \geq 2 \), be the \( k \)th member of the (normal or abnormal) polynomial remainder sequence of \( p_1(x) \) and \( p_2(x) \). Then if \( p_k(x) \) is in row \( i \) of \((T)\), the coefficients of \( p_{k+1}(x) \) (within sign) are obtained from row \( i+j \) of \((T)\), where \( j \) is the smallest integer such that \( n_{i+j} < n_i \) (If \( n = m \) associate both \( p_1(x) \) and \( p_2(x) \) with the first row of \((T)\).)

Notice that as a special case of the above theorem we obtain Von Volkmann's theorem for normal PRS's. We see, therefore, that based on Theorem 2, we have a new method to compute the polynomial remainder sequence and a greatest common divisor of two polynomials. This new method uniformly treats both normal and abnormal PRS's and provides the smallest coefficients that can be expected without coefficient gcd computation.

### 3. Our method and its implementation

The inputs are two (primitive) polynomials in \( \mathbb{Z}[x], p_1(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_0 \) and \( p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \ldots + d_0, c_n \neq 0, d_m \neq 0, n \geq m \).

**Step 1:** Form the resultant \((S)\), \( \text{resg}(p_1(x), p_2(x))\), of the two polynomials \( p_1(x) \) and \( p_2(x) \).

**Step 2:** Using Bareiss's algorithm (described above) transform the resultant \((S)\) into its upper triangular form \((T)\); then the coefficients of all the members of the polynomial remainder sequence of \( p_1(x) \) and \( p_2(x) \) are obtained from the rows of \((T)\) with the help of Theorem 2.

For this method we have proved [2] that its computing time is:

**Theorem 3.** Let \( p_1(x) = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_0 \) and \( p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \ldots + d_0 \). \( c_n \neq 0, d_m \neq 0, n \geq m \) be two (primitive) polynomials in \( \mathbb{Z}[x] \) and for some polynomial in \( \mathbb{Z}[x] \) let \( p|_{\text{max}} \) represent its maximum coefficient in absolute value. Then the method described above computes a greatest common divisor of \( p_1(x) \) and \( p_2(x) \) along with all the polynomial remainders in time

\[
O(n^2 L(p|_{\text{max}})^2)
\]

where \( p|_{\text{max}} = \max \{ |p_1|_{\text{max}}, |p_2|_{\text{max}} \} \).

Below we present efficient exact (maple-like) algorithms for the matrix-triangulization subresultant PRS method. A subalgorithm call is the name of the subalgorithm in all bold letters. All subalgorithm calls are from the main algorithm. Parameters (arguments) are not shown. Comments are made within braces { }. An explanation of the variables is found after the algorithms.
process we have to premultiply $p_1(x)$ by $\text{lcm}(p_2(x))^{-1}$ and then apply algorithm PD. Therefore we have:

$$\text{lcm}(p_2(x))^{-1}p_1(x) = q(x)p_2(x) + p_3(x), \quad \deg(p_3(x)) < \deg(p_2(x)).$$

(1)

Applying the same process to $p_2(x)$ and $p_3(x)$, and then to $p_3(x)$ and $p_4(x)$, etc. (Euclid's algorithm), we obtain a polynomial remainder sequence (prs)

$$p_1(x), p_2(x), p_3(x), \ldots, p_t(x), p_{t+1}(x) = 0,$$

where $p_t(x) \neq 0$ is a greatest common divisor of $p_1(x)$ and $p_2(x)$, $\text{gcd}(p_1(x), p_2(x))$. If $n_i = \deg(p_i(x))$ and we have $n_i \cdot n_{i+1} = 1$, for all $i$, the prs is called normal, otherwise, it is called abnormal. The problem with the above approach is that the coefficients of the polynomials in the prs grow exponentially and hence slow down the computations. We wish to control this coefficient growth. We observe that equation (1) can also be written more generally as

$$\text{lcm}(p_{i+1}(x))^{-1}n_{i+1}^{-1}p_i(x) = q_i(x)p_{i+1}(x) + p_{i+2}(x), \quad \deg(p_{i+1}(x)) < \deg(p_{i+2}(x)).$$

(2)

$i = 1, 2, \ldots, h-1$. That is, if a method for choosing $b_i$ is given, the above equation provides an algorithm for constructing a prs. The obvious choice $b_i = 1$, for all $i$, is called the Euclidean prs; it was described above and leads to exponential growth of coefficients. Choosing $b_i$ to be the greatest common divisor of the coefficients of $p_{i+1}(x)$ results in the primitive prs, and it is the best that can be done to control the coefficient growth. (Notice that here we are dividing $p_{i+1}(x)$ by the greatest common divisor of its coefficients before we use it again.) However, computing the greatest common divisor of the coefficients for each member of the prs (after the first two, of course) is an expensive operation and should be avoided. So far, in order both to control the coefficient growth and to avoid the coefficient gcd computations, either the reduced or the (improved) subresultant prs have been used. In the reduced prs we choose

$$b_1 = 1 \text{ and } b_i = \text{lcm}(p_i) n_i^{-1} + 1, \quad i = 2, 3, \ldots, h-1,$$

(3)

whereas, in the subresultant prs we have

$$b_1 = (-1)^{n_1-2} n_1^{-2} \text{ and } b_i = (-1)^{n_i+1} n_i^{-1} n_{i+1}^{-1} \text{lcm}(p_i) n_i^{-1} n_i^{-1}, \quad i = 2, 3, \ldots, h-1,$$

(4)

where

$$H_2 = \text{lcm}(p_2) n_2^{-2} \text{ and } H_i = \text{lcm}(p_i) n_i^{-1} n_{i+1}^{-1} \cdot (n_i-1 \cdot n_i).$$

That is, in both cases above we divide $p_{i+2}(x)$ by the corresponding $b_i$ before we use it again. The reduced prs algorithm is recommended if the prs is normal, whereas if the prs is abnormal the subresultant prs algorithm is to be preferred. The proofs that the $b_i$'s shown in (3) and (4) exactly divide $p_{i+2}(x)$ are very complicated [7] and have up to now obscured simple divisibility properties [10], (see also [5] and [6]). For a simple proof of the validity of the reduced prs see [1], analogous for the subresultant prs can be found in [8].

In contrast with the above prs algorithms, the matrix-triangularization subresultant prs method avoids explicit polynomial divisions (explained below). In what follows we present efficient, exact algorithms for the implementation of this method. We also present an example where bubble pivot is needed.

2. Gaussian elimination and Sylvester's form of the resultant

Consider the two polynomials in $\mathbb{Z}[x]$.

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_0 \quad \text{and} \quad p_2(x) = d_m x^m + d_{m-1} x^{m-1} + \cdots + d_0, \quad c_n \neq 0, \quad d_m \neq 0, \quad n \geq m.$$

Contrary to established practice, we choose to call Sylvester's form of the resultant of $p_1(x)$ and $p_2(x)$ the one described below; this form was "buried" in Sylvester's 1853 paper [11] and is only once mentioned in the literature in a paper by Van Vleck [12]. Sylvester indicates ([11], p.420) that he had produced this form in 1839 or 1840 and some years later Cayley unconsciously reproduced it as well. It is Sylvester's form of the resultant that forms the foundation of our new method for computing polynomial remainder sequences; however, we first present the following theorem concerning Bruno's form of the resultant (the form encountered most often in the literature under the Sylvester's name):

**Theorem 1 (Laidacker[9]).** If we transform the matrix corresponding to $\text{res}_{\mathbb{F}}(p_1(x), p_2(x))$ into its upper triangular form $T_{\mathbb{F}}(R)$, using row transformations only, then the last nonzero row of $T_{\mathbb{F}}(R)$ gives the coefficients of a greatest common divisor of $p_1(x)$ and $p_2(x)$.

The above theorem indicates that we can obtain only a greatest common divisor of $p_1(x)$ and $p_2(x)$ but none of the remainder polynomials. In order to compute both a $\text{gcd}(p_1(x), p_2(x))$ and all the polynomial remainders we have to use Sylvester's form of the resultant; this is of order $2n$ (as opposed to $n \cdot m$ for the other forms) and of the following form (for $p_2(x)$ has been transformed into a polynomial of degree $n$ by introducing zero coefficients):
initialize

set resultant matrix to zero, and initialize the variables used

getpolys

get coefficients of the first two polynomials

buildmatrix

build the matrix corresponding to Sylvester's form of the resultant

set k to 1

[k is the index for the transformation loop]

while k < n do

(loop n-1 times, unless gcd is found (see pivot))

if r[k,k] = 0 then pivot fi

[need to put a non-zero element into r[k,k]]

if k < n then

(in pivot, if gcd is found k is set to n+1)

do transform; set d to r[k,k] od

fi

set k to k+1 {increment main loop index}

od

generate

n1 := deg(p1(x));

[deg(p1(x)) >= deg(p2(x))]

n := 2*n1;

for i from 1 to n do

for j from 1 to n do

r[i,j] := 0

[see notes on variables below]

od

trans[i] := false

[see notes on variables below]

od

d := 1

[no division for first transformation]

end

pivot

(check across row k for all zeros, this means row k-1 is gcd)

cflag := true;

for i := k+1; [i is the index for loop]

while (i <= L[k]) and cflag do

[loop across row]

if r[i,k] <> 0 then cflag := false fi

i := i + 1

[increment loop index]

od

if cflag then

[need to zero matrix below row k and stop processing]

for i from k+1 to n do

for j from k to n do

r[i,j] := 0

od

od

k := n + 1

[this stops main loop]

else

[need to find a row s without a zero in column k to pivot up]

s := k + 1

[start looking one row below k]

while r[s,k] = 0 do

[loop while value in column k is zero]

s := s + 1

od

[move row s to row k with bubble pivot]
printmatrix
 [this is dependent
he language used; print each row and column]for i from 1 to n do
 for j from 1 to n do
 write r[i,j] {on one line} od;
 advance a line od;
end

The variables
1. r[i,j] is a two dimensional matrix (array).
2. n1 = deg(p1(x)).
3. n = 2*n1 is the length and width of the resultant (matrix).
4. L[i] is the location of the last element in row i; this is important because it is used so that we do not update the zero elements of a row.
5. tran[i] is a one dimensional boolean (or logical) array; it is true when row i was transformed during the last transformation; this is important since only transformed rows may be divided by d.
6. d is the value which a transformed row may be divided by if all other factors allow for division. In the Bareiss transform d is r[k-1,k-1].
7. k is the current transformation number and r[k,k] is the corner element where the next transformation will begin.
8. tempint, tempbool and temprow are temporary variables used for pivoting.
9. ek4gcd is a boolean (logical) variable which will be true when row k is all zeros. This means a greatest common divisor (gcd) has been found and further transformations are not necessary.

Below we present an incomplete example where bubble pivoting is needed. [3]; note that there is a difference of 3 in the degrees of the members of the prs, as opposed to a difference of 2 in Knuth's "classic" incomplete example.

Example. Let us find the polynomial remainder sequence of the polynomials p1(x) = 3x^9 + 5x^8 + 7x^7 - 3x^6 - 5x^5 - 7x^4 + 3x^3 + 5x^2 + 7x - 2 and p2(x) = x^8 - x^5 - x^2 - x - 1. This incomplete prs example presents a variation of three in the degrees of its members (from 7 to 4) and it requires a bubble pivot in the matrix-triangulization method; that is, a pivot will take place between rows that are not adjacent.
The matrix-antiarization subresultant prs method

row 1> 3 5 7 -3 -5 7 -5 7 -2 0 0 0 0 0 0 0 0 (9)
2> 0 1 0 0 -1 0 0 -1 -1 0 0 0 0 0 0 0 0 (8)
3> 0 0 5 7 0 -5 7 6 8 10 -2 0 0 0 0 0 0 (8)
4> 0 0 0 0 -7 0 0 7 -6 -13 -15 -3 0 0 0 0 0 (7)
5> 0 0 0 0 -49 0 0 79 23 19 -55 14 0 0 0 0 0 (7)
6#> 0 0 0 0 0 -343 0 -24 501 73 93 -413 98 0 0 0 0 0 (7)
7#> 0 0 0 0 0 -2401 -510 -1273 1637 -339 56 -2891 686 0 0 0 0 0 (7)
8> 0 0 0 0 0 0 2058 4459 7546 3430 2401 0 0 0 0 0 0 (4)
9> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
10> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
11> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
12> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
13> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
14> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
15> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
16> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
17> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)
18> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (4)

Largest integer generated is 27843817119202448 (17 digits).
Pivoted row 6 during transformation 6. Stored row is:
6> 0 0 0 0 0 0 42 91 154 70 49 0 0 0 0 0 (4)
Pivoted row 7 during transformation 7. Stored row is:
7> 0 0 0 0 0 0 294 637 1078 490 343 0 0 0 0 0 (4)

Bibliography