Serdica J. Computing **10** (2016), No 3–4, 197–217

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

SUBRESULTANT POLYNOMIAL REMAINDER SEQUENCES OBTAINED BY POLYNOMIAL DIVISIONS IN $\mathbb{Q}[x]$ OR IN $\mathbb{Z}[x]$

Alkiviadis G. Akritas, Gennadi I. Malaschonok*,
Panagiotis S. Vigklas

Abstract. In this paper we present two new methods for computing the subresultant polynomial remainder sequence (prs) of two polynomials $f, g \in \mathbb{Z}[x]$. We are now able to also *correctly* compute the Euclidean and modified Euclidean prs[1] of f, g by using either of the functions employed by our methods to compute the remainder polynomials.

Another innovation is that we are able to obtain subresultant prs's in $\mathbb{Z}[x]$ by employing the function `rem(f, g, x)` to compute the remainder polynomials in $\mathbb{Q}[x]$. This is achieved by our method `subresultants_amv_q (f, g, x)`, which is somewhat slow due to the inherent higher cost of computations in the field of rationals.

ACM Computing Classification System (1998): F.2.1, G.1.5, I.1.2.

Key words: Euclidean algorithm, Euclidean polynomial remainder sequence (prs), modified Euclidean (Sturm's) prs, subresultant prs, modified subresultant prs, Sylvester matrices, Bezout matrix, Van Vleck's method, sympy.

*The second author was partially supported by RFBR grant No 16-07-00420.

[1]Also known as generalized Sturmian prs.

To improve in speed, our second method, `subresultants_amv(f, g, x)`, computes the remainder polynomials in the ring $\mathbb{Z}[x]$ by employing the function `rem_z(f, g, x)`;[2] the time complexity and performance of this method are very competitive.

Our methods are two different implementations of Theorem 1 (Section 3), which establishes a one-to-one correspondence between the Euclidean and modified Euclidean prs of f, g, on one hand, and the subresultant prs of f, g, on the other.

By contrast, if – as is currently the practice – the remainder polynomials are obtained by the pseudo-remainders function `prem(f, g, x)`[3], then *only* subresultant prs's are correctly computed. Euclidean and modified Euclidean prs's generated by this function may cause confusion with the signs and conflict with Theorem 1.

1. Introduction. We assume Euclidean and modified Euclidean (or Sturmian) prs's well known and we informally define subresultant prs's.

Consider the polynomials $f, g \in \mathbb{Z}[x]$ of degrees $deg(f) = n$ and $deg(g) = m$ with $n \geq m$. The *subresultant prs* of f, g is a sequence of polynomials in $\mathbb{Z}[x]$ analogous to the Euclidean prs, the sequence obtained by applying on f, g Euclid's algorithm for polynomial greatest common divisors (gcd) in $\mathbb{Q}[x]$.

The subresultant prs differs from the Euclidean prs in that the coefficients of each polynomial in the former are the determinants – also referred to as *subresultants* – of appropriately selected sub-matrices of `sylvester1(f, g, x)`[4], Sylvester's matrix of 1840 of dimensions $(n + m) \times (n + m)$ [14].

Recall that the determinant of `sylvester1(f, g, x)` itself is called the *resultant* of f, g and serves as a criterion of whether the two polynomials have common roots or not.

In the sequel we will be talking about Euclidean, modified Euclidean and subresultant prs's [5]. Statements about a prs – unless specifically identified – apply to all three sequences.

To compute a prs in $\mathbb{Z}[x]$, the current practice is to use pseudo-remainders [7], [8], [9], [10], [11], [12], which are defined by

$$\mathrm{LC}(R_{i-1})^{\delta} \cdot R_{i-2} = q_{i-2} \cdot R_{i-1} + R_i, \tag{1}$$

[2]Defined by equation (4) in Section 1.

[3]Defined by equation (1) in Section 1.

[4]To distinguish it from `sylvester2(f, g, x)`, Sylvester's matrix of 1853 of dimensions $(2 \cdot n) \times (2 \cdot n)$ [15].

where R_i is the pseudo-remainder, $\mathrm{LC}(R_{i-1})$ is the leading coefficient of the divisor R_{i-1}, and

$$\delta = \mathrm{degree}(R_{i-2}, x) - \mathrm{degree}(R_{i-1}, x) + 1. \tag{2}$$

It is of importance to note that equation (1) employs the leading coefficient of the divisor R_{i-1} and *not* its absolute value. In `sympy` – one of the freely available Computer Algebra Systems (CAS's) – pseudo-remainders are computed by the function `prem(f, g, x)`.

As long as

$$\mathrm{LC}(R_{i-1}) > 0$$

or

$$\mathrm{LC}(R_{i-1}) < 0 \text{ and } \delta \text{ is even},$$

any prs is correctly computed with the above definition of pseudo-remainders. Of interest is the fact that *only* subresultant prs's are correctly computed with `prem(f, g, x)`, [12], when

$$\mathrm{LC}(R_{i-1}) < 0 \text{ and } \delta \text{ is odd}. \tag{3}$$

By contrast, when the two conditions in (3) hold, there may be confusion regarding the signs in the Euclidean and modified Euclidean prs's computed with `prem(f, g, x)`; moreover, conflict may arise with Theorem 1.

Theorem 1 establishes a one-to-one correspondence between the Euclidean and modified Euclidean prs of f, g, on one hand, and the subresultant prs of f, g, on the other.

As detailed in Example 1 of Section 17, a conflict with Theorem 1 may arise when the sign sequences (see Definition 2) of the Euclidean or modified Euclidean prs of f, g computed in $\mathbb{Z}[x]$ with `prem(f, g, x)` *are not identical* with the corresponding ones computed in $\mathbb{Q}[x]$. In such a case, the Euclidean and modified Euclidean prs of f, g computed in $\mathbb{Z}[x]$ – unlike their counterparts computed in $\mathbb{Q}[x]$ – *are not* in a one-to-one correspondence with the subresultant prs of f, g.

To facilitate our further discussion we introduce the following definition:

Definition 1. *A polynomial remainder sequence of two polynomials f, g is called* ***complete*** *if the degree difference between any two consecutive polynomials is 1; otherwise, it is called* ***incomplete***.[5]

[5] It is understood that f, g are included in the prs.

By (2) it becomes clear that δ may be odd *only* when the prs is incomplete. Therefore, it is incomplete Euclidean and modified Euclidean prs's that may create confusion with signs and conflict with Theorem 1 [2], [3].

The confusion with the signs was first noted in `http://planetmath.org/sturmstheorem`, from where we quote:

> "Be aware that some computer algebra systems may normalize remainders from the Euclidean Algorithm which messes up the sign."

and was later reiterated in the Wikipedia article on polynomial gcd's,[6] from where we quote:

> "The pseudo-division has been introduced to allow a *variant*[7] of Euclid's algorithm for which all remainders belong to $\mathbb{Z}[x]$."

The last statement raises the question:

> "Why introduce a *variant* of Euclid's algorithm, when the Euclidean algorithm *itself* can be used for the same purpose?"

This is exactly what we did. We first introduced the new `sympy` function `rem_z(f, g, x)`, defined by

$$|\operatorname{LC}(R_{i-1})|^{\delta} \cdot R_{i-2} = q_{i-2} \cdot R_{i-1} + R_i. \tag{4}$$

The difference from `prem(f, g, x)` is that we now use the *absolute* value of the leading coefficient of the divisor.[8]

Then, based on the Pell-Gordon theorem [13] and on Theorem 1 of Section 3, we developed for `sympy` the module `subresultants_qq_zz.py`,[9] which includes various functions for computing Euclidean, modified Euclidean and (modified) subresultant prs's [3].[10] All sequences are computed either in $\mathbb{Q}[x]$, using the function `rem(f, g, x)` or in $\mathbb{Z}[x]$ using the function `rem_z(f, g, x)`.

Our module includes – among others – the functions

`euclid_q(f, g, x)`, `euclid_amv(f, g, x)`,
`subresultants_amv_q(f, g, x)` and `subresultants_amv(f, g, x)`,

[6] see `https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor#Pseudo-remainder_sequences`

[7] Our emphasis.

[8] We understand that many consider `rem_z(f, g, x)` a pseudo-remainders function as well.

[9] `https://github.com/sympy/sympy/blob/master/sympy/polys/subresultants_qq_zz.py`.

[10] Additional details on our module can be found in the Historical Note in Section 6.

which will be used in the sequel. *Caveat*: Module functions ending in "`_amv`" employ the function `rem_z(f, g, x)`.

The last two functions mentioned above are examined in detail in Sections 4 and 5, respectively. The subresultant prs's computed by `subresultants_amv_q (f, g, x)` and by `subresultants_amv(f, g, x)` are identical to those obtained by the `sympy`-core function `subresultants(f, g, x)`, which computes the remainder polynomials by employing the function `prem(f, g, x)`.

1.1. Outline of the paper. In Section 2 we examine in some detail the pseudo-remainders function `prem(f, g, x)`, which has been both a boon and a bane.

In Section 3 we present without proof Theorem 1, which is the theoretical basis of our two new methods for computing subresultant prs's.

In Section 4 we present `subresultants_amv_q(f, g, x)`, the method that computes remainder polynomials in $\mathbb{Q}[x]$ using the function `rem(f, g, x)`. This method is an implementation of equation (16) of Theorem 1 and, as expected, is somewhat slow given the inherently higher cost of rational operations.

In Section 5 we present `subresultants_amv(f, g, x)`, the method that computes remainder polynomials in $\mathbb{Z}[x]$ using the function `rem_z(f, g, x)`. This method is also an implementation of equation (16) of Theorem 1 but its performance is very competitive.

Finally, in Section 6 we present some empirical results and conclusions as well as a Historical Note.

2. On the pseudo-remainders function `prem(f, g, x)`.

The pseudo-remainders function `prem(f, g, x)` has been, and still is, used to compute in $\mathbb{Z}[x]$ the remainder polynomials of subresultant prs's. Its application has been both a boon and a bane.

A boon because – as detailed in Section – by employing `prem(f, g, x)` to compute in $\mathbb{Z}[x]$ the remainder polynomials it became possible, about 50 years ago, to develop `subresultants_cbt(f, g, x)` ([9], pp. 277–283). See also the Historical Note in Section 6.

A bane because – as detailed in Section 17 – employing `prem(f, g, x)` to compute in $\mathbb{Z}[x]$ the remainder polynomials of Euclidean and modified Euclidean prs's may lead to confusion with the signs and to conflict with Theorem 1 of Section 3.[11]

We examine both these cases separately.

[11] For the past forty years we have been trying, off and on, to straighten out this problem. Success came after our discovery of the Pell-Gordon theorem of 1917 [13].

2.1. `prem(f, g, x)`: the boon!

In this section we present `subresultants_cbt(f, g, x)`, the subresultant prs method developed by Collins, Brown and Traub, who introduced the currently used definition of *pseudo-remainders* [7], [8], [10], [11]. The function `subresultants_cbt(f, g, x)`, as presented in Algorithm 1, is equivalent to the `sympy`-core function `subresultants(f, g, x)`.

According to this method, the remainder polynomials are computed in $\mathbb{Z}[x]$ by first premultiplying the dividend times the leading coefficient of the divisor, according to formula (1).

However, repeated applications of (1) renders the coefficients of the remainder polynomials much bigger than the corresponding subresultants.

To reduce this coefficient growth, the algorithm cleverly reduces the resulting coefficients to subresultants by exactly dividing out a certain quantity β_i, defined by (6). We call this β_i the *Collins-Brown-Traub coefficients-reduction factor*, or simply (`cbt`) *coefficients-reduction factor*.

Therefore, to obtain the subresultant prs of f, g with `subresultants_cbt (f, g, x)` involves the following remainder sequence, [9]:

$$
\begin{aligned}
R_{-1} &= f,\\
R_0 &= g,\\
R_1 &= \frac{\texttt{prem}(R_{-1}, R_0, x)}{\beta_1},\\
&\vdots\\
R_i &= \frac{\texttt{prem}(R_{i-2}, R_{i-1}, x)}{\beta_i}, \quad etc,
\end{aligned} \tag{5}
$$

where R_i is exactly divided by the `cbt` coefficients-reduction factor β_i given by[12]

$$
\begin{aligned}
\psi_1 = -1,\ \beta_1 = (-1)^{\delta_1}, &\qquad i = 1,\\
\psi_i = \frac{(-\operatorname{LC}(R_{i-2}, x))^{\delta_{i-1}-1}}{\psi_{i-1}^{\delta_{i-1}-2}}, &\qquad i > 1,\\
\beta_i = -\operatorname{LC}(R_{i-2}, x) \cdot \psi_i^{\delta_i - 1}, &\qquad i > 1,
\end{aligned} \tag{6}
$$

and

$$\delta_i = \operatorname{degree}(R_{i-2}, x) - \operatorname{degree}(R_{i-1}, x) + 1, \qquad i \geqslant 1.$$

An algorithmic description of the above is presented in Algorithm 1.

Input: Two univariate polynomials $f, g \in \mathbb{Z}[x]$, with degree$(f, x) \geq$ degree(g, x), and the variable x.

Output: A list of polynomials $\in \mathbb{Z}[x]$, including f, g, constituting the subresultant prs of f, g. The polynomials in the sequence are computed by employing the pseudo-remainder function `prem(f, g, x)`.

`// make sure degrees are in order`

$[d_0, d_1] \leftarrow [\text{degree}(f, x), \text{degree}(g, x)]$;

if $d_0 = 0$ *and* $d_1 = 0$ **then return** $[f, g]$;

if $d_1 > d_0$ **then** $\{[d_0, d_1] \leftarrow [d_1, d_0]; [f, g] \leftarrow [g, f]\}$;

if $d_0 > 0$ *and* $d_1 = 0$ **then return** $[f, g]$;

`// initialize variables`

$[a_0, a_1, \psi, degdifP1] \leftarrow [f, g, -1, d_0 - d_1 + 1]$;

$a_2 \leftarrow \text{prem}(a_0, a_1, x)/(-1)^{degdifP1}$; `/* operations in` $\mathbb{Z}[x]$ `*/`

$subrList \leftarrow [a_0, a_1, a_2]$;

`// main loop`

while $d_2 > 0$ **do**

  $[a_0, a_1] \leftarrow [a_1, a_2]$;

  $\sigma_0 \leftarrow$ -LC(a_0, x); `/* leading coefficient of` a_0 `*/`

  $\psi \leftarrow \sigma_0^{degdifP1-1}/\psi^{degdifP1-2}$;

  $degdifP1 \leftarrow degree(a_0, x) - d_2 + 1$;

  $a_2 \leftarrow \text{prem}(a_0, a_1, x)/(\psi^{degdifP1-1} \cdot \sigma_0)$; `/* operations in` $\mathbb{Z}[x]$ `*/`

  $subrList \leftarrow \text{append}(subrList, a_2)$;

  $d_2 \leftarrow \text{degree}(a_2, x)$;

end

return $subrList$

Algorithm 1. The `subresultants_cbt(f, g, x)` algorithm. Computes remainder polynomials in $\mathbb{Z}[x]$ using the function `prem(f, g, x)` and implements equations (5) and (6)

2.2. `prem(f, g, x)`: the bane! In this section we present an example, showing the confusion with the signs and the conflict with Theorem 1 that may be caused when the function `prem(f, g, x)` is employed to compute in $\mathbb{Z}[x]$ the remainder polynomials of the Euclidean prs.

The following definition is needed:

Definition 2. *The* ***sign sequence*** *of a polynomial remainder sequence is the sequence of signs of the leading coefficients of its polynomials.*

[12] An explanation of the derivation of the formula for the factor β_i can be found elsewhere [12].

Example 1. *Consider the storied Knuth polynomials* $f = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$ *and* $g = 3x^6 + 5x^4 - 4x^2 - 9x + 21$, *whose incomplete prs has degrees* $8, 6, 4, 2, 1, 0$. *These are the same polynomials used in the Wikipedia article on polynomial gcd* `https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor`.

For incomplete prs's it is well known that the sign sequence of the Euclidean prs of f, g *may differ from the sign sequence of the subresultant prs of* f, g *[1].*

Indeed, in our case, the two sign sequences differ. To wit, the sign sequence of the Euclidean prs of f, g *is*

$$+, +, -, -, +, - \tag{7}$$

because, in $\mathbb{Q}[x]$, *application of the* `sympy` *function* `euclid_q(f, g, x)` *yields*

$$\begin{aligned} x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, 3x^6 + 5x^4 - 4x^2 - 9x + 21,\\ -5x^4/9 + x^2/9 - 1/3, -117x^2/25 - 9x + 441/25,\\ 233150x/19773 - 102500/6591, -1288744821/543589225, \end{aligned} \tag{8}$$

or, in $\mathbb{Z}[x]$, *application of the* `sympy` *function* `euclid_amv(f, g, x)` *yields*

$$\begin{aligned} x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, 3x^6 + 5x^4 - 4x^2 - 9x + 21,\\ -15x^4 + 3x^2 - 9, -65x^2 - 125x + 245, 9326x - 12300, -260708. \end{aligned} \tag{9}$$

Recall that the function `euclid_amv(f, g, x)` *employs the function* `rem_z(f, g, x)` *to compute the remainder polynomials.*

On the other hand, the sign sequence of the subresultant prs of f, g *is*

$$+, +, +, +, +, + \tag{10}$$

because application of the `sympy` *function* `subresultants(f, g, x)`[13] *yields*

$$\begin{aligned} x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, 3x^6 + 5x^4 - 4x^2 - 9x + 21,\\ 15x^4 - 3x^2 + 9, 65x^2 + 125x - 245, 9326x - 12300, 260708. \end{aligned} \tag{11}$$

Recall that `subresultants(f, g, x)` *employs the function* `prem(f, g, x)` *and correctly computes the subresultant prs of* f, g.

[13] Or any one of the various subresultants functions found in the module `https://github.com/sympy/sympy/blob/master/sympy/polys/subresultants_qq_zz.py`. See also the Historical Note in Section 6.

As already stated, according to Theorem 1 in Section 3, there is a one-to-one correspondence between the coefficients in (11), on one hand, and those in either (8) or (9), on the other. Stated another way, the sign sequence (10) of the subresultant prs is in one-to-one correspondence with – or, uniquely *related to – the sign sequence (7), of the Euclidean prs.*

Let us now employ the pseudo-remainders function `prem(f, g, x)` *to compute the remainder polynomials of the Euclidean prs of f, g. In this case we obtain a* variant[14] *of the Euclidean prs of f, g,*[15] *with sign sequence*

$$+, +, -, +, +, +, \tag{12}$$

which is obviously different from (7). Therefore, confusion with the signs arises; moreover, we have a conflict with Theorem 1 since the sign sequence (12) does not *correspond to the one in (10).*

Obviously, by Theorem 1, the variant of the Euclidean prs of f, g with sign sequence (12) uniquely *corresponds to a* variant *of the subresultant prs of f, g with a sign sequence different than (10).*

As demonstrated in the above example, with the help of the function `euclid_amv(f, g, x)` – which employs the function `rem_z(f, g, x)` – we were able to correctly compute the Euclidean prs (9) in $\mathbb{Z}[x]$.

However, the situation now gets more complicated because `rem_z(f, g, x)` *cannot* be used in place of `prem(f, g, x)` in the function `subresultants_cbt (f, g, x)` to correctly compute the subresultant prs of two polynomials.

The theorem in the next section is our "Deus ex Machina."

3. Theoretical background of our subresultant prs methods.

In this section we present Theorem 1, which is the theoretical basis of our new methods for computing subresultant prs's either in $\mathbb{Q}[x]$ or in $\mathbb{Z}[x]$.

Our theorem is an extension and generalization of the Pell-Gordon theorem of 1917 [13]. Due to technical details, its proof is a difficult read, and, since it can be found elsewhere [6], it is omitted here.

Theorem 1. *Let*

$$\begin{aligned} f &= a_0 x^n + a_1 x^{n-1} + \cdots + a_n, \\ g &= b_0 x^n + b_1 x^{n-1} + \cdots + b_n \end{aligned} \tag{13}$$

[14] Same coefficients in absolute value, but different signs.

[15] See also `https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor`.

be two polynomials of degree n *and* $n - p_0$*, respectively, with* $b_0 = b_1 = \ldots = b_{p_0-1} = 0$*,* $b_{p_0} \neq 0$*,* $p_0 \geq 0$*. Moreover, for* $i = 1, 2, \ldots,$ *let*

$$\begin{aligned} R^{(i)} &= r_0^{(i)} x^{m_i} + r_1^{(i)} x^{m_i - 1} + \cdots + r_{m_i}^{(i)}, \\ R^{E(i)} &= r_0^{E(i)} x^{m_i} + r_1^{E(i)} x^{m_i - 1} + \cdots + r_{m_i}^{E(i)}, \end{aligned} \tag{14}$$

be the i*-th modified Euclidean and Euclidean remainders, respectively, of* f, g*, with* $R^{(i)}$ *and* $R^{E(i)}$ *both of degree* $m_i - p_i + 1$*, where* $(m_i + 1)$ *is the degree of the preceding remainder and*

$$r_0^{(i)} = r_0^{E(i)} = \ldots = r_{p_i-2}^{(i)} = r_{p_i-2}^{E(i)} = 0, \ \varrho_i = r_{p_i-1}^{(i)} \neq 0, \ \sigma_i = r_{p_i-1}^{E(i)} \neq 0.$$

Then for $k = 0, 1, \ldots, m_i$ *the coefficients* $r_k^{(i)}$ *and* $r_k^{E(i)}$ *in (14) are given by*

$$r_k^{(i)} = \frac{(-1)^{\varphi_i}}{\varrho_{i-1}^{p_{i-1}+1} \varrho_{i-2}^{p_{i-2}+p_{i-1}} \cdots \varrho_0^{p_0+p_1}} \times \frac{\mathrm{Det}_{i,k}(f,g)}{a_0^{p_0}}, \tag{15}$$

$$r_k^{E(i)} = \frac{(-1)^{\psi_i}}{\sigma_{i-1}^{p_{i-1}+1} \sigma_{i-2}^{p_{i-2}+p_{i-1}} \cdots \sigma_0^{p_0+p_1}} \times \frac{\mathrm{Det}_{i,k}(f,g)}{a_0^{p_0}}, \tag{16}$$

where $\varrho_0 = \sigma_0 = b_{p_0}$*,*

$$\varphi_i = \lfloor (s_{i-1} + 1)/2 \rfloor, \tag{17}$$

$$s_{i-1} = \textit{the number of odd integers in the list } \{p_0, p_1, \ldots, p_{i-1}\}, \tag{18}$$

$$\psi_i = i + \varphi_i + p_1 + p_3 + p_5 + .. + p_{2\lfloor i/2 \rfloor - 1}, \ \textit{with } p_{-1} = 0, \tag{19}$$

$$\mathrm{Det}_{i,k}(f,g) = \begin{vmatrix} a_0 & a_1 & a_2 & \cdots & \cdot & \cdot & \cdots & a_{2v_{i-1}} & a_{2v_{i-1}+k+1} \\ 0 & a_0 & a_1 & \cdots & \cdot & \cdot & \cdots & a_{2v_{i-1}-1} & a_{2v_{i-1}+k} \\ \vdots & & & \ddots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & a_0 & a_1 & \cdots & a_{v_{i-1}} & a_{v_{i-1}+k+1} \\ b_0 & b_1 & b_2 & \cdots & \cdot & \cdot & \cdots & b_{2v_{i-1}} & b_{2v_{i-1}+k+1} \\ 0 & b_0 & b_1 & \cdots & \cdot & \cdot & \cdots & b_{2v_{i-1}-1} & b_{2v_{i-1}+k} \\ \vdots & & & \ddots & & & \ddots & & \vdots \\ 0 & 0 & 0 & \cdots & b_0 & b_1 & \cdots & b_{v_{i-1}} & b_{v_{i-1}+k+1} \end{vmatrix}, \tag{20}$$

and

$$v_{i-1} = p_0 + p_1 + \cdots + p_{i-1}. \tag{21}$$

As mentioned in Section 2, if the Euclidean prs of $f, g \in \mathbb{Z}[x]$ is incomplete, then its sign sequence is not necessarily identical to the sign sequence of the subresultant prs of f, g because, in general, from (16) we have

$$\operatorname{sgn}\left(r_k^{E(i)}\right) \neq \operatorname{sgn}\left(\operatorname{Det}_{i,k}(f,g)\right).$$

Based on our earlier work on modified subresultants [3], we use equation (16) to compute subresultant prs's; that is, we use (16) to compute the sign of the determinant given the sign of the corresponding coefficient of the Euclidean prs; and vice-versa.

We compute, for each remainder, the exact sign of the first fraction in (16), by multiplying times the absolute value of its denominator both sides of equation (16). That is we have

$$\left|\sigma_{i-1}^{p_{i-1}+1}\sigma_{i-2}^{p_{i-2}+p_{i-1}}\cdots\sigma_0^{p_0+p_1}\right| \cdot r_k^{E(i)} = \pi_i \cdot \operatorname{Det}_{i,k}(f,g). \tag{22}$$

where

$$\pi_i = (-1)^{\psi_i} \cdot \left(\operatorname{sgn}(\sigma_{i-1}^{p_{i-1}+1})\operatorname{sgn}(\sigma_{i-2}^{p_{i-2}+p_{i-1}})\cdots\operatorname{sgn}(\sigma_0^{p_0+p_1})\right). \tag{23}$$

Obviously, from equation (22) it follows that

$$\operatorname{sgn}\left(r_k^{E(i)}\right) = \pi_i \cdot \operatorname{sgn}\left(\operatorname{Det}_{i,k}(f,g)\right), \tag{24}$$

and from equation (24) we obtain

$$\operatorname{sgn}\left(\operatorname{Det}_{i,k}(f,g)\right) = \begin{cases} \operatorname{sgn}(r_k^{E(i)}) & \text{if } \pi_i > 0, \\ -\operatorname{sgn}\left(r_k^{E(i)}\right) & \text{if } \pi_i < 0. \end{cases} \tag{25}$$

The above procedure is easily programmed. The only critical point is to effectively compute the absolute value in (22). This value is not computed anew for each remainder $R^{E(i)}$; instead, a multiplication factor, μ, is being updated as new leading coefficients are included in (22). So, if the current multiplication factor is

$$\mu_i = \left|\sigma_{i-1}^{p_{i-1}+1}\sigma_{i-2}^{p_{i-2}+p_{i-1}}\cdots\sigma_1^{p_1+p_2}\sigma_0^{p_0+p_1}\right|,$$

then the updated factor for the next remainder $R^{E(i+1)}$ is

$$\mu_{i+1} = \left|\sigma_i^{p_i+1}\sigma_{i-1}^{p_i-1} \cdot \mu_i\right|,$$

which means that

$$\mu_{i+1} = \left|\sigma_i^{p_i+1}\sigma_{i-1}^{p_{i-1}+p_i} \cdots \sigma_1^{p_1+p_2}\sigma_0^{p_0+p_1}\right|.$$

4. Subresultant prs obtained by polynomial divisions in $\mathbb{Q}[x]$. In Algorithm 2 we present `subresultants_amv_q(f, g, x)`, our first method, which is an implementation of equation (16) of Theorem 1. Our method computes remainder polynomials in $\mathbb{Q}[x]$ using the function `rem(f, g, x)`. Because the cost of performing rational operations is greater that the cost of performing integer operations, as can be seen from Fig. 1 in Section 6, our method is too slow to be of practical use; it is mainly of theoretical interest.

Example 2. *Consider the same polynomials* $f = x^8+x^6-3x^4-3x^3+8x^2+2x-5$ *and* $g = 3x^6+5x^4-4x^2-9x+21$, *used in Example 1, whose incomplete polynomial remainder sequence (prs) has degrees* $8, 6, 4, 2, 1, 0$.

We know that the subresultant prs of f, g *in* $\mathbb{Z}[x]$ *is*

$$\begin{aligned}(26)\quad & x^8+x^6-3x^4-3x^3+8x^2+2x-5, 3x^6+5x^4-4x^2-9x+21, \\ & 15x^4-3x^2+9, 65x^2+125x-245, 9326x-12300, 260708,\end{aligned}$$

where the coefficients of the polynomials in the second row of (26) are all determinants (subresultants) of appropriately selected sub-matrices of `sylvester1(f, g, x)`, *of dimensions* 14×14.

Below we use Algorithm 2 to compute the polynomials in the second row of (26).

Before entering the main loop of the algorithm, the variables i, s, pOddIndexSum are set to zero, whereas μ *is set to 1; the rest of the variables are initialized as follows:*

$$\begin{gathered}a_0 = x^8+x^6-3x^4-3x^3+8x^2+2x-5, a_1 = 3x^6+5x^4-4x^2-9x+2, \\ \sigma_1 = 3, d_0 = 8, d_1 = 6, d_2 = 6, p_0 = 2, s = 0, \varphi = 0, \\ subresL = [x^8+x^6-3x^4-3x^3+8x^2+2x-5, 3x^6+5x^4-4x^2-9x+21].\end{gathered}$$

Inside the main loop the variables are updated as shown below.

Input: Two univariate polynomials $f, g \in \mathbb{Z}[x]$, with degree$(f, x) \geq$ degree(g, x), and the variable x.

Output: A list of polynomials $\in \mathbb{Z}[x]$, including f, g, constituting the subresultant prs of f, g.

```
// make sure degrees are in order; insert lines 1-4 from Algorithm 1.
[d_0, d_1] ← [degree(f, x), degree(g, x)];
// initialize variables
subrList ← [f, g];              /* subresultant prs list, to be returned at the end */
[i, s] ← [0, 0];                /* counters for remainders and odd elements */
pOddIndexSum ← 0;               /* holds the sum p_1 + p_3 + ··· */
[a_0, a_1] ← [f, g];
σ_1 ← LC(a_1, x);               /* leading coefficient of a_1 */
p_0 ← d_0 − d_1;
if mod(p_0, 2) = 1 then s ← s + 1;
φ = ⌊(s + 1)/2⌋;
[μ, d_2] ← [1, d_1]
// main loop
while d_2 > 0 do
    i ← i + 1;
    a_2 ← rem(a_0, a_1, x);                         /* operations in ℚ[x] */
    if i = 1 then
        σ_2 ← LC(a_2, x)
    else
        σ_3 ← LC(a_2, x);
        [σ_1, σ_2] ← [σ_2, σ_3]
    end
    d_2 ← degree(a_2, x);
    p_1 ← d_1 − d_2;
    ψ ← i + φ + pOddIndexSum;
    // initial value of μ
    μ ← σ_1^(p_0+1) × μ;
    // evaluate the sign of the first fraction in (16)
    num ← (−1)^ψ;                                   /* sign of the numerator */
    den ← sgn(μ);                                   /* sign of the denominator */
    // the sign of the determinant in (16) depends on sgn(num · den) ≠ 0
    if sgn(num · den) > 0 then
        subrList ← append(subrList, a_2 × |μ|);     /* a_2 × |μ| ∈ ℤ[x] */
    else
        subrList ← append(subrList, −a_2 × |μ|);    /* −a_2 × |μ| ∈ ℤ[x] */
    end
    // bring into μ the missing power σ_1^(p_1−1) if there was degree gap
    if p_1 − 1 > 0 then μ ← σ_1^(p_1−1) × μ;
    // update variables
    [a_0, a_1, d_0, d_1, p_0] ← [a_1, a_2, d_1, d_2, p_1];
    if mod(p_0, 2) = 1 then s ← s + 1;
    φ ← ⌊(s + 1)/2⌋;
    if mod(i, 2) = 1 then
        pOddIndexSum ← pOddIndexSum + p_0;          /* p_i has odd index */
end
return subrList
```

Algorithm 2. The `subresultants_amv_q(f, g, x)` algorithm. Computes remainder polynomials in $\mathbb{Q}[x]$ using the function `rem(f, g, x)` and implements equation (16)

- *for the first remainder ($i = 1$) we have:*

$$a_2 = -5x^4/9 + x^2/9 - 1/3, \sigma_1 = 3, \sigma_2 = -5/9, d_2 = 4, p_1 = 2, \psi = 1, \\ \mu = 27, \mathrm{sgn}(num \cdot den) = -1, -a_2 \times |\mu| = 15x^4 - 3x^2 + 9,$$

where the last entry above is appended to $subresL$. Then, since there was a degree gap, we "correct" the value of μ to $\mu = 81$. For the next iteration the other updated variables are:

$$a_0 = 3x^6 + 5x^4 - 4x^2 - 9x + 21, a_1 = -5x^4/9 + x^2/9 - 1/3, \\ d_0 = 6, d_1 = 4, p_0 = 2, s = 0, \varphi = 0, pOddIndexSum = 2.$$

- *for the second remainder ($i = 2$) we have:*

$$a_2 = -117x^2/25 - 9x + 441/25, \sigma_1 = -5/9, \sigma_2 = -117/25, \\ d_2 = 2, p_1 = 2, \psi = 4, \mu = -125/9, \\ \mathrm{sgn}(num \cdot den) = -1, -a_2 \times |\mu| = 65x^2 + 125x - 245,$$

where the last entry above is appended to $subresL$. Then, since there was a degree gap, we "correct" the value of μ to $\mu = 625/81$. For the next iteration the other updated variables are:

$$a_0 = -5x^4/9 + x^2/9 - 1/3, a_1 = -117x^2/25 - 9x + 441/25, \\ d_0 = 4, d_1 = 2, p_0 = 2, s = 0, \varphi = 0, pOddIndexSum = 2.$$

- *for the third remainder ($i = 3$) we have:*

$$a_2 = 233150x/19773 - 102500/6591, \sigma_1 = -117/25, \sigma_2 = 233150/19773, \\ d_2 = 1, p_1 = 1, \psi = 5, \mu = -19773/25, \\ \mathrm{sgn}(num \cdot den) = 1, a_2 \times |\mu| = 9326x - 12300,$$

where the last entry above is appended to $subresL$. Since there was no degree gap the value of μ stays the same, whereas for the next iteration the other updated variables are:

$$a_0 = -117x^2/25 - 9x + 441/25, a_1 = 233150x/19773 - 102500/6591, \\ d_0 = 2, d_1 = 1, p_0 = 1, s = 1, \varphi = 1, pOddIndexSum = 3.$$

- *for the fourth remainder ($i = 4$) we have:*

$$a_2 = -1288744821/543589225,$$
$$\sigma_1 = 233150/19773, \sigma_2 = -1288744821/543589225,$$
$$d_2 = 0, p_1 = 1, \psi = 8, \mu = -2174356900/19773,$$
$$\operatorname{sgn}(num \cdot den) = -1, -a_2 \times |\mu| = 260708,$$

where the last entry above is appended to $subresL$. Since there was no degree gap the value of μ stays the same, whereas for the next iteration the other updated variables are:

$$a_0 = 233150x/19773 - 102500/6591, a_1 = -1288744821/543589225,$$
$$d_0 = 1, d_1 = 0, p_0 = 1, s = 2, \varphi = 1, pOddIndexSum = 3.$$

Since $d_2 = 0$, the algorithm terminates, having computed the subresultant prs of f, g as shown in (26).

5. Subresultant prs obtained by polynomial divisions in $\mathbb{Z}[x]$. In Algorithm 3 we present `subresultants_amv(f, g, x)`, our second method, which is also an implementation of equation (16) of Theorem 1 and its performance is quite competitive.

Here is how `subresultants_amv(f, g, x)` works:

- To perform polynomial divisions in $\mathbb{Z}[x]$ it uses the function `rem_z(f, g, x)`, defined by equation (4).
- To reduce the size of the coefficients of the polynomial obtained by `rem_z(f, g, x)` it exactly divides the ith remainder by the *absolute* value of the `cbt` coefficients-reduction factor β_i defined by (6). After this operation, the correct *absolute* value of the determinant has been computed.
- To compute the correct sign of the determinant it implements equation (16), as discussed in Section 3.

The above are incorporated into Algorithm 3.

6. Empirical results and conclusions. This paper continues and brings to a successful ending earlier efforts, [4], to develop subresultant prs algorithms, where the remainder polynomials are computed in such a way that it can be “safely” employed for computing Euclidean and modified Euclidean prs’s.

Input: Two univariate polynomials $f, g \in \mathbb{Z}[x]$, with degree$(f,x) \geq$ degree(g,x), and the variable x.

Output: A list of polynomials $\in \mathbb{Z}[x]$, including f, g, constituting the subresultant prs of f, g.

`// make sure degrees are in order; insert lines 1-4 from Algorithm 1.`
$[d_0, d_1] \leftarrow [\text{degree}(f,x), \text{degree}(g,x)]$;
`// initialize variables`
$[a_0, a_1, c, degdifP1] \leftarrow [f, g, -1, d_0 - d_1 + 1]$;
$subrList \leftarrow [f, g]$; `/* the subresultant prs list, to be returned at the end */`
$\sigma_1 \leftarrow \text{LC}(a_1, x)$; `/* leading coefficient of` a_1 `*/`
$[i, s] \leftarrow [0, 0]$; `/* counters for remainders and odd elements */`
$pOddIndexSum \leftarrow 0$; `/* holds the sum` $p_1 + p_3 + \cdots$ `*/`
$p_0 \leftarrow degdifP1 - 1$;
if $mod(p_0, 2) = 1$ **then** $s \leftarrow s + 1$;
$\varphi = \lfloor (s+1)/2 \rfloor$;
$i \leftarrow i + 1$;
$a_2 \leftarrow \text{rem\_}z(a_0, a_1, x)/|c^{degdifP1}|$; `/* 1st remainder in` $\mathbb{Z}[x]$ `*/`
$\sigma_2 \leftarrow \text{LC}(a_2, x)$;
$d_2 \leftarrow \text{degree}(a_2, x)$;
$p_1 \leftarrow d_1 - d_2$;
$sgnDen \leftarrow \text{sgn}(\sigma_1^{p_0+1})$; `/* sign of the denominator */`
$\psi \leftarrow i + \varphi + pOddIndexSum$;
`// evaluate the sign of the first fraction in (16) and the sign of the determinant`
$[num, den] \leftarrow [(-1)^{\psi}, sgnDen]$;
if $\text{sgn}(num \cdot den) > 0$ **then**
  $subrList \leftarrow \text{append}(subrList, a_2)$; `/*` $a_2 \in \mathbb{Z}[x]$ `*/`
else
  $subrList \leftarrow \text{append}(subrList, -a_2)$; `/*` $-a_2 \in \mathbb{Z}[x]$ `*/`
end
`// bring into` $sgnDen$ `the missing sign` $\text{sgn}(\sigma_1^{p_1-1})$ `if there was a degree gap`
if $p_1 - 1 > 0$ **then** $sgnDen \leftarrow sgnDen \cdot \text{sgn}(\sigma_1^{p_1-1})$;

Algorithm 3. The `subresultants_amv(f, g, x)` algorithm. Computes remainder polynomials in $\mathbb{Z}[x]$ using the function `rem_z(f, g, x)` and implements equations (6) and (16)

We presented `subresultants_amv_q(f, g, x)` and `subresultants_amv(f, g, x)`, two new methods for computing the subresultant prs of two polynomials $f, g \in \mathbb{Z}[x]$. The two methods constitute two different implementations of Theorem 1, whereby the remainder polynomials are computed either in $\mathbb{Q}[x]$ or in $\mathbb{Z}[x]$ by employing respectively the function `rem(f, g, x)` or `rem_z(f, g, x)`.

Moreover, Euclidean and modified Euclidean prs's are obtained in full

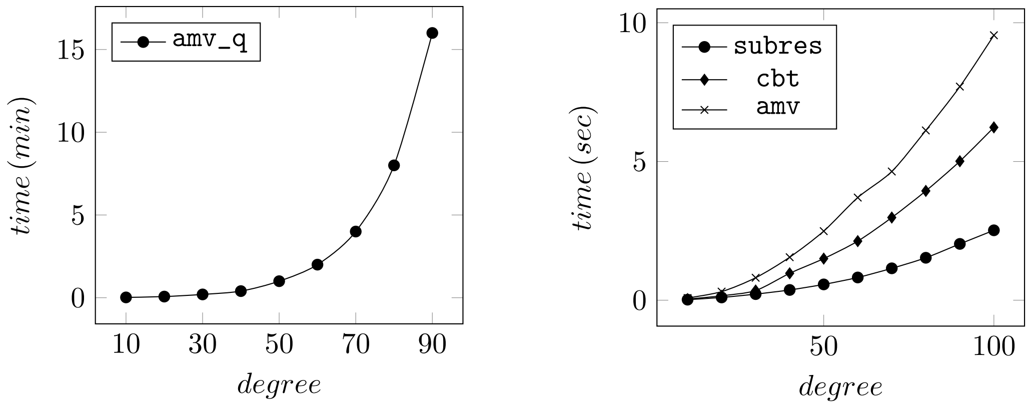
```
// main loop
while d_2 > 0 do
    φ ← ⌊(s + 1)/2⌋;
    if mod(i, 2) = 1 then pOddIndexSum ← pOddIndexSum + p_1;
    ;                                              /* p_i has odd index */
    [a_0, a_1, d_0, d_1, i, p_0] ← [a_1, a_2, d_1, d_2, i + 1, p_1];
    σ_0 ← -LC(a_0);
    c ← σ_0^(degdifP1−1) / c^(degdifP1−2);
    degdifP1 ← degree(a_0, x) − d_2 + 1;
    a_2 ← rem_z(a_0, a_1, x)/|c^(degdifP1−1) · σ_0|;   /* operations in ℤ[x] */
    σ_3 ← LC(a_2, x);
    d_2 ← degree(a_2, x);
    p_1 ← d_1 − d_2;
    ψ ← i + φ + pOddIndexSum;
    [σ_1, σ_2] ← [σ_2, σ_3];
    sgnDen ← sgnDen · sgn(σ_1^(p_0+1));             /* sign of the denominator */
    // evaluate the sign of the first fraction in (16) and the sign of the
       determinant
    [num, den] ← [(−1)^ψ, sgnDen];
    if sgn(num · den) > 0 then
        subrList ← append(subrList, a_2);                 /* a_2 ∈ ℤ[x] */
    else
        subrList ← append(subrList, −a_2);               /* −a_2 ∈ ℤ[x] */
    end
    if mod(p_1, 2) = 1 then s ← s + 1;
    // bring into sgnDen the missing sign sgn(σ_1^(p_1−1)) if there was degree gap
    if p_1 − 1 > 0 then sgnDen ← sgnDen · sgn(σ_1^(p_1−1));
end
return subrList
```

Algorithm 3. The `subresultants_amv(f, g, x)` algorithm *(continued)*

agreement with Theorem 1, when the remainder polynomials are computed either by the function `rem(f, g, x)` or by the function `rem_z(f, g, x)`.

As indicated by graph (1a) of Fig. 1, the first method is inherently slow due to rational arithmetic. However, graph (1b) of Fig. 1 indicates that the performance of our second method is quite competitive.

All methods were implemented in `sympy` and run through Spyder. The computer was a mac-mini with 2 GHz Intel Core 2 Duo and with 2 GB 667 MHz DDR2 SDRAM running Version 10.7.5 Mac OS X. They were tested on pairs of random, dense polynomials with single digit coefficients of degrees $d, d-2$, where



(a) Polynomial divisions in $\mathbb{Q}[x]$. (b) Polynomial divisions in $\mathbb{Z}[x]$.

Fig. 1. The time is in minutes in graph (1a) and in seconds in graph (1b). In graph (1a) the time for the pair of polynomials of degrees 100 and 98 was more than half an hour. The function `subresultants(f, g, x)` belongs to the `sympy`-core functions.

$d = 10, 20, \ldots, 100$.

As was expected, `subresultants_amv(f, g, x)` is somewhat slower than `subresultants_cbt(f, g, x)` because of the extra work it does to compute the correct signs of the coefficients. However, timing considerations were not our concern.

Historical Note: In the statement of the Pell-Gordon theorem of 1917 [13] we encounter the first algorithm in the History of Mathematics for the computation of (modified) subresultant prs's *without* determinant evaluations!

The Pell-Gordon algorithm, which employs the function `rem(f, g, x)` for the computation of the remainder polynomials, had been dormant for almost a century, but is now included as function `modified_subresultants_pg(f, g, x)` in our `sympy` module `subresultants_qq_zz.py`[9].

Included in our module are also the functions `subresultants_pg(f, g, x)`, `euclid_pg(f, g, x)` and `sturm_pg(f, g, x)`, all based on the Pell-Gordon theorem and using the function `rem(f, g, x)`, as well as the functions

`subresultants_amv(f, g, x)`,
`modified_subresultants_amv(f, g, x)`,
`euclid_amv(f, g, x)` and `sturm_amv(f, g, x)`,

all based on Theorem 1 and using the function `rem_z(f, g, x)`.

An exception is the function `subresultants_amv_q(f, g, x)`, which employs the function `rem(f, g, x)`, despite the fact that it implements Theorem 1. We decided to include the function `subresultants_amv_q(f, g, x)` in our module in order to show that both theorems mentioned above can be implemented using either the function `rem(f, g, x)` or the function `rem_z(f, g, x)`. For clearly historical reasons – since the `cbt` coefficients-reduction factor β_i was not available in 1917 – we have implemented the Pell-Gordon theorem with the function `rem(f, g, x)` and Theorem 1 with the function `rem_z(f, g, x)`.

Three functions in our module are independent of both theorems mentioned above; namely, `subresultants_rem(f, g, x)`, `subresultants_vv(f, g, x)` and `subresultants_vv_2(f, g, x)`. All three functions evaluate one determinant per remainder polynomial; this is the determinant of an appropriately selected sub-matrix of `sylvester1(f, g, x)`, Sylvester's matrix of 1840.[16]

To compute the remainder polynomials the function `subresultants_rem (f, g, x)` employs `rem(f, g, x)`.[17] By contrast, `subresultants_vv(f, g, x)` and `subresultants_vv_2(f, g, x)` implement Van Vleck's ideas of 1900, [2], and compute the remainder polynomials by triangularizing `sylvester2(f, g, x)`, Sylvester's matrix of 1853.

It goes without saying that our module includes the function `sylvester(f, g, x, method=0)` to compute *either* Sylvester matrix.

REFERENCES

[1] Akritas A. G. A Simple Proof of the Validity of the Reduced PRS Algorithm. *Computing*, **38** (1987), 369–372.

[2] Akritas A. G., G. I. Malaschonok, P. S. Vigklas. On a Theorem by Van Vleck Regarding Sturm Sequences. *Serdica Journal of Computing*, **7** (2013), No 4, 101–134.

[16]The functions `euclid_q(f, g, x` and `sturm_q(f, g, x)` in our module are not only independent of both theorems mentioned above, but neither do they evaluate determinants.

[17]Elsewhere, [4], we present the function `subresultants_prem2(f, g, x)`, which employs `prem2(f, g, x)` to compute the remainder polynomials; the function `prem2(f, g, x)` is the same as `rem_z(f, g, x)`. We decided against including `subresultants_rem_z(f, g, x)` in our module because then we would have two options for converting the remainder coefficients into subresultants: either to perform one determinant evaluation per division, as is currently the case with `subresultants_rem(f, g, x)`, or to divide them by the `cbt` coefficients-reduction factor β_i.

[3] Akritas A. G., G. I. Malaschonok, P. S. Vigklas. Sturm Sequences and Modified Subresultant Polynomial Remainder Sequences. *Serdica Journal of Computing*, **8** (2014), No 1, 29–46.

[4] Akritas A. G. Three New Methods for Computing Subresultant Polynomial Remainder Sequences (PRS's). *Serdica Journal of Computing*, **9** (2015), No 1, 1–26.

[5] Akritas A. G., G. I. Malaschonok, P. S. Vigklas. On the Remainders Obtained in Finding the Greatest Common Divisor of Two Polynomials. *Serdica Journal of Computing*, **9** (2015), No 2, 123–138.

[6] Akritas A. G., G. I. Malaschonok, P. S. Vigklas. A Basic Result on the Theory of Subresultants. *Serdica Journal of Computing*, **10** (2016), No 1, 31–48.

[7] Brown W. S. The subresultant PRS Algorithm. *ACM Transactions on Mathematical Software*, **4** (1978), No 3, 237–249.

[8] Brown W. S., J. F. Traub. On Euclid's Algorithm and the Theory of Subresultants. *Journal of the Association for Computing Machinery*, **18** (1971), 505–514.

[9] Cohen J. E. Computer Algebra and Symbolic Computation—Mathematical Methods. A. K. Peters, Massachusetts, 2003.

[10] Collins G. E. Polynomial Remainder Sequences and Determinants. *American Mathematical Monthly*, **73** (1966), No 7, 708–712.

[11] Collins G. E. Subresultants and Reduced Polynomial Remainder Sequences. *Journal of the Association for Computing Machinery*, **14** (1967), 128–142.

[12] Knuth D. E. The Art of Computer Programming, Vol. 2. Addison-Wesley, Massachusetts, 1981.

[13] Pell A. J., R. L. Gordon. The Modified Remainders Obtained in Finding the Highest Common Factor of Two Polynomials. *Annals of Mathematics*, Second Series, **18** (1917), No 4, 188–193.

[14] Sylvester J. J. A method of determining by mere inspection the derivatives from two equations of any degree. *Philosophical Magazine*, **16** (1840), 132–135.

[15] Sylvester J. J. On the Theory of Syzygetic Relations of Two Rational Integral Functions, Comprising an Application to the Theory of Sturm's Functions, and that of the Greatest Algebraical Common Measure. *Philosophical Transactions*, **143** (1853), 407–548.

Alkiviadis G. Akritas
Department of Electrical and Computer Engineering
University of Thessaly
GR-38221, Volos, Greece
e-mail: akritas@uth.gr

Gennadi I. Malaschonok
Laboratory for Algebraic Computations
Tambov State University
Internatsionalnaya, 33
RU-392000 Tambov, Russia
e-mail: malaschonok@ya.ru

Panagiotis S. Vigklas
Department of Electrical and Computer Engineering
University of Thessaly
GR-38221, Volos, Greece
e-mail: pviglas@uth.gr

Received May 19, 2016
Final Accepted April 28, 2017