Three New Methods for Computing Subresultant Polynomial Remainder Sequences (PRS's)

Alkiviadis G. Akritas
University of Thessaly
Department of Electrical and Computer Engineering
GR-38221, Volos, Greece

August 4, 2014

To the memory of Anna Johnson Pell[1] *and R. L. Gordon, for their inspiring Theorem of 1917!*

Abstract

Given the polynomials $f, g \in \mathbb{Z}[x]$ of degrees n, m, respectively, with $n > m$, three new, and easy to understand methods — along with the more efficient variants of the last two of them — are presented for the computation of their subresultant polynomial remainder sequence (prs).

All three methods evaluate a single determinant (`subresultant`) of an appropriate submatrix of `sylvester1`, Sylvester's widely known and used matrix of 1840 of dimension $(m+n) \times (m+n)$, in order to compute the correct sign of each polynomial in the sequence and — except for the second method — to force its coefficients to become subresultants.

Of interest is the fact that only the first method uses pseudo remainders. The second method uses regular remainders and performs operations in $\mathbb{Q}[x]$, whereas the third one triangularizes `sylvester2`, Sylvester's little known and hardly ever used matrix of 1853 of dimension $2n \times 2n$.

All methods mentioned in this paper (along with their supporting functions) have been implemented in `Sympy` and can be downloaded from the link `http://inf-server.inf.uth.gr/~akritas/publications/subresultants.py`.

Contents

[1] See the link `http://en.wikipedia.org/wiki/Anna_Johnson_Pell_Wheeler` for her biography.

1 Introduction

Given the polynomials $f, g \in \mathbb{Z}[x]$, of degrees n, m respectively, with $n \geq m$, various methods exist to compute their subresultant polynomial remainder sequence (prs). Some of these methods use only divisions, while others only evaluate minors of certain matrices and avoid divisions [16], [21]. However, as indicated by all authors, the optimal choice depends dramatically on the concrete polynomial pair under consideration and typically requires some experimentation.

Motivated by our work on the Pell-Gordon Theorem of 1917, [7], [8], [24], we increased by three the number of available methods to compute the subresultant prs. As we will see in the sequel, the new methods that we present use both polynomial divisions and one determinant evaluation per remainder. Assuming that a fast (probabilistic) algorithm is available to compute the determinant, these methods can be quite efficient.

We begin by first describing Sylvester's two matrices.

Sylvester's matrix `sylvester1` was discovered in 1840 [25] and its dimensions are $(n+m) \times (n+m)$; it consists of two groups of rows, the first one with m rows and the second one with n. Concatenation of the two groups yields matrix `sylvester1`.

In the first row of the first group (of m rows) are the coefficients of $f(x)$ with $m-1$ trailing zeros. The second row in this group differs from the first one in that its elements have been rotated to the right by one. A total of $m-1$ rotations are needed to construct the first group of rows.

In the first row of the second group (of n rows) are the coefficients of $g(x)$ with $n-1$ trailing zeros. The second row in this group differs from the first one in that its elements have been rotated to the right by one. A total of $n-1$ rotations are needed to construct the second group of rows.

Sylvester's matrix, `sylvester2` was discovered in 1853, its dimensions are $2n \times 2n$ and it consists of n pairs of rows [26]. In the first row of the first pair are the coefficients of $f(x)$ whereas in the second row of the first pair are the coefficients of $g(x)$; $n-m$ zeros have been prepended to $g(x)$ to also make it of degree n. Both rows in the first pair have $2n-(n+1)$ trailing zeros and both rows of the last pair have $2n-(n+1)$ leading zeros. The second pair of rows differs from the first one in that the elements of both rows have been rotated to the right by one. A total of $2n-(n+1)$ rotations are needed to construct Sylvester's matrix.

In `Xcas/Giac` Sylvester's `sylvester1` matrix is given by the built-in function `sylvester`, whereas Sylvester's `sylvester2` matrix is given by our own function `sylvester2`. In `Sympy` we have written the function `sylvester`[2] which returns either matrix depending on the last optional argument; by default matrix `sylvester1` is returned.

[2] All `Sympy` functions mentioned in this paper can be downloaded from the link `http://inf-server.inf.uth.gr/~akritas/publications/subresultants.py`.

Example 1 Take $f(x) = ax^3 + bx^2 + cx + d$ and $g(x) = 3ax^2 + 2bx + c$. Then, S_1, their `sylvester1` matrix, is[3]

```
Python] from sympy import *
Python] a, b, c, d, x = var('a b c d x')
Python] S1 = sylvester(a*x**3 + b*x**2 + c*x + d, 3*a*x**2 + 2*b*x + c, x, 1)
Python] S1
```

$$\begin{pmatrix} a & b & c & d & 0 \\ 0 & a & b & c & d \\ 3a & 2b & c & 0 & 0 \\ 0 & 3a & 2b & c & 0 \\ 0 & 0 & 3a & 2b & c \end{pmatrix},$$

whereas S_2, their `sylvester2` matrix, is

```
Python] S2 = sylvester(a*x**3 + b*x**2 + c*x + d, 3*a*x**2 + 2*b*x + c, x, 2)
Python] S2
```

$$\begin{pmatrix} a & b & c & d & 0 & 0 \\ 0 & 3a & 2b & c & 0 & 0 \\ 0 & a & b & c & d & 0 \\ 0 & 0 & 3a & 2b & c & 0 \\ 0 & 0 & a & b & c & d \\ 0 & 0 & 0 & 3a & 2b & c \end{pmatrix}.$$

.

For the sequences of polynomial remainders examined in this paper the following definition is needed:

Definition 1 *The* ***sign sequence*** *of a polynomial remainder sequence (prs) is the sequence of signs of the leading coefficients of its polynomials.*

Given $f(x), g(x) \in \mathbb{Z}[x]$ of degrees $deg(f) = n$ and $deg(g) = m$ with $n \geq m$ their (proper) *subresultant* prs is a sequence of polynomials similar to the *Euclidean* prs, the sequence obtained by applying in $\mathbb{Q}[x]$[4] Euclid's polynomial gcd algorithm on $f(x), g(x)$.[5,6] The two sequences differ in that the coefficients of each polynomial in the subresultant prs are the determinants, *subresultants*, of sub-matrices of `sylvester1`; moreover, the two sign sequences may differ. The determinant of `sylvester1` itself is called the *resultant* of $f(x), g(x)$ and serves as a criterion of whether the two polynomials have common roots or not.

[3] `TeXmacs` has been used as interface. The `Sympy` matrix output has been slightly modified to conform to mathematical notation.

[4] Or in $\mathbb{Z}[x]$, if we use our `Sympy` function `euclid_PG(p, q, x, method = 0)`, mentioned in Example 2, Section 2.

[5] A formal definition of a subresultant prs can be found in almost all references (see for example the one by Kerber, [21]) and, hence, it is omitted in this paper in order to save space.

[6] If there are no gaps in the degree sequence, the prs is called *complete* else it is called *incomplete*. For complete prs's the sign sequences of the subresultant and Euclidean prs's are identical.

By contrast, the determinant of matrix `sylvester2` defines the *modified resultant* of the two polynomials f, g and, therefore, the determinant of any sub-matrix of `sylvester2` is called a *modified subresultant*. Modified supresultant prs's are related to Sturm sequences and have been studied elsewhere [7].

The determinants of the two matrices `sylvester1` and `sylvester2` *may* differ in sign. Indeed, for the polynomials of Example 1 the determinant of S_1 is

```
Python] S1.det()
```

$$27 \cdot a^3 \cdot d^2 - 18 \cdot a^2 \cdot b \cdot c \cdot d + 4 \cdot a^2 \cdot c^3 + 4 \cdot a \cdot b^3 \cdot d - a \cdot b^2 \cdot c^2\,,$$

whereas the determinant of S_2 is

```
Python] S2.det()
```

$$-27 \cdot a^3 \cdot d^2 + 18 \cdot a^2 \cdot b \cdot c \cdot d - 4 \cdot a^2 \cdot c^3 - 4 \cdot a \cdot b^3 \cdot d + a \cdot b^2 \cdot c^2\,.$$

It has been shown in the literature that, using `sylvester1`, the polynomials in the subresultant prs are proportional to those obtained by the Euclidean algorithm. Moreover, for complete sequences, the subresultant prs is identical to the Euclidean prs (computed in $\mathbb{Z}[x]$), where the polynomial remainder p_{i+2} in the latter sequence has been divided by the square of the leading coefficient of the polynomial p_i [2].

As Sylvester pointed out, the coefficients of the polynomial remainders obtained as subresultants are the *smallest possible* without introducing rationals and without computing (integer) greatest common divisors. However, since it is time consuming — and tiring — to evaluate a large number of determinants, the subresultant prs algorithm using the function `prem` was developed [10], [11], [15], [22], and is described for completion in Section 2.1; see also [19]. This well known algorithm is the only one in the paper that does not use the fundamental procedure with `sylvester1` described below and, instead, uses the quantities β_i in formula (7).

1.1 The fundamental procedure with `sylvester1`

The purpose of this procedure is to evaluate the determinant of the submatrix of `sylvester1` corresponding to the leading coefficient of a remainder we have computed. This way, we obtain the exact sign and value of the subresultant and we can adjust the remainder accordingly.

It turns out that the objective stated above can be achieved with other matrices as well, as for example with the matrix Bezout — computed with the function `bezout` — or with the matrix Hybrid Bezout — computed with the function `hybrid_bezout` — to name a couple [16]. Moreover, additional approaches to compute just the sign of the determinant of an integer matrix exist in the literature [20], and can be used in Section 3.1.

We hope to implement these various approaches in the future, and to compare our results with others existing in the literature [1], but for the purposes of this paper we will restrict ourselves to `sylvester1`.

Sylvester's matrix `sylvester1` is used by all *new* methods in this paper to evaluate the correct sign of a polynomial remainder and to make its coefficients subresultants. It works as follows:[7]

Suppose that we have computed — with any one of the new methods — the i-th polynomial remainder of *expected*[8] degree (`expDeg`) m_i

[7] This is a variation of the fundamental procedure with `sylvester2` of the Van Vleck-Pell-Gordon triangularization method for computing the correct sign and coefficient values of a Sturmian remainder [7], [8].

[8] The expected degree, `expDeg`, of a remainder is deg(*divisor*) − 1 and may differ from its actual degree, `actDeg`.

$$s^{(i)} = s_0^{(i)} x^{m_i} + s_1^{(i)} x^{m_i - 1} + \cdots + s_{m_i}^{(i)}, \tag{1}$$

where all the coefficients $s_k^{(i)}, 0 \leq k \leq m_i$ are either integers or rationals, depending on the method used. Some of the leading coefficients will be zero, if the expected degree is greater than the actual degree (`actDeg`). The difference `expDeg`$-$`actDeg` is the variable `degDiffer` in Equation (9), which was first defined in relation to the Pell-Gordon theorem [7], [8], [24].

To compute the correct sign of the polynomial $s^{(i)}$ and to make its coefficients subresultants we use our own `Sympy` function `correctSign(degF, degG, S1, expDeg, degDiffer)` to evaluate the single subresultant corresponding to its leading (first nonzero) coefficient. In this function `degF, degG` are the degrees of the original polynomials that define `S1`, Sylvester's matrix `sylvester1`, and `expDeg, degDiffer` define the number of rows[9] and columns, respectively, to be deleted from `S1` in order to obtain the appropriate subresultant.

In other words, we evaluate the determinant $\mathrm{Det}\,(i,j) \neq 0$ of the sub-matrix of `sylvester1` corresponding to the *actual* leading coefficient $s_j^{(i)}$ of $s^{(i)}$. Then by forming the product

$$\frac{s^{(i)}}{s_j^{(i)}} \cdot \mathrm{Det}\,(i,j)$$

we obtain the i-th polynomial of the subresultant prs. This is so, because of the existing ratio equality

$$\left| \frac{\mathrm{Det}\,(i,j)}{s_j^{(i)}} \right| = \left| \frac{\mathrm{Det}\,(i,k)}{s_k^{(i)}} \right|$$

that holds for $j < k \leq m_i$; recall that $s_j^{(i)}$ is the *actual* leading coefficient of the polynomial remainder $s^{(i)}$, in equation (1).

1.2 Outline of the paper

In Section 2 we compute the subresultant prs of two polynomials using pseudo remainders. For completion we first describe the well known "old" method, which uses the function `prem`. Following that, the new subresultant prs method is presented, which uses the function `prem2` along with the fundamental procedure with `sylvester1` stated in Section 1.1.

In Section 3 we compute the subresultant prs of two polynomials by performing divisions in $\mathbb{Q}[x]$ and by applying the Pell-Gordon theorem of 1917, [24], to force the remainders into $\mathbb{Z}[x]$. The absolute values of the coefficients of the remainders thus computed are equal to the absolute values of the corresponding subresultants. To obtain the correct signs we evaluate for each remainder the (single) subresultant — the determinant of an appropriately chosen sub-matrix of `sylvester1` — corresponding to its leading coefficient and, if needed, adjust the sign of the remainder accordingly.[10] Its more efficient version, performs operations in $\mathbb{Q}\,[x]$ but does not use the Pell-Gordon Theorem.

In Section 4 we compute the subresultant prs of two polynomials by triangularizing Sylvester's matrix `sylvester2`. This is a variation of the Van Vleck-Pell-Gordon triangularization method for

[9] To be deleted from *both* groups of rows in `S1`.

[10] In other words, we use just the first half of the fundamental procedure with `sylvester1` stated in Section 1.1.

computing Sturm sequences [8]. We also take advantage of the special nature of `sylvester2` and triangularize matrices with just three rows when the sequence is complete. When pivoting, we use the Dodgson[11]-Bareiss integer preserving transformation [9], [17], which means that the remainders will all be in $\mathbb{Z}[x]$. However, as we see in Example 2, of Section 2, the signs of the remainders in an incomplete prs do not always agree with the signs of the subresultants. Therefore, here again, to obtain the correct signs and values of the coefficients of each remainder we apply the fundamental procedure with `sylvester1` mentioned in Section 1.1. Its more efficient version, triangularizes all the smaller matrices encountered above but does not update the matrix `sylvester2`.

Finally, in Section 5 we present our conclusions.

2 Subresultant PRS's Using Pseudo Remainders and Operations in $\mathbb{Z}[x]$

We describe two methods for computing subresultant prs's with pseudo remainders.

In Section 2.1 we present the well known method that uses `prem` along with the quantities β_i, in equation (7). We do this both for completion and comparison with the new method of Section 2.2.

In Section 2.2 we present a new method, that uses `prem2` along with the fundamental procedure with `sylvester1`, stated in Section 1.1, and altogether avoids the quantities β_i in equation (7).

The "old" and well known subresultant prs algorithm uses the pseudo-remainder function `prem(f, g, x)`, which initially was the only way to keep the remainders R_i in $\mathbb{Z}[x]$. To exactly divide by the leading coefficient of the divisor, `prem(f, g, x)` applies the remainder function `rem(f, g, x)` *after* premultiplying the dividend times the leading coefficient of the divisor according to the formula

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

where $\mathrm{LC}(R_{i-1})$ is the leading coefficient of the divisor R_{i-1}, and $\delta = \mathrm{degree}(R_{i-2}, x) - \mathrm{degree}(R_{i-1}, x) + 1$.[12]

This way, however, there is an increase in the size of coefficients, and a certain factor, β_i, is divided out to reduce the coefficients to subresultants. The whole procedure is described below in Section 2.1.

When a prs is complete, Equation (2) can be safely used because the sign sequence of the Euclidean prs coincides with the corresponding one of the Euclidean pseudo remainder prs obtained using `prem(f, g, x)`; see also Footnote 6. That is, the sequences of signs of the leading coefficients of the polynomials in the two prs's are identical.

However, for incomplete sequences the sign sequences are not necessarily identical and this can be really confusing as illustrated in the following example.[13]

[11] Charles Ludwidge Dodgson (1832-1898) is the same person widely known for his writing *Alice in Wonderland* under the pseudonym Lewis Carroll.

[12] In `Sympy` there exist the built-in function `prem(f, g, x)` that computes the pseudo remainder R according to Equation (2).

[13] On July 14, 2014 the author of this paper had to change the sign of the first polynomial in the subresultant pseudo-remainder sequence in wikipedia's article on polynomial greatest common divisor; see `https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor`.

Example 2 Consider the polynomials $f = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$ and $g = 3x^6 + 5x^4 - 4x^2 - 9x + 21$. These are the same polynomials used in the wikipedia article on polynomial gcd `https://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor`. The "correct" sign sequence of the Euclidean prs of these two polynomials is

$$-1, -1, 1, -1 \tag{3}$$

as can be easily verified by the following script:

```
Python]i = 1
       while degree(g, x) > 0:
           r = rem(f, g, x)
           print "r[",i,"] = ", sign(LC(r, x))
           i = i + 1
           f = g
           g = r


   r[ 1 ] =  -1
   r[ 2 ] =  -1
   r[ 3 ] =  1
   r[ 4 ] =  -1
```

In the script above we use the `Sympy` functions `rem(f, g, x)`, to compute the Euclidean prs in $\mathbb{Q}[x]$ and `LC(f, x)` to compute the leading coefficient of a polynomial.

If we now use the same script above with `rem(f, g, x)` replaced by `prem(f, g, x)`, the Euclidean pseudo remainder prs is computed in $\mathbb{Z}[x]$ with very big coefficients and the sign sequence changes to

$$-1, 1, 1, 1. \tag{4}$$

This inconsistency regarding the sign sequence is due to Equation (2), which does not use the absolute value of the leading coefficient of the divisor.[14] Equation (2) is used by the built-in function `prem(f, g, x)`, which, in Section 2.1, has been incorporated in our own procedure `subresultants_prem(f, g, x)`.

We can get the correct sign sequence (3) if instead of equation (2) we use the equation

$$|\operatorname{LC}(R_{i-1})|^{\delta} \cdot R_{i-2} = q_{i-2} \cdot R_{i-1} + R_i, \tag{5}$$

where we take the absolute value of the leading coefficient of the divisor. Equation (5) is implemented by our own function `prem2`,

```
Python] def prem2(p, q, x):
            delta = (degree(p, x) - degree(q, x) + 1)
            return rem(Abs(LC(q, x))**delta  *  p, q, x)
```

[14] Maybe this inconsistency is the reason why computer algebra systems such as `Xcas`, `maple` and *Mathematica* do not have a function to perform pseudo divisions.

However, this new function `prem2(f, g, x)` *cannot* replace `prem(f, g, x)` in the procedure `subresultants_prem(f, g, x)`, since in some cases we compute the wrong signs.[15] Therefore, in Section 2.2 we describe a new method, implemented as `subresultants_prem2(f, g, x)`, which performs *all* operations in $\mathbb{Z}[x]$, does not use the quantities β_i, and gives the "correct" sign sequence (3).

Therefore, we now have a choice and do not have to always use the "wrong" sign sequence (4), in which case, the reader is invariably asked to ignore the signs [12].

Moreover, we now have our own `Sympy` function `euclid_PG(f, g, x, method = 0)`, which computes the Euclidean prs in $\mathbb{Z}[x]$ *without* pseudo remainders; that is, each remainder is computed in $\mathbb{Q}[x]$ and then it is multiplied times a factor — derived from the Pell-Gordon theorem of 1917, which is Theorem 1 in this paper — to force it into $\mathbb{Z}[x]$. Calling this function with `method` set to 1, i.e. as `euclid_PG(f, g, x, method = 1)`, we can compute just the *first* remainder in $\mathbb{Z}[x]$, *without* pseudo division![16] This way we again obtain the "correct" sign sequence (3).

```
Python]i = 1
       while degree(g, x) > 0:
           r = euclid_PG(f, g, x, 1)
           print "r[",i,"] = ", sign(LC(r, x))
           i = i + 1
           f = g
           g = r

   r[ 1 ] =  -1
   r[ 2 ] =  -1
   r[ 3 ] =  1
   r[ 4 ] =  -1
```

An additional nice feature of `euclid_PG(f, g, x, method = 0)` is that the absolute values of the coefficients of the polynomials in the sequence equal the absolute values of the corresponding subresultants.

In view of the recent discovery of the Pell-Gordon theorem of 1917, and despite established practice ([22], p. 407), it is our belief that any attempt to force the polynomials of a prs from $\mathbb{Q}[x]$ into $\mathbb{Z}[x]$ should preserve the sign sequence computed in $\mathbb{Q}[x]$, otherwise confusion arises.

Caveat: Only subresultant prs's can have different sign sequences from Euclidean prs's computed in $\mathbb{Q}[x]$.

2.1 Subresultant prs's using the function `prem`

The algorithm for computing (in $\mathbb{Z}[x]$) the subresultant prs of $f(x), g(x)$ with pseudo remainders using the function `prem` — which is based on equation (2) — has been extensively studied in the

[15] For example, replace `prem(f, g, x)` by `prem2(f, g, x)` in `subresultants_prem(f, g, x)` and try it on the polynomials $2x^4+5x^3+5x^2-2x-1$ and $3x^3+3x^2+3x-4$; compare the answer with that of the built-in function `subresultants(f, g, x)`.

[16] The output of `euclid_PG(f, g, x, method = 1)` is the same as that of `prem2(f, g, x)`; they differ in that the first does the division in $\mathbb{Q}[x]$ whereas the second in $\mathbb{Z}[x]$.

literature [10], [11], [12], [14], [15], [22]. It involves the remainder sequence

$$\begin{aligned} R_{-1} &= \mathrm{pp}(f), \\ R_0 &= \mathrm{pp}(g), \\ R_1 &= \frac{\mathrm{prem}(R_{-1}, R_0, x)}{\beta_1}, \\ &\vdots \\ R_i &= \frac{\mathrm{prem}(R_{i-2}, R_{i-1}, x)}{\beta_i}, \\ &\vdots \end{aligned} \tag{6}$$

which reduces the coefficient explosion by dividing the pseudo remainders by the expression β_i given by the recurrence relations[17] [13]

$$\begin{aligned} \psi_1 &= -1, & \beta_1 &= (-1)^{\delta_1}, \\ \psi_i &= \frac{(-\mathrm{LC}(R_{i-2}, x))^{\delta_{i-1}-1}}{\psi_{i-1}^{\delta_{i-1}-2}}, & i &> 1, \\ \beta_i &= -\mathrm{LC}(R_{i-2}, x) \cdot \psi_i^{\delta_i - 1}, & i &> 1, \end{aligned} \tag{7}$$

where

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

We have implemented this method in `Sympy` as the function `subresultants_prem(f, g, x)`. Testing it on the functions of Example 2 we obtain:[18]

```
Python] f = x**8 + x** 6 - 3*x** 4 - 3*x** 3 + 8*x **2 + 2*x - 5
Python] g = 3*x** 6 + 5*x** 4 - 4*x** 2 - 9*x + 21
Python] subresultants_prem(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

which agrees with the output of the built-in function `subresultants`:

```
Python] subresultants(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

The weak point of this "old" method is that the function `prem` *distorts* the sign sequence of the Euclidean prs — see Example 2 above. Consequently, to the best of our knowledge, `prem` has been implemented only in the computer algebra system `Sympy`, which, however, computes the subresultant prs *without* divisions [21].

[17] An explanation of the derivation of the formula for the factor β_i can be found elsewhere [22].

[18] This pair of polynomials is a typical example of an incomplete subresultant prs whose signs are completely different from the signs of the Euclidean prs, computed in $\mathbb{Q}[x]$; see also Example 2.

2.2 Subresultant prs's using the function prem2

The new method for computing in $\mathbb{Z}[x]$ the subresultant prs of $f(x), g(x)$ premultiplies the dividend times the *absolute* value of the leading coefficient of the divisor — that is, it is based on equation (5) — and uses the fundamental procedure with `sylvester1` described in Section 1.1.

In other words, the new method uses Sylvester's matrix `sylvester1` to evaluate one subresultant per remainder, in order to obtain the correct sign of the remainder and to force its coefficients to become subresultants *without* the quantities β_i.

We have implemented the new subresultant prs method in the function `subresultants_prem2(f, g, x)`, and for the same two polynomials as in Section 2.1 we obtain the same result.

```
Python] subresultants_prem2(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

The strong and most important point of this new method is that the function `prem2` does *not* distort the sign sequence of the Euclidean prs — see Example 2 above! Therefore, `prem2` along with the function `subresultants_prem2(f, g, x)` can be safely implemented in all computer algebra systems.

3 Subresultant PRS's Using Regular Remainders and Operations in $\mathbb{Q}[x]$

In this Section we describe two methods for computing subresultant prs's with operations performed in $\mathbb{Q}[x]$.

In Section 3.1 we present a method that uses the Pell-Gordon theorem — to force the coefficients of the polynomials in the sequence into $\mathbb{Z}[x]$ — along with the fundamental procedure with `sylvester1`, stated in Section 1.1, to obtain the correct signs of their coefficients.

In Section 3.2 we present a method that only uses the fundamental procedure with `sylvester1` to accomplish both objectives stated above.

3.1 Subresultant prs's using the Pell-Gordon Theorem of 1917

The Pell-Gordon Theorem of 1917, [24], helps us compute the correct sign of a Sturmian remainder, of a complete or incomplete sequence, by using Sylvester's matrix `sylvester2`; details can be found elsewhere [7].

Here we use the Pell-Gordon Theorem of 1917 to force into $\mathbb{Z}[x]$ the coefficients of the remainders computed in $\mathbb{Q}[x]$. The correct sign of each remainder is obtained by applying the fundamental procedure with `sylvester1`, that is, by evaluating the sign of the subresultant corresponding to its leading coefficient.[19]

Theorem 1 ***(Pell-Gordon, 1917)*** *Let*

$$A = a_0 x^n + a_1 x^{n-1} + \cdots + a_n$$

[19] Recall that the absolute value of any modified subresultant (`sylvester2`) equals the absolute value of the corresponding subresultant (`sylvester1`).

and

$$B = b_0 x^n + b_1 x^{n-1} + \cdots + b_n$$

be two polynomials of the n-th degree. Modify the process of finding the highest common factor of A and B by taking at each stage the negative of the remainder. Let the i-th modified remainder be

$$R^{(i)} = r_0^{(i)} x^{m_i} + r_1^{(i)} x^{m_i - 1} + \cdots + r_{m_i}^{(i)}$$

where $(m_i + 1)$ *is the degree of the preceeding remainder, and where the first* $(p_i - 1)$ *coefficients of* $R^{(i)}$ *are zero, and the* p_i*-th coefficient* $\varrho_i = r_{p_i - 1}^{(i)}$ *is different from zero. Then for* $k = 0, 1, \ldots, m_i$ *the coefficients* $r_k^{(i)}$ *are given by*[20]

$$r_k^{(i)} = \frac{(-1)^{u_{i-1}} (-1)^{u_{i-2}} \cdots (-1)^{u_1} (-1)^{v_{i-1}}}{\varrho_{i-1}^{p_{i-1}+1} \varrho_{i-2}^{p_{i-2}+p_{i-1}} \cdots \varrho_1^{p_1+p_2} \varrho_0^{p_1}} \cdot \mathrm{Det}\,(i,k)\,, \tag{8}$$

where $u_{i-1} = 1 + 2 + \cdots + p_{i-1}$, $v_{i-1} = p_1 + p_2 + \cdots + p_{i-1}$ *and*

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

Proof The proof by induction of this theorem depends on two Lemmas and can be found in the original paper of Pell and Gordon.

As indicated elsewhere [7], we use a modification of formula (8) to compute the coefficients of the Sturm sequence. In that case $p_0 = \deg(A) - \deg(B) = 1$, since B is the derivative of A and, hence, the modified formula is shown below with the changes appearing in bold:

$$r_k^{(i)} = \frac{(-1)^{u_{i-1}} (-1)^{u_{i-2}} \cdots (-1)^{u_1} (-1)^{u_0} (-1)^{v_{i-1}}}{\varrho_{i-1}^{\boldsymbol{p_{i-1}+p_i-\mathtt{degDiffer}}} \varrho_{i-2}^{p_{i-2}+p_{i-1}} \cdots \varrho_1^{p_1+p_2} \varrho_0^{\boldsymbol{p_0}+p_1}} \cdot \frac{\mathrm{Det}\,(i,k)}{\boldsymbol{\varrho_{-1}}}, \tag{9}$$

where $\varrho_{-1} = a_0$, the leading coefficient of A and `degDiffer` is the difference between the expected degree m_i and the actual degree of the remainder.

It should be noted that in our (general) case $p_0 = \deg(A) - \deg(B)$ and that the division $\frac{\mathrm{Det}(i,k)}{\varrho_{-1}}$ is possible if the leading coefficient of A is the only element in the first column of `sylvester2`. Moreover, if the leading coefficient of A is negative we work with the polynomial negated and at the end we reverse the signs of all polys in the sequence. □

In our case we are given two polynomials A, B in $\mathbb{Z}[x]$ and we want to compute in $\mathbb{Z}[x]$ the subresultant prs of A, B using Theorem 1 — that is, performing divisions in $\mathbb{Q}[x]$. Our goal is achieved by: (a) multiplying, at each step, the remainder $R^{(i)} \in \mathbb{Q}[x]$ times the absolute value of the denominator of the first fraction in (9), and, (b) picking the correct sign of the leading coefficient.

[20] It is understood in (8) that $\varrho_0 = b_0$, $p_0 = 0$, and that $a_i = b_i = 0$ for $i > n$.

To wit, if we take the absolute value of the first fraction in (9) and multiply both sides of the equation times the denominator we obtain the following equation:

$$\left|\varrho_{i-1}^{p_{i-1}+\boldsymbol{p_i}-\mathbf{degDiffer}}\varrho_{i-2}^{p_{i-2}+p_{i-1}}\cdots\varrho_1^{p_1+p_2}\varrho_0^{\boldsymbol{p_0}+p_1}\right|\cdot r_k^{(i)}=\frac{\mathrm{Det}\,(i,k)}{\boldsymbol{\varrho_{-1}}}. \tag{10}$$

In equation (10) recall that $r_k^{(i)}$ is the k-th coefficient of the remainder $R^{(i)}\in\mathbb{Q}\,[x]$; in addition, notice that the expression $\frac{\mathrm{Det}(i,k)}{\boldsymbol{\varrho_{-1}}}$ is an integer because the division is exact.[21]

Therefore, we conclude that

$$\left|\varrho_{i-1}^{p_{i-1}+\boldsymbol{p_i}-\mathbf{degDiffer}}\varrho_{i-2}^{p_{i-2}+p_{i-1}}\cdots\varrho_1^{p_1+p_2}\varrho_0^{\boldsymbol{p_0}+p_1}\right|\cdot R^{(i)}\in\mathbb{Z}\,[x]\,, \tag{11}$$

that is, after we multiply the i-th remainder $R^{(i)}$ times the absolute value of the denominator of the first fraction in (9), the result will be in $\mathbb{Z}\,[x]$.

Having computed in $\mathbb{Z}\,[x]$ the correct absolute values of the coefficients of the remainder we use the fundamental procedure with `sylvester1`, stated in Section 1.1, to evaluate the sign of the single subresultant, corresponding to its leading coefficient; with that sign available we adjust, if needed, the sign of the polynomial. After that, the remainder becomes part of the subresultant prs.

The above procedure is easily programmed. The only critical point is to effectively compute the absolute value in (11). This value is not computed anew for each remainder $R^{(i)}$; instead, a multiplication factor, `mulFac`, is being updated as new leading coefficients are included in (11). So, if the current multiplication factor is

$$\mathrm{mulFac}_i=\left|\varrho_{i-1}^{p_{i-1}+\boldsymbol{p_i}-\mathbf{degDiffer_i}}\varrho_{i-2}^{p_{i-2}+p_{i-1}}\cdots\varrho_1^{p_1+p_2}\varrho_0^{\boldsymbol{p_0}+p_1}\right|,$$

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

$$\mathrm{mulFac}_{i+1}=\left|\varrho_i^{p_i+\boldsymbol{p_{i+1}}-\mathbf{degDiffer_{i+1}}}\varrho_{i-1}^{\mathbf{degDiffer_i}}\cdot\mathrm{mulFac}_i\right|,$$

which means that

$$\mathrm{mulFac}_{i+1}=\left|\varrho_i^{p_i+\boldsymbol{p_{i+1}}-\mathbf{degDiffer_{i+1}}}\varrho_{i-1}^{p_{i-1}+p_i}\cdots\varrho_1^{p_1+p_2}\varrho_0^{\boldsymbol{p_0}+p_1}\right|.$$

We have implemented this method as the `Sympy` procedure `subresultants_PG(f, g, x)`[22] and testing it on the same polynomials as in Example 2 we obtain the same sequence:

```
Python] f = x**8 + x** 6 - 3*x** 4 - 3*x** 3 + 8*x **2 + 2*x - 5
Python] g = 3*x** 6 + 5*x** 4 - 4*x** 2 - 9*x + 21
Python] subresultants_PG(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

[21] If $\deg(B)<\deg(A)$ the leading coefficient of A is the only element in the first column of the `sylvester2` matrix of A, B.

[22] See Footnote 2.

3.2 A more efficient version

Notice that in `subresultants_PG(f, g, x)` we first use the Pell-Gordon Theorem in order to convert the rational coefficients of each remainder to absolute values of subresultants and then our own function `correctSign(degF, degG, S1, expDeg, degDiffer)` to compute the correct sign of its leading coefficient.

In fact we can do better. To wit, we can use only the fundamental procedure with `sylvester1` to both convert the rational coefficients of each remainder to absolute values of subresultants and to compute the correct sign of its leading coefficient, as described in Section 1.1.

We have implemented this method in the function `subresultants_PG2(f, g, x)`, which for the same polynomials as above yields the same sequence:

```
Python] subresultants_PG2(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

4 Subresultant PRS's Using Matrix Triangularizations and Operations in $\mathbb{Z}[x]$

In this Section we describe two methods for computing subresultant prs's with matrix triangularizations and with operations in $\mathbb{Z}[x]$, since we use the Dodgson-Bareiss integer preserving transformation [9], [17].

In Section 4.1 we present a method that triangularizes Sylvester's matrix `sylvester2` and at the same time uses the fundamental procedure with `sylvester1`, stated in Section 1.1, to adjust the sign and value of the coefficients of each remainder.

In Section 4.2 we present a method that differs from the previous one in that it does not triangularize `sylvester2`.

4.1 Subresultant prs's using Sylvester's matrix `sylvester2`

Given the polynomials f, g, only Sylvester's matrices `sylvester1` and `sylvester2` are considered in this paper as the two possible candidates for triangularization in order to obtain the subresultant prs of the given polynomials. As stated in Section 1.1, other matrices will be considered in a future paper.

Of those two matrices, `sylvester1` is excluded because — as it was shown by Laidacker [23] — it provides information *only* about a greatest common divisor of f, g.

Theorem 2 ***(Laidacker, 1969)*** *Let f and g be two polynomials and let S_1 be their Sylvester matrix `sylvester1(f, g)`. If, using row transformations only, we bring S_1 into its upper triangular form, $T(S_1)$, then the last nonzero row of $T(S_1)$ gives us the coefficients of a greatest common divisor of f, g.*

On the other hand, it was realized that triangularizing `sylvester2` can give us additional information — initially, about the polynomials of a (complete) Sturm sequence [7], [8] and, now, in this paper, about the polynomials in a subresultant prs.

Regarding complete Sturn sequences, Van Vleck, [27], realized that one does not have to compute modified subresultants [7] of Sylvester's matrix `sylvester2` in order to find the coefficients of the polynomial remainders. Instead, it suffices to simply triangularize `sylvester2` using integer preserving transformations, in which case the modified subresultants (the coefficients) can be read off the triangularized matrix. We have the following ([27], p. 8):

Theorem 3 ***(Van Vleck, 1899)*** *Let f and $g = f'$ be two polynomials of degree n and $n-1$ respectively and let S_2 be their Sylvester matrix* `sylvester2(f, g)`*. If, using integer preserving transformations, we bring S_2 into its upper triangular form, $T(S_2)$, then the even rows of $T(S_2)$ furnish the coefficients of the successive polynomial remainders of the Sturm sequence. The coefficients taken from a given row are multiplied times $(-1)^k$, where k is the number of negative elements on the principle diagonal above the row under consideration.*

Van Vleck takes advantage of the special form of `sylvester2` and computes $T(S_2)$ by updating only two rows at a time; to update these two rows he triangularizes a matrix of only three rows, a fact that makes his procedure extremely efficient. To keep the coefficients small he removes at each step the greatest common divisor (content) of the elements in both updated rows, and uses those reduced coefficients in the next three-row matrix.[23]

Van Vleck's computation is justified by the fact that in `sylvester2` the elements (entries) of any two consecutive rows are the same as those of the two preceding rows.

Therefore, if in any row the values of the elements are changed by adding a multiple of the preceding row, exactly the same change can be made in the elements of each alternate row thereafter, without altering the value of any modied subresultant that appears as a coefficient in one of the polynomials of the Sturm sequence.

In conclusion, Van Vleck presented a very efficient procedure for computing complete Sturm sequences in $\mathbb{Z}[x]$. With the help of the Pell-Gordon Theorem of 1917, [24], Van Vleck's method was extended to handle both complete and incomplete Sturm sequences; a description of its extension, the Van Vleck-Pell-Gordon method, can be found elsewhere [8].

Our purpose is to compute subresultant prs's using a slight variation of the Van Vleck-Pell-Gordon method [8]. Namely, we triangularize Sylvester's matrix `sylvester2` to compute polynomials - candidates for the subresultant prs and, then, we apply the fundamental procedure with `sylvester1`, stated in Section 1.1, to make the candidates members of the prs.[24]

An example will make everything clear.

Example 3 Using our subresultant matrix triangularization method we will compute the incomplete subresultant prs of the polynomials $f = 2x^5 - 3x^4 - 3$ and $g = 10x^4 - 12x^3$. We choose this example because the dimensions of the Sylvester matrices `sylvester1` and `sylvester2` can fit in a page.

After we construct S_1 and S_2 we form matrix M from the first two rows of S_2.

```
Python]  M = Matrix([S2[1, :], rotateR(S2[1, :], 1), rotateR(S2[0, :], 1)])
Python]  M
```

$$\begin{pmatrix} 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & -3 & 0 & 0 & 0 & -3 & 0 & 0 & 0 \end{pmatrix}$$

[23] It turns out that computationally this is the fastest way to proceed [18].

[24] By contrast, Van Vleck's extended method uses Sylvester's matrix `sylvester2` to accomplish both objectives.

Notice that our matrix M computes the remainder and not the Sturmian remainder as in Van Vleck's paper [27].[25] We pivot twice and obtain the matrices M_1 and M_2:

```
Python]  M1 = pivot(M, 0, 1)
Python]  M2 = pivot(M1, 1, 2)
Python]  M2
```

$$\begin{pmatrix} 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \end{pmatrix}$$

So, the first polynomial candidate for the subresultant prs is the last row of $M2$; its degree, 3, is determined with the help of our `Sympy` function `findDegree`, which counts the number of leading zeros in the last row. Moreover, to determine if the coefficients of $-72x^3 - 300$ are subresultants we use matrix S_1 to compute the subresultant corresponding to the leading coefficient. Since the expected degree equals the actual one we call our `Sympy` function `correctSign(5, 4, S1, 3, 0)` and the result is -72. So this indeed is a member of the subresultant prs.

We now use the last two rows of M_2 to replace the third and fourth rows of S_2, respectively, and we have

```
Python]  S2
```

$$\begin{pmatrix} 2 & -3 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \\ 0 & 0 & 2 & -3 & 0 & 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -3 & 0 & 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & 0 & 0 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 \end{pmatrix}$$

To compute the second polynomial of the subresultant prs we use rows 3 and 4 of S_2 to form the 3-rows matrix M.

```
Python]  M = Matrix([S2[3, :], rotateR(S2[3, :], 1), rotateR(S2[2, :], 1)])
Python]  M
```

$$\begin{pmatrix} 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 \\ 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

We pivot twice and obtain the matrices M_1 and M_2:

```
Python]  M1 = pivot(M, 0, 3)
Python]  M2 = pivot(M1, 1, 4)
```

[25] Matrix M in Van Vleck's paper is like ours, but with rows 2 and 3 swapped, to obtain the remainder negated.

```
Python]  M2
```

$$\begin{pmatrix} 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -216000 & 259200 & 0 & 0 \end{pmatrix}$$

The second polynomial candidate for the subresultant prs is the last row of $M2$; its degree, 1, is determined with the help of our `Sympy` function `findDegree`. Moreover, to determine if the coefficients of $-216000x + 259200$ are subresultants we use matrix S_1 to compute the subresultant corresponding to the leading coefficient. Since, now, the expected degree (2) minus the actual degree (1) equals one we call our `Sympy` function `correctSign(5, 4, S1, 2, 1)` and the result is -2160. So the second member of the subresultant prs is

$$\frac{-216000x + 259200}{-216000} \cdot (-2160) = -2160x + 2592. \tag{12}$$

Updating S_2 is a bit tricky. Obviously, the second row of M_2 will replace the 5-th row of S_2 but the — updated in Equation (12) — 3-rd row of M_2 will now replace the 7-th row of S_2![26] Row 6 in S_2 is a redundant row and so it can be replaced again by the second row of M_2 rotated by one.

```
Python]  S2
```

$$\begin{pmatrix} 2 & -3 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & 0 & 0 & 0 & -3 \\ 0 & 0 & 0 & 0 & 0 & 10 & -12 & 0 & 0 & 0 \end{pmatrix}$$

To compute the third, and final, constant polynomial in the subresultant prs we form matrix M, which now has 4 rows!

```
Python]  M = Matrix([S2[6, :], rotateR(S2[6, :], 1), rotateR(S2[6, :], 2),
                     rotateR(S2[5, :], 1)])
Python]  M
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 \\ 0 & 0 & 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 \end{pmatrix}$$

We pivot three times and obtain the matrices M_1, M_2 and M_3:

```
Python]  M1 = pivot(M, 0, 6)
Python]  M2 = pivot(M1, 1, 7)
```

[26] If we start enumeration with 0, the number of the row in S_2 that will be replaced is indicated by the number of leading zeros in the 3-rd row of M_2.

```
Python]  M3 = pivot(M2, 2, 8)
Python]  M3
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4277135425536 \end{pmatrix}$$

The third and last (constant) polynomial candidate for the subresultant prs is the last row of M_3; its degree, 0, is determined from its position in M_3 with the help of our `Sympy` function `findDegree`. Moreover, to determine if 4277135425536 is the resultant we take the determinant os matrix S_1. Since, now, the expected degree minus the actual one equals 0 we call our `Sympy` function `correctSign(5, 4, S1, 0, 0)` and the result is 11459232. So the last member of the subresultant prs is

$$\frac{4277135425536}{4277135425536} \cdot 11459232 = 11459232.$$

To update S_2 we copy the last three rows of M_3 — with the last row updated — into the last three rows of S_2.

```
Python]  S2
```

$$\begin{pmatrix} 2 & -3 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & -12 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -72 & 0 & 0 & -300 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2160 & 2592 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11459232 \end{pmatrix}$$

Therefore, as can seen from the triangularized form of S_2, the subresultant prs of the polynomials $f = 2x^5 - 3x^4 - 3$ and $g = 10x^4 - 12x^3$ is

$$\left[2x^5 - 3x^4 - 3,\ \ 10x^4 - 12x^3,\ \ -72x^3 - 300,\ \ -2160x + 2592,\ \ 11459232\right],$$

which agrees with the result obtained with the function `subresultants` of `Sympy`.

```
Python]  subresultants(2*x**5 - 3*x**4 - 3, 10*x**4 - 12*x**3, x)
 [2*x**5 - 3*x**4 - 3, 10*x**4 - 12*x**3, -72*x**3 - 300, -2160*x + 2592,
11459232]
```

We have implemented this method into the `Sympy` procedure `subresultants_triang(f, g, x, method = 0)`; see Footnote 2. Since the dimensions of a matrix can be very big, the default value of the optional fourth argument is 0, which does not print out the triangularized matrix S_2. Set `method = 1` if you want to see S_2. We test `subresultants_triang(f, g, x, method = 0)` on the same polynomials used in Example 2 and obtain the same subresultant prs:

```
Python] f = x**8 + x** 6 - 3*x** 4 - 3*x** 3 + 8*x **2 + 2*x - 5
Python] g = 3*x** 6 + 5*x** 4 - 4*x** 2 - 9*x + 21
Python] subresultants_triang(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

4.2 A more efficient version

While working on the choice of making S_2 visible or not, it occurred to us that matrix S_2 does not have to be updated at all. We have implemented this version into the `Sympy` procedure `subresultants_triang2(f, g, x)`.

Testing `subresultants_triang2(f, g, x)` on the same polynomials as above we obtain the same subresultant prs:

```
Python] subresultants_triang2(f, g, x)
 [x**8 + x**6 - 3*x**4 - 3*x**3 + 8*x**2 + 2*x - 5, 3*x**6 + 5*x**4 - 4*x**2 - 9*x
+ 21, 15*x**4 - 3*x**2 + 9, 65*x**2 + 125*x - 245, 9326*x - 12300, 260708]
```

5 Conclusions

The Pell-Gordon Theorem of 1917, [24], was instrumental in helping us generalize Van Vleck's method for computing Sturm sequences — complete or incomplete — by triangularizing Sylvester's matrix `sylvester2`. It showed us that the main idea of that method is the following: in order to obtain the correct coefficients — in value and / or sign — of each Sturmian remainder we *have* to evaluate a single *modified* subresultant corresponding to its leading coefficient [7], [8].

A variation of the main procedure of Van Vleck's generalized method, [8], is used by all three methods described in this paper in order to compute the subresultant prs of two polynomials. This variation is called the "fundamental procedure with `sylvester1`" and is stated in Section 1.1.

In the first method, `subresultants_prem2(f, g, x)`, we incorporate the new pseudo remainder, `prem2(f, g, x)`, which uses the absolute value of the leading coefficient of the divisor. This way, we preserve the "correct" sign sequence of the Euclidean prs, as discussed in Example 2.

The second method, `subresultants_PG(f, g, x)`, does divisions over the rationals and uses the Pell-Gordon theorem to convert the coefficients of the polynomial remainders to integers. Here we have an implicit interplay between the two Sylvester matrices, `sylvester1` and `sylvester2`. Its more efficient method does not use the Pell-Gordon theorem.

Finally, in the third method, `subresultants_triang(f, g, x)`, we see — for the first time in the literature — an explicit interplay between the two Sylvester matrices, `sylvester1` and `sylvester2`. While we triangularize the latter to obtain polynomials - candidates for the subresultant prs, we evaluate determinants of sub-matrices of the former in order to make the candidates actual members of the prs by adjusting their coefficients accordingly! Its more efficient method does not triangularize `sylvester2`.

References

[1] Abdeljaoued, J., Diaz–Toca, G.M., Gonzalez–Vega, L.:"Bezout Matrices, Subresultant Polynomials and Parameters". *Applied Mathematics and Computation*, **214**, (2009), 588–594.

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

[3] Akritas, A.G.:"A New Method for Computing Polynomial Greatest Common Divisors and Polynomial Remainder Sequences". *Numerische Mathematik*, **52**, (1988), 119–127.

[4] Akritas, A.G.:"Exact algorithms for the matrix-triangularization subresultant prs method". In: Erich Kaltofen & Stephen M. Watt (Eds): Proceedings of the Conference on Computers and Mathematics, Boston, Massachusetts, (June, 1989), 145–155.

[5] Akritas, A.G.:*Elements of Computer Algebra with Applications*. John Wiley Interscience, New York, (1989).

[6] Akritas, A.G.:"Sylvester's Forgotten Form of the Resultant.". *Fibonacci Quarterly*, **31**, (1993), 325–332.

[7] Akritas, A.G., Malaschonok, G.I., Vigklas, P.S.:"Sturm Sequences and Modified Subresultant Polynomial Remainder Sequences". To appear, 2014.

[8] Akritas, A.G., Malaschonok, G.I., Vigklas, P.S.:"On a Theorem by Van Vleck Regarding Sturm Sequences.". To appear, 2014.

[9] Bareiss, E.H.:"Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination". *Mathematics of Computation*, **22**, (1968), 565–578.

[10] Brown, W.S.:"The subresultant PRS Algorithm". *ACM Transactions on Mathematical Software*, **4**(3), (1978), 237–249.

[11] Brown, W.S., Traub, J.F.:"On Euclids Algorithm and the Theory of subresultants". *Journal of the Association for Computing Machinery*, **18**, (1971), 505–514.

[12] Chen, R.:"The subresultant Polynomial Remainder Sequence Algorithm". `www.math.ubc.ca/~reichst/423-project-subresultant.pdf`, (March 23, 2013), 1–12.

[13] Cohen, J.E.:*Computer Algebra and Symbolic Computation – Mathematical Methods*. A.K. Peters, Massachusetts, (2003).

[14] Collins, G.E.:"Polynomial Remainder Sequences and Determinants". *American Mathematical Monthly*, **73**(7), (1966), 708–712.

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

[16] Diaz–Toca, G.M., Gonzalez–Vega, L.:"Various New Expressions for Subresultants and Their Applications". *Applicable Algebra in Engineering, Communication and Computing*, **15**, (2004), 233–266.

[17] Dodgson, C.L.: "Condensation of Determinants". *Proceedings of the Royal Society of London*, **15**, (1866), 150–155.

[18] von zur Gathen, J., Lücking, T.: "Subresultants Revisited". *Theoretical Computer Science*, **297**, (2003), 199–239.

[19] Habicht, W.: "Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens". *Commentarii Mathematici Helvetici*, **21**, (1948), 99–116.

[20] Kaltofen, E., Villard, G.: "Computing the sign or the value of the determinant of an integer matrix, a complexity survey". *Journal of Computational and Applied Mathematics*, **162**(1), (2004), 133–146.

[21] Kerber, M.: " Division-Free computation of subresultants using Bezout matrices". Tech. Report MPI-I-2006-1-006, Saarbrucken, 2006.

[22] Knuth, D.E.: *The Art of Computer Programming, Vol. 2.* Second Edition, Addison-Wesley, Massachusetts, (1981).

[23] Laidacker, M.A.: "Another theorem relating Sylvester's matrix and the greatest common divisor. *Mathematics Magazine*, **42**, (1969), 126-128.

[24] Pell, A.J., Gordon, R.L.: "The Modified Remainders Obtained in Finding the Highest Common Factor of Two Polynomials". *Annals of Mathematics*, Second Series, **18**(4), (Jun., 1917), 188–193.

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

[26] 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.

[27] Van Vleck, E.B.: "On the Determination of a Series of Sturm's Functions by the Calculation of a Single Determinant". *Annals of Mathematics*, Second Series, **1**(1/4), (1899 - 1900), 1–13.