https://doi.org/10.53656/math2024-4-3-ani

Science in Education
Научно-методически статии

AN ITERATIVE ALGORITHM FOR DETERMINING THE GREATEST COMMON DIVISOR OF TWO OR MORE UNIVARIATE POLYNOMIALS

Verica Milutinović
Faculty of Education, University of Kragujevac – Jagodina (Serbia)

Abstract. The GCD problem in polynomial rings has long intrigued mathematicians for its diverse applications, leading to methods like the Euclidean algorithm, Routh array, and matrix-based approaches. Despite the low costs of the Euclidean algorithm, it faces numerical instability, while matrix-based techniques, though stable, involve higher computational expenses. The goal of this paper is to introduce a novel approach to determining the greatest common divisor (GCD) of multiple polynomials in a single variable, particularly suitable for interdisciplinary teaching in mathematics and programming. Our methodology involves iterating through the entire set of polynomials directly, aiming to enhance the procedure's efficiency while maintaining low computational costs. Numerous examples are provided to illustrate its practical application in teaching, ranging from easy to challenging scenarios, as well as Python implementation of the given procedure.

Keywords: greatest common divisor; univariate polynomials; algorithm; Python program

1. Introduction

The issue of determining the greatest common divisor (GCD) of two or more members of a polynomial ring has long attracted the attention of mathematicians, and it has a wide range of applications (Christou et al. 2011; Pace & Barnett 1973). Over the years different methods have been developed. Some of them are the Euclidien algorithm, Routh's array algorithm (Fryer 1959), Barnett's use of the companion matrix (Barnett 1970), Blankinship's matrix methods (Blankinship 1963), Weinstock's iterative method (Pace & Barnett 1973; Weinstock 1960), Extended Row Equivalence and Shifting operations (ERES) method (Christou et al. 2010; Karcanias 1987), and a lot of different matrix methods (Boito 2012; Christou et al. 2017; Christou et al. 2011; Mitrouli & Karcanias 1993). The basic Euclidean algorithm is computationally efficient, with a cost that scales quadratically with the degree of the

input polynomials. However, it suffers from numerical instability. However, while the majority of matrix-based variants are numerically stable, they often have higher than quadratic computational costs (Boito 2012).

Any method that works for two polynomials may be adapted to work for more. However, the fact that we typically have to deal with a high number of polynomials is a key difficulty for some applications of the GCD, and the pairwise type techniques for GCD (Blankinship 1963) are not suited for such applications. Matrix-based approaches tend to have higher performance and numerical stability, especially when dealing with large sets of polynomials, due to the usage of the complete set of polynomials. In this paper, based on algorithm developed by Prešić (Prešić 1997) that refers to systems of two algebraic equations, we have come to an iterative new algorithm for finding GCD of more polynomials in one variable simultaneously. The objective is to reduce computation costs while achieving improved performance and numerical stability. The main advantage of the presented method is that it works with complete set of polynomials directly.

In Section 2 of this paper, we present several definitions, theorems, and lemmas related to our method of computing polynomials' GCD. This was the mathematical background for the algorithm we developed. A few examples are given in Section 3 along with a practical explanation of how to use this approach to find GCD. The description of the algorithm was provided and computational costs are discussed in Section 4. The Python implementation of the method and the testing outcomes are shown in Section 5. The conclusion with practical implications is presented in Section 6.

2. Mathematical background for calculating greatest common divisor of polynomials

In this section, we provide an overview of the definitions, theorems, and lemmas related to GCD of polynomials.

Let R be a commutative ring with identity. Considerations below are for the ring of integers (although the procedure works with some rational coefficients too). Let $R[x]$ be the polynomial ring over R.

Theorem 1. *(Division Algorithm) For any polynomials* $f(x) \in R[x]$ *and* $0 \neq g(x) \in R[x], \exists!$ *polynomials* $q(x), r(x) \in R[x]$ *and* $f(x) = q(x)g(x)+r(x)$, *where either* $deg(r(x)) < deg(g(x))$ *or* $r(x) = 0$.

We denote traditionally $f|g$ when $g = q.f$.

Definition 1. A monic polynomial $d(x) \in R[x]$ is called *greatest common divisor* (GCD) of $f(x)$ and $g(x) \in R[x]$ iff $d|f, d|g$, and $\forall e \in R[x](e|f \wedge e|g \Rightarrow e|d)$. **Remark** A monic polynomial is a polynomial with only one variable whose leading coefficient is equal to 1.

In this paper the greatest common divisor of $f(x)$ and $g(x)$ is denoted by

$gcd(f(x), g(x))$ or $\langle f(x), g(x)\rangle$. If $gcd(f(x), g(x)) = 1$, then the polynomials $f(x)$ and $g(x)$ are said to be *relatively prime.*

Lemma 1. Let $p_1(x), p_2(x) \in R[x]$ *be any nonzero polynomials. Then*

$$\langle p_1(x), p_2(x)\rangle = \langle p_1(x), p_2(x) - A(x) \cdot p_1(x)\rangle.$$

The Euclidean algorithm for finding GCD of two polynomials actually represents a special case of Lemma 1 when for two nonzero polynomials $f(x)$ and $g(x) \in R[x], deg(f) \geq deg(g)$ and $f(x) = q(x)g(x) + r(x)$ $(deg(r(x)) < deg(g(x))$ or $r(x) = 0)$, we choose $q(x)$ such that $r(x)$ is division remainder of polynomial $f(x)$ with polynomial $g(x)$. In this case:

• If $r(x) = 0$ then $g(x)$ divides $f(x)$ and $\langle f(x), g(x)\rangle = c \cdot g(x)$ for some constant $c \in R$.

• If $r(x) \neq 0$ then it is easy to show that $\langle f(x), g(x)\rangle = \langle g(x), r(x)\rangle$.

The Euclidean algorithm continually implements the special case of Lemma 1 and the division algorithm to achieve a remainder equal to 0. The degrees of the polynomials are decreasing while the procedure is repeated, and the GCD of the two polynomials may then be calculated in a limited number of steps.

Lemma 2. *Let* $p_1(x), p_2(x), \ldots, p_n(x) \in R[x]$ *be* n *nonzero polynomials. Then we have:*

$$\langle p_1(x), p_2(x), \ldots, p_n(x)\rangle = \langle p_i(x), p_1(x) - \lambda_1(x)p_i(x),$$
$$p_2(x) - \lambda_2(x)p_i(x), \ldots, p_n(x) - \lambda_n(x)a_i(x)\rangle,$$

where $p_i(x) \neq 0$ *is a polynomial of minimal degree among* $p_1(x), \ldots, p_n(x)$.

Let us denote by $rem(p(x), q(x))$ the remainder of the division of $p(x)$ by the polynomial $q(x)$. Then a special case of Lemma 2 would be:

$$\langle p_1(x), p_2(x), \ldots, p_n(x)\rangle = \langle p_i(x), rem(p_1(x), p_i(x)),$$
$$rem(p_2(x), p_i(x)), \ldots, rem(p_n(x), p_i(x))\rangle.$$

3. New method for determining the GCD of two or more univariate polynomials

Let the set of polynomials $P = \{p_1, p_2, \ldots, p_k\}, k > 1$, is given and corresponding degrees of the polynomials are $(n_1, n_2, \ldots, n_k)$. The set P will be called an $(n_1, n_2, \ldots, n_k)$*-ordered polynomial set.* This section provides detailed explanations of our method using examples.

Task. Find the GCD of the following sets of polynomials in $R[x]$:

a. $x^3 + x^2 - x + 2, 2x^3 - 5x^2 + 5x - 3$;

b. $x^3 + 1, x^2 + 1$;

c. $7x^{11} + x^9, 7x^2 + 1, -7x^7 - x^5 + 7x^2 + 1$;

d. $2x^3+5x^2+2x+5, x^4+x^3+12x^2+x+11, 3x^4+7x^3+7x^2+7x+1,$
$-4x^4+5x^3-x^2+5x+3.$

Instruction. The method we will employ to calculate the GCD of a set of polynomials implements Lemma 2 and the Division algorithm. This method is comparable to Gauss' algorithm for solving a systems of linear equations. We consider each polynomial p of degree n of the set as a linear equations $p=0$ with unknowns $x^n, x^{n-1}, \dots x^0$. In order to obtain the unique GCD of the polynomial set (if there is one) or some non-zero element of R (in case the polynomials are relatively prime), we build a chain of equivalences successively for as long as needed.

Solutions:

a) Let us denote $g_a(x)=\langle x^3+x^2-x+2, 2x^3-5x^2+5x-3\rangle$.

In this subtask we are dealing with a $(3,3)$-ordered polynomial set, where the degrees of polynomials are equal. **Rule:** when the degrees are equal, the first polynomial will always be retained. So, we multiply the first polynomial by -2 and add the obtained polynomial to the second (like in Gauss elimination). Then, as $(2x^3-5x^2+5x-3)-2(x^3+x^2-x+2)=-7x^2+7x-7$, based on Lemma 2 we get (multiplying result by $-1/7$ in order to simplify it)

$$g_a(x)=\langle x^3+x^2-x+2, x^2-x+1\rangle.$$

It is worth noticing that now, rather than working with two cubic polynomials, we have "shifted situation down" to finding the GCD of one cubic and one quadratic polynomial. So, we "descended" from a $(3,3)$-ordered polynomial set to a $(3,2)$-ordered polynomial set.

Now we multiply the second polynomial by $-x$ (because of the lower degree) and add it to the first:

$$(x^3+x^2-x+2)-x(x^2-x+1)=2x^2-2x+2.$$

After multiplying this polynomial by $1/2$ we get

$$g_a(x)=<x^2-x+1, x^2-x+1>,$$

shifting down from a $(3,2)$ to a $(2,2)$ set. Because $(x^2-x+1)-(x^2-x+1)=0$, we have $g_a(x)=\langle x^2-x+1, 0\rangle = x^2-x+1$. □

b) Let us denote $g_b(x)=\langle x^3+1, x^2+1\rangle$.

In this subtask we have a $(3,2)$ set. We rewrite the second polynomial at the first place, and as $(x^3+1)-x(x^2+1)=-x+1$ we get

$$g_b(x)=\langle x^2+1, -x+1\rangle,$$

reducing to a $(2,1)$ set.

With a similar to the previous step, since $(x^2+1)+x(-x+1)=x+1$,

we get

$$g_b(x) = \langle -x+1, x+1 \rangle.$$

Now, we have an $(1,1)$ set and as $(-x+1)+(x+1)=2$ we get

$$g_b(x) = \langle -x+1, 2 \rangle = 1.$$

The new case is an $(1,0)$ set but we came to the non-zero value 2 as a second polynomial in the set. This leads to the conclusion that the GCD of our polynomial set is 1, i.e. the polynomials are relatively prime. □

c. Let us denote $g_c(x) = \langle 7x^{11}+x^9, 7x^2+1, -7x^7-x^5+7x^2+1 \rangle$.

In this subtask we should find the GCD of three univariate polynomials that form an (11, 2, 7) set. First, we find a polynomial of the lowest degree (which is the second). We rewrite it on the first place in the list. Every other polynomial will be replaced by a new one, calculated as in previous examples, except that now we look at the pairs composed of the polynomial with the lowest degree and another one whose turn is for replacement. The new polynomial that we get by transformations of polynomials $p(x)$ and $q(x)$ is denoted by $rep(f(x), g(x))$. The procedure could be written in this way:

$$g_c(x) = \langle 7x^2+1, rep(7x^{11}+x^9, 7x^2+1), rep(-7x^7-x^5+7x^2+1, 7x^2+1) \rangle.$$

For the first replacement we obtain $7x^{11}+x^9-x^9(7x^2+1)=0$ and for the second $(-7x^7-x^5+7x^2+1)+x^5(7x^2+1)=7x^2+1$ thus we now have

$$g_c(x) = \langle 7x^2+1, 0, 7x^2+1 \rangle$$

and the GCD of the three polynomials is $g_c(x) = 7x^2+1$. □

d. Let us denote $g_d(x) = \langle 2x^3+5x^2+2x+5,\ x^4+x^3+12x^2+x+11,\ 3x^4+7x^3+7x^2+7x+1, -4x^4+5x^3-x^2+5x+3 \rangle = \langle p_1, p_2, p_3, p_4 \rangle$ – a $(3,4,4,4)$ set. The polynomial of the lowest degree is p_1 so

$$g_d(x) = \langle p_1, rep(p_2, p_1), rep(p_3, p_1), rep(p_4, p_1) \rangle.$$

Now we calculate three replacements

• $q_2(x) = rep(p_2, p_1) = rep(x^4+x^3+12x^2+x+11, 2x^3+5x^2+2x+5) =$

$= 2(x^4+x^3+12x^2+x+11) - x(2x^3+5x^2+2x+5) = -3x^3+22x^2-3x+22;$

• $q_3(x) = rep(p_3, p_1) = rep(3x^4+7x^3+7x^2+7x+1, 2x^3+5x^2+2x+5) =$

$= 2(3x^4+7x^3+7x^2+7x+1) - 3x(2x^3+5x^2+2x+5) = -x^3+2x^2-x+2$

• $q_4(x) = rep(p_4, p_1) = rep(-4x^4+5x^3-x^2+5x+3, 2x^3+5x^2+2x+5) =$

$(-4x^4+5x^3-x^2+5x+3) + 2x(2x^3+5x^2+2x+5) = 15x^3+3x^2+15x+3$

Note that in order to avoid rational coefficients in the second replacement, we multiplied the first polynomial by 2, and second by $-3x$, and then calculated the addition. And more, we simplify the fourth polynomial multiplying it by $1/3$ and $q_4(x) = 5x^3 + x^2 + 5x + 1$. In this way we got a new (3,3,3,3)-set. By applying the rule when degrees are equal, the first polynomial p_1 is retained, so

$$g_d(x) = \langle p_1, q_2, q_3, q_4 \rangle = \langle p_1, rep(q_2, p_1), rep(q_3, p_1), rep(q_4, p_1) \rangle.$$

We calculate the three replacements:

- $rep(q_2, p_1) = 2(-3x^3+22x^2-3x+22)+3(2x^3+5x^2+2x+5) = 59x^2+59$,
- $rep(q_3, p_1) = 2(-x^3 + 2x^2 - x + 2) + (2x^3 + 5x^2 + 2x + 5) = 9x^2 + 9$,
- $rep(q_4, p_1) = 2(5x^3 + x^2 + 5x + 1) - 5(2x^3 + 5x^2 + 2x + 5) = -23x^2 - 23$,

and after simplifying

$$g_d(x) = \langle 2x^3 + 5x^2 + 2x + 5, x^2 + 1, x^2 + 1, x^2 + 1 \rangle$$

Now we got a new (3,2,2,2)-set, where the second, third and fourth polynomials are equal and of the lowest degree. But $2x^3+5x^2+2x+5-2x(x^2+1) = 5x^2+5$ is simplified to x^2+1 and this is the GCD of the four polynomials. □

4. Algorithm for calculating GCD of two or more univariate polynomials

For two polynomials $p(x)$ and $q(x), deg(p) = d_p \geq d_q = deg(q)$ and Ax^{d_p} and Bx^{d_q} are their monomials of higher degree we define the function $rep(p(x), q(x)) = Bp(x) + (-Ax^{d_p - d_q})q(x)$.

Let us describe the general procedure for calculating GCD of list $L = (p_1, p_2, \ldots, p_k)$ of polynomials of variable x and $(n_1, n_2, \ldots, n_k)$ are the degrees of the polynomials. Let us also denote: with M – the minimal degree of a polynomial of L, with i – the ordinal of the current polynomial, and with D – the length of the list.

4.1. Algorithm for determining GCD of polynomials

Step 1 (Initialisation). $D = k, M = \min(n_1, n_2, \ldots, n_D)$ and let (without limiting the generality) the polynomial of degree M be p_1.

Step 2 (Outer loop). If $D > 1$ then go to Step 3. If $deg(p_1) > 0$ then it is the GCD of our polynomials. Otherwise the polynomials are relatively prime. Stop.

Step 3 (Inner loop). Create a new list L_1. Its first polynomial will be p_1. Then loop for $i = 2$ to D with step 1 and do:

Step 3.1. $q(x) = rep(p_i, p_1)$.

Step 3.2. If $deg(q) > 0$ push it in L_1 and continue.

Step 3.3. If $q(x) = 0$ continue, else the polynomials are relatively prime. Stop.

Step 4. Rearrange the elements of L_1 in order that the polynomial with smaller degree to be $p_1, M = deg(p_1)$ and rename L_1 to L. Let D be the length of L. Go to Step 2.

4.2. Complexity of the algorithm

For estimating the complexity of our algorithm we will compare it with the complexity of the Euclidean algorithm for finding GCD of two polynomials $p(x)$ and $q(x)$:

```
while(deg(r(x) = p(x) % q(x)) != 0)
{ p(x) = q(x); q(x) = r(x); }
return q(x);
```

where the % sign denote the operation finding remainder of $p(x)$ modulo $q(x)$.

Suppose we must find GCD of the polynomials $p_1(x)$ of degree n_1 and $p_2(x)$ of degree n_2. The Euclidean algorithm generates a sequence of polynomials:

$$p_1, p_2, p_3 = p_1 \% p_2, p_4 = p_2 \% p_3, \ldots, p_m = p_{m-2} \% p_{m-1}, p_{m+1} = p_{m-1} \% p_m,$$

where $deg(p_{m+1}) = 0$, the degrees of the polynomials are

$$n_1 \geq n_2 > n_3 > \cdots > n_m > n_{m+1} = 0,$$

and $g(x) = p_m(x)$. The division of the polynomials of degrees n_i and n_{i+1} needs $n_i - n_{i+1} + 1$ rounds with n_{i+1} multiplications on each round. The worst case will be the sequence of degrees

$$n > (n-1) > (n-2) > \ldots > 2 > 1$$

and the complexity will be

$$T = (n-n+1+1)(n-1) + (n-1-n+2+1)(n-2) + \cdots + (2-1+1).1 =$$

$$= 2(n-1) + 2(n-2) + \cdots + 2.1 = n(n-1) = O(n^2).$$

Let now estimate the complexity of our algorithm. Instead of finding the remainder it uses the *replacement* operation:

```
while(deg(r(x) = rep(p(x), m(x))) != 0)
{ p(x) = r(x); rearrange(p(x), m(x)) }
return r(x);
```

So, our algorithm also generates a sequence of replacement polynomials of decreasing degrees. Because the replacement operation in our method involves

multiplying two polynomials of degrees m and n by a monomial, which requires $m+1+n+1$ multiplications, the worst-case scenario is $2(n+1)$ multiplications (because $n > m$). Additionally, we must perform $\max deg(r(x))$ replacements, which, in the worst case, is n replacements, and the rearrangement of the two polynomials has a complexity of $O(1)$. When both polynomials are of degree n and are relatively prime, the number of multiplications is of the order n^2+n, indicating that our algorithm will find the GCD of two polynomials with asymptotically the same time complexity as the Euclidean algorithm.

When the algorithm must find GCD of $|P|$ polynomials the asymptotic estimation will be the same because $|P|$ is not a function of n.

5. Algorithm implementation in Python

Let us now describe the Python implementation of algorithm. Various programming paradigms, such as functional, object-oriented, and procedural programming, are supported by the flexible, high-level programming language Python. Currently, it is one of the most widely used programming languages. Algebra, numerical mathematics, number theory, calculus, and combinatorics are just a few of the mathematical topics that are covered in its extensive library. Despite this, we wanted to develop the fundamental program using structured programming without utilizing the library NumPy to enable big, multi-dimensional arrays and matrices.

For the program, a polynomial is presented as a list of ordered integer triples $[a, b, c]$, where a is the coefficient numerator, b is the coefficient denominator, and c is the degree of the monomial ax^c/b. For example, the polynomial $5x^3 - 2x + 3$ is presented by the list $[[5, 1, 3], [-2, 1, 1], [3, 1, 0]]$.

The program will utilize a list of lists to store and manage the set of polynomials we are operating with. This structure will be used to store both the initial set and the additional sets (mid-sets, let's call them "drop down" sets) that are obtained using the procedure.

The initial step, after starting the program, is entering a set of polynomials to determine their GCD with the procedure `inputPolys()` which puts each of them in a list of ordered triples.

```
def inputPolys():
  PSet = []
  i = 1
  n = int(input("How many polynomials do you want to input?"))
  while n != 0:
    print("Input {0}. polynomial: ".format(i))
    Poly = []
```

```
    Term = list(map(int,input("Enter the term in the form:
      coeff_numerator coeff_denominator integer_exponent\n
      (For example 3 4 2  for 3/4x^2 or 1 1 3 for x^3),\n
      or for the end of polynomial press <Enter> twice.\n
      Note: coeff_numerator, coeff_denominator, and
      integer_exponent should be separated with space:
      ").strip().split()))[:3]
    while Term != []:
      if Term[1] == 0:
        print("Coefficient denominator is 0,
          please enter again")
      else:
        Poly.append(Term)
        Term = list(map(int,input("Enter the next term:
          coeff_numerator coeff_denominator integer_exponent,
          or press <Enter> twice for the end of polynomial:
          ").strip().split()))[:3]
    n = n - 1
    i = i + 1
    if Poly != []:
      PSet.append(Poly)
    print("Find GCD of: ", PSet)
    return PSet
```

The main program executes a procedure for the polynomial set input (`inputPolys`), and a function for that set's GCD calculation (`poly_GCD`).

```
PolySet=inputPolys();
print("Find GCD of: ", PolySet)
poly_GCD(PolySet)
```

The main function of the program, `poly_GCD(k)`, will be executed after inputting data. This is its code with explanations in the comments:

```
def poly_GCD(k):
  while len(k) > 1: # set k of more then 1 element
    newk = []
    m = min_degree(k) # find minimal degree of polynomial
    newk.append(m)
    # while there are polynomials in the set
    # take the next polynomial
    for p in k:
      n = replace(m,p)
      if n! = [] and len(n) > 1:
```

```
        new = arrange(n)
      else:
        new = n
      if new! = [] and exppoly(new) > 0:
        newk.append(new)
      elif new! = [] and exppoly(new) == 0:
        print("GCD is 1")
        return [(1,0)]
      elif new == []:
        continue
    k = newk
    print("<=> \n", k)
  print("GCD is: ", k)
  return k
```

This function calls the following auxiliary functions:

• `addpoly(p1,p2)` and `multipoly(p1,p2)` – to add or multiply two polynomials;

• `poly_coefficients(p)` and `poly_exponents(p)` – to list the polynomial's coefficients or exponents;

• `exppoly(p)` – to return the degree of a polynomial;

• `min_degree(s)` – to return a polynomial with minimal degree in the list of polynomials s;

• `replace(p, m)` – to implement the $rep(p, m)$ operation defined above;

• `gcd_2num(x, y)` – to find the GCD of two numbers;

• `gcd_array(l)` – to find the GCD of an array of numbers;

• `div_poly_w_const(a, p)` – to divide the polynomial `p` by the constant `a`;

• `multi_poly_w_const(a, p)` – to multiply the polynomial `p` by the constant `a`;

• `arrange(p)` – to arrange the polynomial `p` in descending order of exponents.

For example the source code of the important function `replace(p, m)` is given below:

```
def replace(m, p):
    newpoly = [[-p[0][0], p[0][1], p[0][2] - m[0][2]]]
    multim = multipoly(newpoly, m)
    newpoly1 = [[m[0][0], m[0][1], 0]]
    multip = multipoly(newpoly1, p)
    fin = addpoly(multim, multip)
    return fin
```

The program for calculating the GCD prints all intermediate results. Let us look at the output from the following examples:

Task. Find the GCD of the following set of polynomials from $R[x]$:

a. $x^3-x^2-4x+4, x^4+3x^3-4x^2-12x, x^4+2x^3-7x^2-8x+12, x^7-16x^3-x^2+4.$

b. $\frac{29}{10}x^2+\frac{297}{20}x+\frac{63}{4}, \frac{61}{10}x^3+\frac{233}{20}x^2+\frac{237}{20}x+\frac{243}{20}, \frac{37}{10}x^3+\frac{341}{20}x^2+\frac{607}{20}x+\frac{393}{20}.$

Solutions:

a. After the polynomials' data has been entered, the set will be stored in the following form:

[[[1,1,3], [-1,1,2], [-4,1,1], [4,1,0]],
[[1,1,4], [3,1,3], [-4,1,2], [-12,1,1]],
[[1,1,4], [2,1,3], [-7,1,2], [-8,1,1], [12,1,0]],
[[1,1,7], [-16,1,3], [-1,1,2], [4,1,0]]]

This is the output we obtain (with intermediate results "drop down" sets) after entering the data and running the function poly GCD:

```
Find GCD of:
[[[1,1,3], [-1,1,2], [-4,1,1], [4,1,0]],
[[1,1,4], [3,1,3], [-4,1,2], [-12,1,1]],
[[1,1,4], [2,1,3], [-7,1,2], [-8,1,1], [12,1,0]],
[[1,1,7], [-16,1,3], [-1,1,2], [4,1,0]]]
<=>
[[[1,1,3], [-1,1,2], [-4,1,1], [4,1,0]],
[[-1,1,3], [4,1,1]], [[1,1,3], [-1,1,2], [-4,1,1], [4,1,0]],
[[1,1,6], [4,1, 5], [-4,1,4], [-16,1,3], [-1,1,2], [4,1,0]]]
<=>
[[[1,1,3], [-1,1,2], [-4,1,1], [4,1,0]],
[[-1,1,2], s[4,1,0]],
[[5,1,5], [-20, 1,3], [-1,1,2], [4,1,0]]]
<=>
[[[-1,1,2], [4,1,0]], [[-1,1,2], [4,1,0]],
[[-1,1,2], [4,1,0]]]
<=>
[[[-1,1,2], [4,1,0]]]
GCDis:[[[-1,1,2],[4,1,0]]]
```

Since we obtained the set with a single polynomial, the GCD is that polynomial i.e. $x^2 - 4$.

b) The polynomial data set is stored in the following form:
[[[29,10,2], [297,20,1], [63,4,0]],
[[61,10,3], [233,20,2], [237,20,1], [243,20,0]],
[[37,10,3], [341,20,2], [607,20,1], [393,20,0]]].

The output with intermediate "drop down" sets obtained after running the function poly GCD:

```
Find GCD of:
[[[29,10,2], [297,20,1], [63,4,0]],
[[61,10,3], [233,20,2], [237,20,1], [243,20,0]],
[[37,10,3], [341,20,2], [607,20,1], [393,20,0]]]
<=>
[[[29,10,2], [297,20,1], [63,4,0]],
[[-284,1,2], [-6171,20,1], [7047,40,0]],
[[-11,1,2], [1487,25,1], [11397,100,0]]]
<=>
[[[29,10,2], [297,20,1], [63,4,0]],
[[1,1,1], [3,2,0]], [[1,1,1], [3,2,0]]]
<=>
[[[1,1,1], [3,2,0]], [[1,1,1], [3,2,0]]]
<=>
[[[1,1,1], [3,2,0]]]
GCDis:[[[1,1,1], [3,2,0]]]
```

The GCD is $x + 3/2$.

6. Discussion and conclusion

It is a challenging task to compute the GCD of a set of many polynomials. Numerous algorithms exist, the majority of which are based on matrix techniques. However, the computational costs associated with these techniques are rather considerable. In this study, we suggested the application of the algebraic properties of the GCD of sets of numerous univariate polynomials in order to compute their GCD in a more efficient way.

The program is completely adequate to the proposed algorithm and might have a wide range of applications, particularly in teaching mathematics and informatics. Applying it allows us to not only observe all intermediate outcomes (also referred as "drop down" sets) in the computation of GCD, but also solve systems of algebraic equations. It may be applied in the process of system elimination using Alfred Tarski's method (Van Den Dries 1988).

The Euclidean algorithms are among the fundamental examples in any serious book on algorithms (Knuth 2014), and the Euclidean algorithm is the special case of our method. The associated problem, which is of importance in some applications, is finding polynomials b_i (termed multipliers) such that $\sum_{i=1}^{n} a_i b_i = g$, where g is the GCD of $P = \{a_1, a_2, \ldots, a_n\}, n > 1$.

The program we provided might be simply modified to compute multipliers simultaneously with the GCD. Furthermore, the proposed program might be of use in creating educational software for the purpose of assisting students in learning the given procedure and checking their outcomes.

REFERENCES

BARNETT, S., 1970. Greatest common divisor of two polynomials. *Linear algebra and its applications*, vol. 3, no. 1, pp. 7 – 9. doi: 10.1016/0024-3795(70)90023-6.

BLANKINSHIP, W.A., 1963. A new version of the Euclidean algorithm. *The American mathematical monthly*, vol. 70, no. 7, p. 742. doi: 10.2307/2312260.

BOITO, P., 2012. *Structured matrix based methods for approximate polynomial GCD (Vol. 15)*. Springer Science+Business Media, New York.

CHRISTOU, D., KARCANIAS, N., MITROULI, M., TRIANTAFYLLOU, D., 2011. Numerical and symbolical methods for the GCD of several polynomials. In *Lecture Notes in Electrical Engineering*, vol. 80. Dordrecht: Springer Netherlands, pp. 123 – 144. doi: 10.1007/978-94-007-0602-6_7

CHRISTOU, D., KARCANIAS, N., MITROULI, M., 2010. The ERES method for computing the approximate GCD of several polynomials. *Applied numerical mathematics: transactions of IMACS*, vol. 60, no. 1 – 2, pp. 94 – 114. doi: 10.1016/j.apnum.2009.10.002.

CHRISTOU, D., MITROULI, M., TRIANTAFYLLOU, D., 2017. Structured matrix methods computing the greatest common divisor of polynomials. *Special Matrices*, vol. 5, no. 1, pp. 202 – 224. doi: 10.1515/spma-2017-0015.

FRYER, W., 1959. Applications of Routh's algorithm to network-theory problems. *IRE transactions on circuit theory*, vol. 6, no. 2, pp. 144 – 149. doi: 10.1109/tct.1959.1086534.

KARCANIAS, N., 1987. Invariance properties, and characterization of the greatest common divisor of a set of polynomials, *International journal of control*, vol. 46, no. 5, pp. 1751 – 1760. doi: 10.1080/00207178708934007.

KNUTH, D. E., 2014. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison Wesley Professional, Boston, MA.

MITROULI, M., KARCANIAS, N., 1993. Computation of the GCD of polynomials using gaussian transformations and shifting, *International journal of control*, vol. 58, no. 10, pp. 211 – 228. doi: 10.1080/00207179308922998.

PACE, I.S., BARNETT, S., 1973. Comparison of algorithms for calculation of g.c.d. of polynomials. *International journal of systems science*, vol. 4, no. 2, pp. 211 – 226. doi: 10.1080/00207727308920007.

PREŠIĆ, S., 1997. *Raznice*. Prosvetni pregled, Beograd. ISBN 86-7055-023-7

VAN DEN DRIES, L., 1988. Alfred Tarski's elimination theory for real closed fields. *Journal of Symbolic Logic*, vol. 53, no. 1, pp. 7 – 19. doi: 10.2307/2274424.

WEINSTOCK, R., 1960. Greatest common divisor of several integers and an associated linear Diophantine equation. *The American mathematical monthly*, vol. 67, no. 7, p. 664. doi: 10.2307/2310105.

✉ **Dr. Verica Milutinović, Assoc. Prof.**
ORCID iD: 0000-0003-0325-7285
Faculty of Education, University of Kragujevac
14, Milan Mijalković St.
Jagodina, Serbia
E-mail: verica.milutinovic@pefja.kg.ac.rs