Numerical methods for computing the greatest common divisor of univariate polynomials using floating point arithmetic

Markus Hegland[1]

(Received 27 February 2019; revised 11 July 2019)

Abstract

Computing the greatest common divisor (GCD) for two polynomials in floating point arithmetic is computationally challenging and even standard library software might return the result GCD = 1 even when the polynomials have a nontrivial GCD. Here we review Euclid's algorithm and test a variant for a class of random polynomials. We find that our variant of Euclid's method often produces an acceptable result. However, close monitoring of the norm of the vector of coefficients of the intermediate polynomials is required.

DOI:10.21914/anziamj.v60i0.14059 gives this article, Published August 15, 2019, as part of the Proceedings of the 18th Biennial Computational Techniques and Applications Conference. ISSN 1445-8810. (Print two pages per sheet of paper.)

Contents

1 Euclid's algorithm

The computation of the greatest common divisor (GCD) of two polynomials is an important problem and has received some attention. In algebraic geometry, the GCD is the simplest example of a Groebner basis. Euclid's algorithm is widely used to compute the GCD using rational arithmetic or finite fields. However, the algorithm may fail when used with floating point arithmetic due to rounding errors. Further background is provided by Stetter [2], Corless et al. [1] and Zeng [3], which contain comprehensive references to further literature. In this literature the connection between Gaussian elimination and Euclid's method is well understood. In Section 2 we derive a matrix factorisation which is based on elimination in the polynomial ring. To the best of my knowledge, this approach is new.

In this section we review Euclid's algorithm and its reliance on long division. We also give an application to the computation of multiple zeros of polynomials and several examples which demonstrate the performance in the floating point context.

Definition 1 (GCD). *A polynomial* g *is a greatest common divisor (*GCD*) of two polynomials* p *and* q *if* g *is a polynomial with maximal degree which*

divides p *and* q.

Note that if g is a GCD of p and q then so is λg for any $\lambda \neq 0$.

The main tool for both theory and computing of the GCD is Euclid's algorithm. To formulate our algorithms we use conventions of the Python module sympy which provides a polynomial type (and class). In sympy the polynomials are represented by an array of monomial coefficients. Euclid's algorithm in Python is stated concisely as follows.

```
def gcd(p, q):
    r = p % q
    if r == 0: return q
    else: return gcd(q, r)
```

At every recursive call gcd(p,q) the degree of the second argument is reduced. The GCD divides the remainder p % q and if the remainder is zero, then p is a multiple of q and thus q is a GCD and Euclid's algorithm terminates. The set of GCDs forms a one-dimensional linear space of polynomials.

The computation of the GCD has many applications where the GCD is often a component of a larger procedure. For example, in numerical analysis the GCD is used to compute multiple zeros of a polynomial. A multiple zero x^* is a number for which both $p(x^*)$ as well as one or several derivatives are zero. In this case the common Newton's method for zeros shows very slow convergence. However, one has the following the proposition due to Lagrange.

Proposition 2 (Lagrange). *Let* $p(x)$ *be a real polynomial of degree* d *with* $m \leqslant d$ *(distinct) zeros and derivative* $p'(x)$. *Then there exists a real number* γ *such that*

$$\frac{p(x)}{\gcd(p(x), p'(x))} = \gamma \prod_{i=1}^{m} (x - x_i),$$

where the x_i *are the (complex) zeros of* $p(x)$.

Thus the zeros of $p(x)$ are equal to the zeros of polynomial $q = p/\gcd(p, p')$. The attraction is that the polynomial $q(x)$ has only simple zeros.

The Python code given above actually works reasonably well for polynomials with rational and integer coefficients. However, trying to work with the operator % in floating point leads to substantial problems. This is why we break down the division of two polynomials into the more basic steps of long division, and recast Euclid's algorithm in terms of these steps. The long division algorithm is as follows.

```
def div(p,q):
    s, r = 0, p
    while (r != 0) & (degree(r) >= degree(q)):
        t = LT(r)/LT(q)
        s = s+t
        r = r-t*q
    return s, r
```

Here, LT(p) denotes the leading (highest degree) monomial term of p.

We now consider some examples. All the computations were performed in 64 bit standard floating point arithmetic. The first example is to confirm that the algorithm performs correctly for a simple case.

Example 3. Consider the two polynomials $p(x) = 10x^4 - 100x^3 + 350x^2 - 500x + 240$ and $q(x) = 4x^2 - 28x + 40$. When running the above version of Euclid's algorithm one gets

$$\gcd(p, q) = 80x - 160,$$

which is correct.

Example 4. Consider the two polynomials $p(x) = 10s^4x^4 - 100s^3x^3 + 350s^2x^2 - 500sx + 240$ and $q(x) = 4s^2x^2 - 28sx + 40$ where $s = 1 + 10^{-10}$. The $\gcd(p, q)$ should be a multiple of $sx - 2$. Running our code gives $\gcd(p, q) = -4.263 \times 10^{-14}$. If the GCD of two polynomials is a constant, then the two polynomials

have no common nontrivial GCD. Our algorithm failed because it was not able to deal with the rounding error.

Example 5. Consider a polynomial $p(x)$ and its derivative $p'(x)$ where $p(x) = (x-1)^4(x-2)^4(x-3)^4(x-4)^4$. The GCD of p and p' is a nontrivial polynomial divisible by $(x-1)^3(x-2)^3(x-3)^3(x-4)^3$. Our algorithm returns $\gcd(p,p') = -1.3460$ which is clearly wrong.

Example 6. Choose $p(x) = (x-X_1)(x-X_2)$ and $q(x) = (x-X_1)(x-X_3)$ where the X_i are chosen to be standard normally distributed random numbers. Clearly the GCD is $x - X_1$ with probability one, but often Euclid's algorithm returns a constant.

The above four examples demonstrate that Euclid's algorithm for polynomials using floating point arithmetic can produce wrong results. Note that when rational (exact) arithmetic is used, then Euclid's algorithm produces the correct result. In order to get correct results with floating point arithmetic one needs to consider rounding errors and use stable algorithms.

One way to model floating point errors is by small random perturbations and thus defining the polynomials to have random coefficients. One can show that the set of pairs of random polynomials with a nontrivial common divisor has measure zero.

2 A modified Euclid algorithm

Here we combine long division with Euclid's algorithm to generate a sequence p_i of polynomials with non-increasing degree. We assume that the degree of p is less than the degree of q. The sequence is initiated by $p_0 = p$ and $p_1 = q$ and satisfies the recursion

$$p_{i+1} = t_i x^{\delta_i} p_i - s_i p_{i-1}\,, \tag{1}$$

where $\delta_i = d_{i-1} - d_i$ and t_i and s_i are such that $d_{i+1} < d_{i-1}$. If some p_i is equal to zero, then we choose all the following polynomials p_{i+k} to be zero.

Here d_i denotes the degree of p_i. Choices for the coefficients t_i and s_i include $t_i = 1$ and $s_i = \operatorname{lc}(p_i)/\operatorname{lc}(p_{i-1})$ where $\operatorname{lc}(p_i)$ denotes the leading coefficient, that is, the coefficient of the highest power x^{d_i} in p_i. This leading coefficient is nonzero unless the polynomial is zero. The choice used here is

$$t_i = \frac{\operatorname{lc}(p_{i-1})}{\max(|\operatorname{lc}(p_i)|, |\operatorname{lc}(p_{i-1})|)}, \quad s_i = \frac{\operatorname{lc}(p_i)}{\max(|\operatorname{lc}(p_i)|, |\operatorname{lc}(p_{i-1})|)}, \tag{2}$$

motivated by the partial pivoting approach in Gaussian elimination. The polynomial p_{i+1} is an instance of an S-polynomial used for multivariate problems where one often uses $t_i = \operatorname{lc}(p_{i-1})$ and $s_i = \operatorname{lc}(p_i)$. The normalisation of t_i and s_i controls the growth of the coefficients of the polynomial p_i. Furthermore, the choice of the coefficients t_i and s_i guarantees that $d_{i+1} < d_{i-1}$. As the degrees are integers one thus has

$$d_{2i} \leqslant d_0 - i \quad \text{and} \quad d_{2i+1} \leqslant d_1 - i. \tag{3}$$

It follows that after $2d_1$ elimination steps one obtains a zero polynomial and thus the algorithm terminates.

Now let the vector $\underline{p} = (p_0, p_1, \ldots, p_n)^T = \sum_{i=0}^{n} p_i e_i$ where e_i is the ith standard basis vector in $\mathbb{R}^{n+1}$. From the definition of p_i in equation (1) and using vector notation one then obtains (I is the identity matrix)

$$\underline{p} = (I + t_{n-1} x^{\delta_{n-1}} e_n e_{n-1}^T - s_{n-1} e_n e_{n-2}^T) \cdots (I + t_1 x^{\delta_1} e_2 e_1^T - s_1 e_2 e_0^T)(p e_0 + q e_1). \tag{4}$$

This formula is very similar to formulas occurring in Gaussian elimination, except that now the matrices may contain powers of x. As for Gaussian elimination, one now defines the lower triangular matrix $L_n(x)$ with nonzero components

$$l_{i,j}(x) = \begin{cases} 1 & \text{if } j = 1, \\ -t_{i-1}\, x^{\delta_{i-1}} & \text{if } j = i-1, \\ s_{i-1} & \text{if } j = i-2. \end{cases} \tag{5}$$

Then it follows that for all n,

$$p_n(x) = e_n^T L_n^{-1}(x) e_0\, p(x) + e_n^T L_n^{-1} e_1\, q(x), \tag{6}$$

that is, one has $p_n(x) = a_n(x)p(x) + b_n(x)q(x)$, which proves that the polynomials $p_n(x)$ are all in the ideal generated by $p(x)$ and $q(x)$ in a constructive way. From equation (6) it then follows that $p_n(x)$ is a linear combination of polynomials of the form $x^k p(x)$ and $x^j q(x)$. A careful inspection then shows that for $\underline{x} = (1, x, x^2, \ldots, x^n)^T$ one has

$$\underline{p} = M_n S \underline{x}\,, \tag{7}$$

where S is the Sylvester matrix and M_n is a matrix which, like the Sylvester matrix, does not contain any powers of x.

When using rational arithmetic the above algorithm computes the GCD of p and q, which is equal to the polynomial p_n for which $p_{n+1} = 0$. When utilising the method with floating point arithmetic, rounding errors may provide a substantial challenge. Euclid's method is basically Gaussian elimination with a somewhat predefined pivoting procedure. The computations may be affected by errors in the input data p and q, which typically perturb the data such that the GCD of the perturbed data is one. But even if there is still a nontrivial GCD, the rounding errors in the Euclidean algorithm may destroy any common divisors of the polynomials p_n and thus also not produce a GCD. We found that often monitoring the coefficient norm of the polyomials p_n indicates when a p_n is an approximate GCD. This is the case when the norm of the coefficients of p_{n+1} is small. Section 4 provides some examples to illustrate the algorithm.

We retested Examples 3–6 with the new algorithm presented above and it was able to compute the GCDs in all cases. Especially for the GCD of p and its derivative (Example 5), we observed behaviour similar to the random polynomials discussed in Section 4.

3 The Sylvester matrix

Let the matrices $S_1 \in \mathbb{R}^{d_2, d_1+d_2}$ and $S_2 \in \mathbb{R}^{d_1, d_1+d_2}$ be two matrices which satisfy

$$\begin{bmatrix} x^{d_2-1} \\ \vdots \\ 1 \end{bmatrix} p(x) = S_1 \begin{bmatrix} x^{d_1+d_2-1} \\ \vdots \\ 1 \end{bmatrix}, \tag{8}$$

and similarly for S_2. We then denote the Sylvester matrix as

$$S = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}. \tag{9}$$

Here d_1 and d_2 are the degrees of p and q, respectively. The blocks S_i are banded Toeplitz matrices.

Euclid's algorithm is an elimination algorithm for the Sylvester matrix which maintains the Toeplitz structure and eliminates whole diagonals at a time. This elimination process is illustrated in the following diagram where zero elements are not displayed and elements modified are denoted by '+' while the elements unchanged are denoted by '$*$'. Consider the case $d_1 = 2$ and $d_2 = 4$ for which one has

$$S = \begin{bmatrix} * & * & * & & & \\ & * & * & * & & \\ & & * & * & * & \\ & & & * & * & * \\ * & * & * & * & * & \\ & * & * & * & * & * \end{bmatrix}. \tag{10}$$

The first four rows define the block S_1 and the last two rows define S_2. Both blocks are Toeplitz, that is, constant along the diagonals. In the first step one subtracts a multiple of the first row from row five in order to eliminate the element s_{51}—as in Gaussian elimination. In contrast to Gaussian elimination, one then subracts the same multiple of the second row from the last row which

eliminates element s_{62}. As both blocks are Toeplitz this does not require extra work as the new last row is just a shifted version of the second last row. Now the first row and column are no longer required and we just display the nonzero structure of the reduced matrix $S^{(1)}$. This elimination step is then repeated until only one row in the upper block remains, producing $S^{(2)}$ and $S^{(3)}$ in the process. The same elimination process is then performed on the two blocks of $S^{(3)}$. The whole process is repeated until no more eliminations are required. This is just Euclid's algorithm in matrix notation and for our example we produced $S^{(0)} \to S^{(1)} \to S^{(2)} \to S^{(3)} \to S^{(4)} \to S^{(5)} \to S^{(6)}$ with the nonzero structures

$$\begin{bmatrix} * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * & * \\ * & * & * & * & * & \\ & * & * & * & * & * \end{bmatrix} \to \begin{bmatrix} * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ + & + & * & * & \\ & + & + & * & * \end{bmatrix} \to \begin{bmatrix} * & * & * & \\ & * & * & * \\ + & + & * & \\ & + & + & * \end{bmatrix}$$

$$\to \begin{bmatrix} * & * & * \\ + & + & \\ & + & + \end{bmatrix} \to \begin{bmatrix} + & + & \\ & + & + \\ * & * & * \end{bmatrix} \to \begin{bmatrix} + & + \\ + & + \end{bmatrix} \to \begin{bmatrix} + \end{bmatrix}. \tag{11}$$

Here we have omitted the pivot row and column after each elimination step as it is not required for computing the GCD. The matrix $S^{(4)}$ is obtained from $S^{(3)}$ by moving the first row to the last position. This is consistent with the standard version of Euclid's algorithm. The matrices $S^{(i)}$ are all Sylvester matrices related to the polynomials $p^{(i)}$ which occur in Euclid's algorithm. The algorithm terminates at step i when the matrix $S^{(i)}$ is zero. The GCD is then retrieved from $S^{(i-1)}$.

A big advantage of the Sylvester matrix approach is that robust matrix methods including Gaussian elimination with pivoting, QR factorisation with column pivoting, SVD and even parallel algorithms can be used to factorise this matrix and reveal the GCD. However, these advantages have a substantial computational overhead as the coefficients of p and q are stored multiple

times. The basic idea in Sylvester's approach is to group the whole GCD computation into one big matrix task. We are currently working on new methods which groups computational tasks during Euclid's algorithm into several subtasks which can then be processed more efficiently and stably. A first step is the formulation of the matrix-based Euclid method discussed in Section 2 and an experimental framework for random polynomials.

Efficiency and stability are further enhanced by using sparse matrices for sparse polynomials, and in that context setting small elements to zero during the computations (thresholding).

4 Experiments with stochastic polynomials

We now investigate the algorithm introduced in Section 2 for randomly generated polynomials p and q. We first explain in detail how the polynomials are generated before providing two illustrative samples.

In order to test the accuracy of the algorithm we construct polynomials with known GCD. This is achieved by choosing p and q to be products of polynomials $\pi_1(x), \pi_2(x)$ and $\pi_3(x)$, that is, $p(x) = \pi_1(x)\pi_2(x)$ and $q(x) = \pi_1(x)\pi_3(x)$. When constructing the factors π_i, first the degrees of the π_i are drawn from integers between 1 and 30 using a uniform distribution. Then the coefficients of the π_i are chosen uniformly from the set $\{i/k \mid i = -k, \ldots, k\}$. For the parameter k we consider both $k = 4096$ and $k = 4000$. We check that π_1 is indeed a GCD of p and q (which is true with high probability). The difference between the two experiments is that for $k = 4096$ the coefficients of the polynomial factors π_i can be exactly represented in floating point arithmetic. This is not the case for $k = 4000$. However, in both algorithms the actual GCD computations do incur rounding errors.

The sequence of polynomials $p_i(x)$ generated by our variant of Euclid's algorithm for the polynomials p and q is determined using Python and in particular the Polynomial class of sympy. The polynomials $p_i(x)$ are both

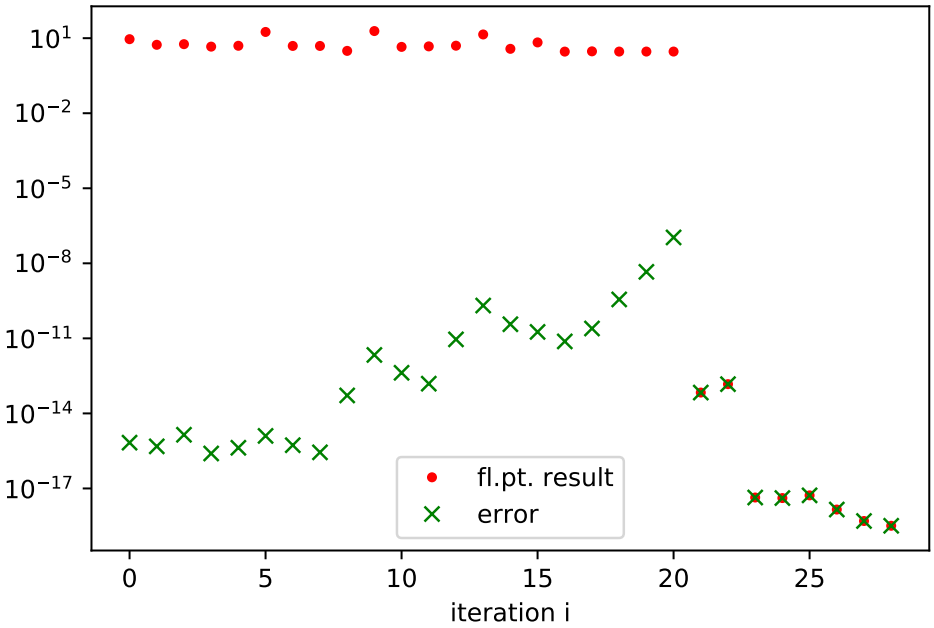

Figure 1: norm of coefficients of p_i and its difference to the reference solution (case $k = 4096$).

computed in floating point arithmetic and, as reference, exactly in rational arithmetic. Figures 1 and 2 plot the l_2 norm of the coefficients of the p_i and of the coefficients of the difference between the floating point p_i and the rational reference solution for $k = 4096$ and $k = 4000$, respectively. In the first example (Figure 1) the floating point sympy library code 'GCD' returns an accurate GCD. However, in the second example (Figure 2), the library code returns $\text{GCD} = 1$. This is reasonable as p and q have an exact GCD of one due to the floating point errors of the rounded coefficients— note the high rounding error of the new algorithm in Figure 2. We intend to perform future systematic studies of the rounding error growth for this case.

One of the most robust numerical GCD solvers available is based on the singular value decomposition (SVD) of the Sylvester matrix [1]. We have

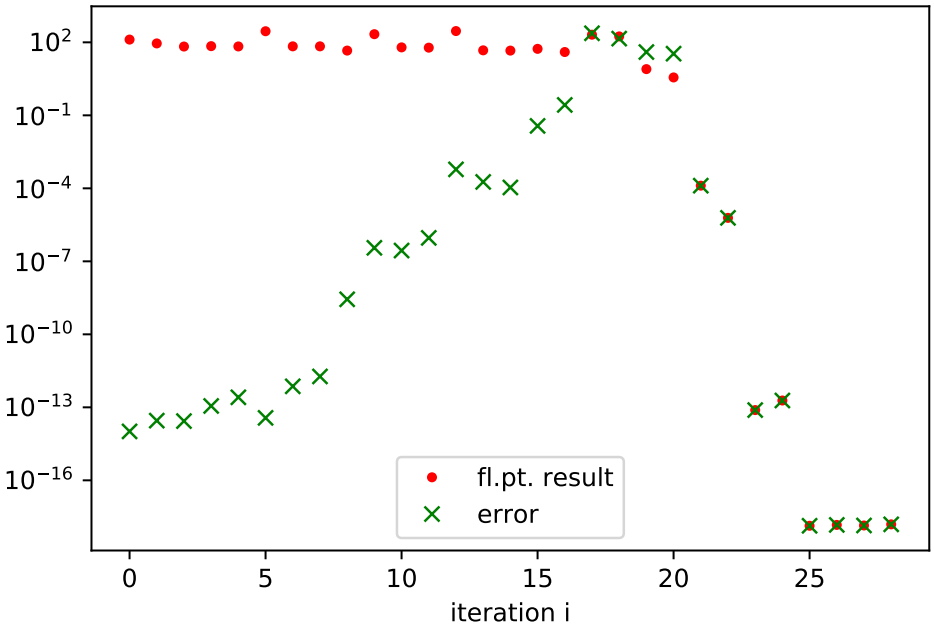

Figure 2: norm of coefficients of p_i and its difference to the reference solution (case $k = 4000$).

applied our approach to the problems considered by Corless et al. [1] and were able to reproduce their results at a much lower cost (see Section 5).

5 Outlook

While stable methods based on the SVD of the Sylvester matrix work well for the numerical computation of the GCD, they are expensive as the SVD requires $O(n^3)$ floating point operations for computing the GCD of two polynomials of degree n. This is in contrast to Euclid's algorithm which has complexity of $O(n^2)$ for the same problem. Experiments with random polynomials suggest that a stable variant of the classical Euclid algorithm based

on monitoring the size of the remainders often works reasonably well while maintaining the same complexity order as the original Euclidean algorithm.

In future work we intend to study methods which aggregate multiple steps of Euclid's algorithm. These methods display scope for stability and efficiency improvements. The methods based on the Sylvester matrix are an extreme version where all the steps are treated as one big elimination problem. Our planned approach balances speed and stability.

References

[1] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. "The singular value decomposition for polynomial systems". In: *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*. ISSAC '95. Montreal, Quebec, Canada: ACM, 1995, pp. 195–207. DOI: 10.1145/220346.220371 (cit. on pp. C128, C137, C138).

[2] H. J. Stetter. *Numerical polynomial algebra*. SIAM, 2004. DOI: 10.1137/1.9780898717976 (cit. on p. C128).

[3] Z. Zeng. "The numerical greatest common divisor of univariate polynomials". In: *Randomization, relaxation, and complexity in polynomial equation solving*. Vol. 556. Contemp. Math. Amer. Math. Soc., 2011, pp. 187–217. DOI: 10.1090/conm/556/11014 (cit. on p. C128).

Author address

1. **Markus Hegland**, Mathematical Sciences Institute, ANU, Australia
mailto:markus.hegland@anu.edu.au
orcid:https://orcid.org/0000-0002-5136-2883