Computing GCDs of polynomials over univariate quotient rings

John Kluesner. Mathematics Graduate Student, Simon Fraser University, British Columbia.

GCDs of polynomials over algebraic number fields

Let $L = \mathbb{Q}(\alpha)$ where α is an algebraic number; that is, there is an irreducible monic polynomial $m(z) \in \mathbb{Q}[z]$ such that $m(\alpha) = 0$. Given $a, b \in L[x]$, we'd like to compute $\gcd(a, b)$. Since L is a field, the Euclidean algorithm will work. For example, let $\alpha = \sqrt{2}$ with $a = 2x^4 - x^3 - 3x^2 + (14\alpha - 2)\,x - \alpha + 1$ and $b = 4\alpha x^3 + (\alpha - 1)\,x^2 + (\alpha - 4)\,x - \alpha - 1$. The Euclidean algorithm proceeds by repeatedly computing remainders: set $r_1 := a$, $r_2 := b$, and then

$$r_3 := \left(\frac{3}{4}\alpha - \frac{49}{16}\right)x^2 + \left(\frac{215}{16}\alpha - \frac{7}{8}\right)x - \frac{9}{8}\alpha + \frac{11}{16},$$
$$r_4 := \left(\frac{932781440}{4464769}\alpha + \frac{745931632}{4464769}\right)x - \frac{62941056}{4464769}\alpha - \frac{25830208}{4464769},$$
$$r_5 := \frac{3785009996727209}{1751196256378909 52}\alpha + \frac{23782663312092435}{350239251275781904},$$

which shows $\gcd(a, b) = 1$ since r_5 is a unit. The problem is clear: the coefficients explode. This is typical for the Euclidean algorithm — it's not a special case.

In [1] Lars Langemyr, Scott McCallum, and [2] Mark Encarnacion gave new algorithms for computing the GCDs of polynomials over algebraic number fields. Their algorithms are applications of the ideas of Brown and Collins [3]. Given $a, b \in \mathbb{Q}(\alpha)[x]$, compute $\gcd(a, b)$ by reconstructing it from modular images $\gcd(a \mod p, b \mod p) \in \mathbb{Z}_p(\alpha)[x]$ for sufficiently many primes p. Since the coefficients are bounded in size in $\mathbb{Z}_p(\alpha)[x]$, it circumvents coefficient explosion. Additionally, Michael Monagan and Mark Hoeij [4] gave algorithms for computing over multiple algebraic numbers, such as $\mathbb{Q}(\sqrt{2}, \sqrt{3})$.

GCDs of polynomials over univariate quotient rings

The computational model for an algebraic number field $\mathbb{Q}(\alpha)$ is $\mathbb{Q}[z]/m(z)$. With that in mind, a natural generalization is to consider $m(z)$ that is reducible. This comes up naturally when solving systems of polynomial equations as well.

Questions: Let $R = \mathbb{Q}[z]/m(z)$.

- Does $\gcd(a, b)$ exist for all $a, b \in R[x]$?
- Will the Euclidean algorithm still work?
- Can we generalize the modular algorithms above?

Theorem 1. If $m(z)$ is square-free, then $\gcd(a, b)$ exists for all $a, b \in R[x]$.

The next step is to develop an algorithm to compute it. To use the Euclidean algorithm, some form of the division algorithm is required.

Theorem 2. Suppose $m(z)$ is square-free, and $a, b \in R[x]$ where $b \neq 0$. Then, there exists q, r where $\deg(r) < \deg(b)$ and $a = qb + r$. Such r and q are unique if and only if $\mathrm{lc}(b)$ is a unit. In the latter case, the standard division algorithm can compute r and q.

The last sentence of the prior theorem is important: if $\mathrm{lc}(b)$ isn't a unit, the standard division algorithm can not be used. Instead, pass to smaller rings if a non-unit is encountered; see Algorithm 1.

The value of $m(z)$ being square-free should be apparent. From here out, this will be assumed.

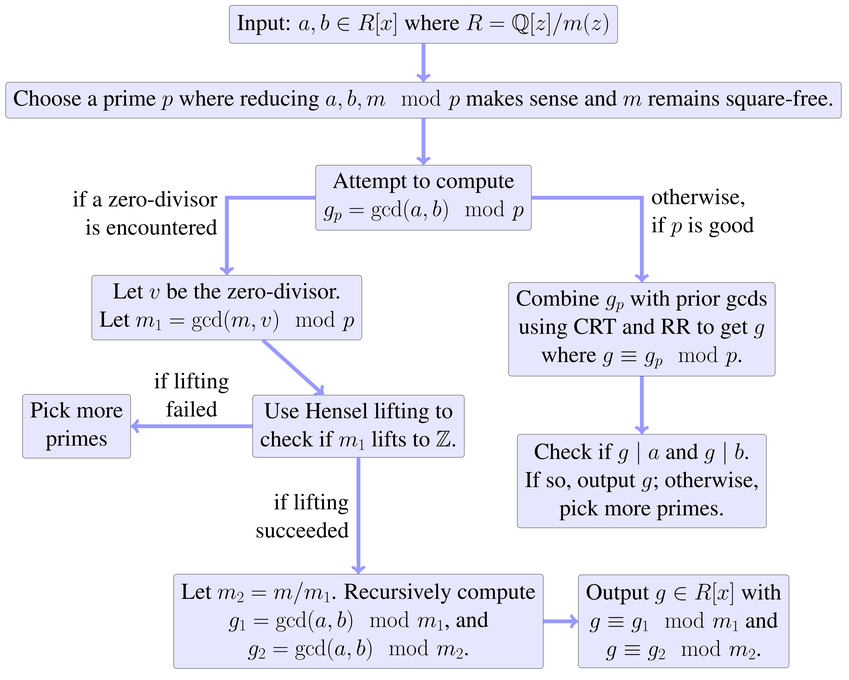
Algorithm 1: The Euclidean-style GCD algorithm.

1. Set $r_1 := a$ and $r_2 := b$ and $i := 2$. Go to step 2.
2. Attempt to compute the remainder r of r_{i-1} divided by r_i. If successful, go to step 3. If a zero-divisor is encountered, go to step 4.
3. Set $r_{i+1} := r$ and $i := i + 1$. If $r_i = 0$, return r_{i-1}; Else, go to step 2.
4. Let v be the zero-divisor. Set $m_1 := \gcd(v, m)$ and $m_2 := m/m_1$. Recursively compute $g_1 := \gcd(r_i, r_{i-1}) \mod m_1$ and $g_2 := \gcd(r_i, r_{i-1}) \mod m_2$. Combine results using CRT to get $g \in R[x]$ where $g \equiv g_i \mod m_i$. Return g.

However, this algorithm, being a variant of the Euclidean algorithm, will suffer from coefficient explosion. Borrowing the ideas of Brown and Collins, a modular algorithm is desired.

It is possible that $g_p = \gcd(a, b) \mod p$ isn't the modular image of $g = \gcd(a, b)$. If it is though, we call such a prime a good prime. We can detect if a prime is good. For the algorithm to be viable, it is necessary for "most" primes to be good. This is indeed the case.

Algorithm 2: The modular GCD algorithm.



Theorem 3. Let $a, b \in R[x]$. Suppose the Euclidean algorithm doesn't encounter any zero-divisors. All but finitely many primes are good primes.

The assumption that the Euclidean algorithm doesn't encounter any zero-divisors may seem to make this result weak. However, if the Euclidean algorithm does encounter a zero-divisor, that gives a factor m_1 of m modulo p. We can then check if m_1 lifts to a factor of m over $\mathbb{Q}$ using Hensel lifting. If it does, recursively run the algorithm similarly to the naive algorithm above. If it doesn't, simply discard the prime. The number of primes that will be discarded in this fashion can also be proven to be finite.

Timings and Maple implementation

I implemented the Euclidean-style and modular algorithms in Maple. I tested them on dense randomly-generated polynomials $a = \overline{a}g$ and $b = \overline{b}g$ where $\deg(\overline{a}) = \deg(\overline{b}) = 2\deg(g)$. The modulus $m(z)$ was the product of 3 randomly-generated irreducibles of degrees 1, 2, and 3. The following table shows the results:

degree	CPU time for modular	CPU time for Euclidean
30	5.377	2.616
45	12.067	9.681
60	13.273	7.446
75	35.495	53.539
90	51.663	84.928
105	48.858	100.065
120	74.999	—

Timings for computing $g = \gcd(a, b)$ where $\deg(g) \geq \deg(a)/3$. The last test for the Euclidean algorithm ran for over 200 seconds of CPU time when halted.

It is also important to test with irreducible $m(z)$ versus Encarnacion's. If a $\gcd(a, b) \mod m(z)$ is needed and the reducibility of $m(z)$ is unconfirmed, it'd be nice to be able to run my new modular algorithm without any fear of overhead. I generated polynomials the same way as above and obtained the following results:

degree	CPU time for modular	CPU time for Encarnacion
30	3.601	3.624
45	10.773	10.816
60	21.372	21.294
75	35.098	35.519
90	50.823	50.956
105	57.227	57.187
120	64.345	66.241

Timings for computing $g = \gcd(a, b)$ where $\deg(g) \geq \deg(a)/3$.

Conclusions

I proved sufficient conditions for the existence of GCDs in $(\mathbb{Q}[z]/m(z))[x]$. I developed new and efficient algorithms for computing these GCDs, and implemented them in Maple.

Acknowledgements

I would like to thank my supervisor Dr. Michael Monagan for tailor-making this project for me as part of his Topics in Computer Algebra course.

References

[1] Lars Langemyr, Scott McCallum. The computation of polynomial greatest common divisors over algebraic number fields. *J. Symbolic Computation* **8**, (1989) 429–448.

[2] Mark Encarnacion. Computing GCDs of Polynomials over Algebraic Number Fields. *J. Symbolic Computation* **20** (1995) pp. 299–313.

[3] W.S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18** (1971), pp. 476–504.

[4] Mark Hoeij, Michael Monagan. A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *Proceedings ISSAC '2002*, ACM Press (2002), pp. 109–116.