Applications of Computer Algebra – ACA 2022
Gebze-Istanbul, Turkey, | August 15-19, 2021
Session on *"Session Title"*

On computing isomorphisms between algebraic number fields

Michael Monagan[1] [mmonagan@sfu.ca]

[1] Department of Mathematics, Simon Fraser University, Vancouver, Canada

Let $K = \mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ be an algebraic number field. For example $K = \mathbb{Q}(\sqrt{2}, \sqrt{3})$. Then K is a vector space over $\mathbb{Q}$. Let $d = \dim(K : \mathbb{Q})$. Without loss of generality we assume $\mathbb{Q}(\alpha_1, \ldots, \alpha_i)$ is a proper subfield of $\mathbb{Q}(\alpha_1, \ldots, \alpha_i, \alpha_{i+1})$ for $1 \le 1 < k$.

Let $c_1, c_2, \ldots, c_k$ be integers and let $\gamma = \sum_{i=1}^{k} c_i\, \alpha_i$. For almost all c_i we have $K \simeq \mathbb{Q}(\gamma)$. In this work we want to compute the field isomorphism $\varphi : K \to \mathbb{Q}(\gamma)$ as fast as possible.

Our motivation is the modular GCD algorithm of van Hoeij and Monagan from [3]. For two polynomials $A, B \in K[x]$ their algorithm computes $G = \gcd(A, B)$ modulo a sequence of primes $p_1, p_2, \ldots$, then applies the Chinese remainder theorem to compute G modulo m where m is the product of primes, and then uses Wang's rational number reconstruction from [4] to recover the rational coefficients of G from their images modulo m. The speed of their algorithm depends on the speed of arithmetic in K modulo a prime p.

How do we represent the elements of K and K mod p and how do we do arithmetic in K and in K mod p? The approach taken by the computer algebra systems Pari and Maple is to construct K as a sequence of quotients (see below) and use a recursive polynomial data structure to represent the elements of K.

> Set $K_0 = \mathbb{Q}$.
> For $i = 1$ to k do
>> Let $m_i(z_i)$ be the minimal polynomial for α_i over K_{i-1} and let $d_i = \deg(m_i, z_i)$.
>> Set $K_i = K_{i-1}[z_i]/\langle m_i \rangle$.

We have $K \simeq K_k$ and $d = \prod_{i=1}^{k} d_i$. Also K is isomorphic to the quotient ring $R = \mathbb{Q}[z_1, \ldots, z_k]/I$ where I is the ideal $\langle m(z_1), \ldots, m(z_k) \rangle$.

One way to do arithmetic in R would be to represent elements of R as sparse multivariate polynomials in $\mathbb{Q}[z_1, z_2, \ldots, z_k]$ and use Gröbner bases. We have $\{m_1, m_2, \ldots, m_k\}$ is a Gröbner basis for I in lexicographical order with $z_1 < z_2 < \cdots < z_k$. However, this is expensive as a multiplication in R will do many multivariate polynomial operations.

Pari represents multivariate polynomials recursively, that is, Pari thinks of a polynomial in $\mathbb{Q}[z_1, z_2, \ldots, z_k]$ as a polynomial in $\mathbb{Q}[z_1][z_2]\cdots[z_k]$ and it uses a dense recursive polynomial data structure so that it needs univariate polynomial arithmetic only. Inspired by Pari's representation, van Hoeij and Monagan [3] also used a dense recursive representation for polynomials for their Maple implementation of the modular GCD algorithm in $K[x]$. For example, the polynomial $7x^2 + 5z_2^2 + 3z_1^2$ in $\mathbb{Q}[z_1][z_2][x]$ is stored as the Maple list of lists of lists of integers `[[[0,0,3],0,[5]],0,[[7]]]`.

We have observed that when $k > 1$ and m_1 has low degree, which is often the case practice, it is faster (typically 5 to 10 times faster) to multiply in $\mathbb{Q}(\gamma)$ mod p than to multiply in K mod p. One reason for this is that to multiply in K_3 mod p we do many multiplications in K_2 mod p, each of which does many multiplications in K_1, each of which requires memory to be allocated for the intermediate product and several function calls. This overhead is minimized when $k = 1$. In our talk we will present timing data to measure the overhead in Pari, Maple and Magma. **Thus our hypothesis:** to compute $\gcd(A, B)$ mod p, for $\deg(A, x)$ and $\deg(B, x)$ sufficiently large, it should be faster if we first compute φ mod p and map the GCD computation from K mod p into $\mathbb{Q}(\gamma)$ mod p.

How do we compute the isomorphism $\varphi : K \to \mathbb{Q}(\gamma)$? In our talk we present three methods (sketched below) to compute φ. The first method uses Gröbner bases, the second uses Linear Algebra, and the third uses iterated resultants. We have implemented the second method in C modulo a prime p. Our C implementation uses a dense recursive representation for elements of K mod p and supports primes up to 63 bits. We present timings for computing GCDs in $K[x]$ mod p comparing Pari, Magma, and Maple with our C code.

Method 1: Gröbner Bases.

Let $\gamma = \sum_{i=1}^{k} c_i z_i$ and let $m(z)$ be the minimal polynomial for γ over $\mathbb{Q}$. Let

$$F = [m_1(z_1), \ldots, m_k(z_k), z - \gamma]$$

and let G be the reduced Gröbner basis for F in lexicographical order with $z < z_1 < \cdots < z_k$. For almost all c_i we have $G \cap \mathbb{Q}[z] = \{m(z)\}$ and the remaining elements of G give us $\varphi(z_i)$. We give an example to illustrate.

Example 1. For $K = \mathbb{Q}(\sqrt{2}, \sqrt{3})$ we have $m_1(z_1) = z_1^2 - 2$ and $m_2(z_2) = z_2^2 - 3$ and a basis for K over $\mathbb{Q}$ is $[1, z_1, z_2, z_1 z_2]$. For $c_1 = c_2 = 1$ we have $\gamma = z_1 + z_2$ and $F = [z_1^2 - 2, z_2^2 - 3, z - z_1 - z_2]$. We obtain the Gröbner basis

$$G = [z^4 - 10z^2 + 1, z_1 + \tfrac{9}{2}z - \tfrac{1}{2}z^3, z_2 - \tfrac{11}{2}z + \tfrac{1}{2}z^3].$$

Thus $m(z) = z^4 - 10z^2 + 1$, $\varphi(z_1) = -\frac{9}{2}z + \frac{1}{2}z^3$ and $\varphi(z_2) = \frac{11}{2}z - \frac{1}{2}z^3$. We have $\varphi(1) = 1$ and we compute $\varphi(z_1 z_2) = \varphi(z_1)\varphi(z_2)$.

Notice that F is also a Gröbner basis for the ideal generated by F in lexicographical order with $z_1 < z_2 < \cdots < z_k < z$ because the leading monomials of the polynomials in F are $z_1^{d_1}, z_2^{d_2}, \ldots, z_k^{d_k}$ and z which are all relatively prime! Therefore, we may compute G from F using FGLM, the Gröbner basis conversion algorithm of Faugere, Gianni, Lazard and Mora [2]. The FGLM algorithm does $O(kd^3)$ arithmetic operations in $\mathbb{Q}$.

Method 2: Linear Algebra.

The number field $K = \mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ is a vector space over $\mathbb{Q}$. Let $d = \dim(K : \mathbb{Q})$ and let $m(z) = z^d + \sum_{i=0}^{d-1} x_i z^i$ be the minimal polynomial for γ over $\mathbb{Q}$ for x_i unknown. Equating $m(\gamma) = 0$ we obtain a linear system $\sum_{i=0}^{d-1} x_i \gamma^i = -\gamma^d$. In matrix form we have $Ax = b$ where $A = [\,1\,|\,\gamma\,|\,\gamma^2\,|\ldots|\,\gamma^{d-1}\,]$ and $b = -\gamma^d$. We construct A then invert A and obtain x from $x = A^{-1}b$. The matrix A^{-1} is the mapping $\varphi : K \to \mathbb{Q}(\gamma)$ thus A gives us φ^{-1}. Method 2 does $O(d^3)$ arithmetic operations in $\mathbb{Q}$.

Method 3: Iterated Resultants.

Let $\gamma = \sum_{i=1}^{k} c_i z_i$. Starting with the polynomial $z - \gamma$ we use the subresultant algorithm (see [4]) to first use m_k to eliminate z_k then to use m_{k-1} to eliminate z_{k-1}, etc., until we have eliminated all z_i and we obtain the minimal polynomial $m(z)$. In a second stage we successively obtain $\varphi(z_1)$, $\varphi(z_2)$, ..., $\varphi(z_k)$ using the penultimate polynomials in the subresultant remainder sequences which are linear for almost all c_i.

Example 1 (continued). First we apply the subresultant algorithm to $z - z_1 - z_2$ and $z_2^2 - 2$ to eliminate z_2. We obtain 3 polynomials $z_2^2 - 2$, $z - z_1 - z_2$ (which is linear in z_2) and $-2zz_1 + z^2 + 1$. Next we apply the subresultant algorithm to $-2zz_1 + z^2 + 1$ and $z_1^2 - 3$ to eliminate z_1. We obtain 3 polynomials $z_1^2 - 3$, $-2zz_1 + z^2 + 1$ (which is linear in z_1) and $z^4 - 10z^2 + 1$ (the minimal polynomial for γ).

Now we compute $\varphi(z_1)$ by solving $-2zz_1 + z^2 + 1 = 0$ for z_1 mod $m(z)$. We must invert $-2z$ in $\mathbb{Q}[z]/\langle m(z)\rangle$ using he Euclidean algorithm. We then solve $z - \varphi(z_1) - z_2 = 0$ for z_2 to determine $\varphi(z_2)$. Finally we compute $\varphi(z_1 z_2) = \varphi(z_1)\varphi(z_2)$.

Method 3 also does $O(d^3)$ arithmetic operations in $\mathbb{Q}$. But unlike methods 1 and 2 which solve linear systems of size $d \times d$, it only does polynomial arithmetic. We are currently investigating whether we can accelerate method 3.

Keywords
Grobner Bases, Algebraic number fields, Polynomial GCD, Field isomorphisms, Resultants

References

[1] B. BUCHBERGER, G.E. COLLINS, R. LOOS, R. ALBRECHT. *Computer Algebra*. Springer, 1983.
[2] J.C. FAUGERE, P. GIANNI, D. LAZARD, T. MORA. Efficient Computation of Zero-dimensional Gröbner Bases by Change of Ordering. *J. Symb. Comp.* **16**(4), 329–344 (1993).
[3] M. VAN HOEIJ, M. MONAGAN., A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. In *Proceedings of ISSAC '02*, 109–116. ACM, 2002.
[4] P. WANG. A p-adic algorithm for univariate partial fractions. In *Proceedings of SYMSAC '81*, 212–217, ACM, 1981.