Computing over algebraic number fields

Michael Monagan

Department of Mathematics, Simon Fraser University
mmonagan@sfu.ca

Let $K = \mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ denote the algebraic number field containing k algebraic numbers $\alpha_1, \alpha_2, \ldots, \alpha_k$. So K is the smallest field containing $\mathbb{Q}, \alpha_1, \ldots, \alpha_k$. Suppose we are given two non-zero polynomials f_1 and f_2 in $K[x]$ and we want to compute their monic GCD $g = \gcd(f_1, f_2)$.

For $k=1$, Encarnacion [3] devised a modular GCD algorithm (see [2]) which computes g modulo a sequence of primes $p_1, p_2, \ldots, p_j$ using the monic Euclidean algorithm, then applies the Chinese remainder theorem to recover g mod M where $M = \prod_{i=1}^{j} p_i$, then recovers the rational coefficients of g using rational number reconstruction (see [6]).

In [4] van Hoeij and Monagan extended Encarnacion's algorithm for $k > 1$. Their algorithm is implemented in Pari and Maple. Both implementations construct the number field K by building a tower of k field extensions and they compute in that tower modulo primes.

Let $c_1, c_2, \ldots, c_k$ be non-zero integers and let $\gamma = \sum_{i=1}^{k} c_i \alpha_i$. It is known (see [2]) that for all but finitely many c_i, the number fields K and $\mathbb{Q}(\gamma)$ are isomorphic. In [1], Ansari and Monagan speed up a GCD computation over K mod p by converting it to a GCD computation over $\mathbb{Q}(\gamma)$ mod p. The goal of our work here is to study the relative efficiency of computing in the tower K mod p verses computing in $\mathbb{Q}(\gamma)$ mod p for a prime p.

We first show how Maple and Pari compute in K and K mod p for a prime p and then we compare the cost of computing in K mod p with computing in $\mathbb{Q}(\gamma)$ mod p. To speed up computation in K mod p we have developed our own C library which supports in place arithmetic in K mod p for $p < 2^{63}$ and $k \geq 1$. We then give a polynomial GCD benchmark which compares the speed of Magma V2.26-12, Pari 2.13.3, Maple 2022, and our C library that illustrates the cost of computing in K mod p verses $\mathbb{Q}(\gamma)$ mod p and the speed of our C library.

To compute in $K = \mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$, Maple and Pari construct the number field K as a tower of n field extensions as follows.

1 Set $L_0 = \mathbb{Q}$.
2 For $i = 1$ to k do
  2a Let $m_i(z_i)$ be the minimal polynomial for α_i over L_{i-1}.
  2b Set $L_i = L_{i-1}[z_i]/\langle m_i \rangle$.
3 Set $L = L_k$ and $d = \prod_{i=1}^{k} d_i$ where $d_i = \deg(m_i, z_i)$.

We have $K \cong L$ so to compute in K we simply replace α_i with z_i and compute in the tower L. The integer d is the degree of the number field L over $\mathbb{Q}$. Also $B = \{\prod_{i=1}^{k} z_i^{e_{ij}} : 0 \leq e_{ij} < d_i\}$ is a basis for L over $\mathbb{Q}$ of dimension d. For $a \in L$ let $[a]_B : L \to \mathbb{Q}^d$ denote the co-ordinate vector of a wrt the basis B and let $v^B : \mathbb{Q}^d \to K$ where $v^B = \sum_{i=1}^{d} v_i B_i$ denote the inverse mapping.

Example 1. Consider $\mathbb{Q}(\sqrt{2},\sqrt{3})$. Here $m_1(z_1) = z_1^2 - 2$, $m_2(z_2) = z_2^2 - 3$, $d_1 = 2, d_2 = 2, d = 4$, $B = \{1, z_1, z_2, z_1z_2\}$ and $[1,2,3,4]^B = 1 + 2z_1 + 3z_2 + 4z_1z_2$. In Maple, the element $a = (a_0 + a_1z_1) + (b_0 + b_1z_1)z_2$ in L_2 is represented as the Maple list of lists $[[a_0, a_1], [b_0, b_1]]$. Maple lists are read only arrays.

Maple and Pari both compute in L_i using univariate polynomial arithmetic, that is, to multiply $a, b \in L_i$, for $i > 0$, they first compute $c = a \times b$ in $L_{i-1}[z_i]$ then divide c by m_i in $L_{i-1}[z_i]$. Thus each multiplication in L_i does many multiplications in L_{i-1}. The following theorem counts the multiplications in L_0.

Theorem 1. *Let $a, b \in L$. If classical multiplication and division in $L_i[z_i]$ is used, the number of multiplications in the ground field $L_0 = \mathbb{Q}$ done to multiply $a \times b$ in L is $M(k) = \prod_{i=1}^{k} \left(d_i^2 + d_i(d_i - 1)\right) = \prod_{i=1}^{k} d_i(2d_i - 1)$.*

The following theorem (see [1]) says that instead of computing in L of degree d with k field extensions, we can compute in $\widehat{L} = \mathbb{Q}(\gamma)$ of degree d with one field extension, and how to do it.

Theorem 2. *Let $c_1, c_2, \ldots, c_k$ be non-zero integers, $\gamma = \sum_{i=1}^{k} c_i z_i$, and A be the d by d matrix over $\mathbb{Q}$ whose $i'th$ column is $[\gamma^{i-1}]_B$. Then $\det(A) \neq 0$ for all but finitely many c_i and if $\det(A) \neq 0$ then $L \cong \mathbb{Q}(\gamma)$. If $y = A^{-1} \cdot [\gamma^d]_B$, then $M(z) = z^d - \sum_{i=0}^{d-1} y_i z^i$ is the minimal polynomial for γ in $\mathbb{Q}[z]$. Let $\widehat{L} = \mathbb{Q}[z]/M(z)$. Then $L \cong \widehat{L}$ and $C = \{1, z, z^2, \ldots, z^{d-1}\}$ is a basis for $\widehat{L}$. Furthermore, if $\phi : L \to \widehat{L}$ with $\phi(a) = (A^{-1} \cdot [a]_B)^C$ then ϕ is an isomorphism between L and $\widehat{L}$. The cost of computing A and A^{-1} is $O(d^3)$ arithmetic operations in $\mathbb{Q}$. The cost of applying ϕ is d^2 multiplications in $\mathbb{Q}$ because of the matrix vector multiplication.*

We compare the cost of a multiplication in the fields $\widehat{L}$ and L, that is, working in a simple algebraic extension instead of a tower. Consider the case where $d_i = 2$ for $1 \leq i \leq k$, that is, all extensions in L are quadratic thus $d = 2^k$. For $a, b \in L$, from Theorem 1, $a \times b$ does $M(k) = 6^k$ multiplications in the ground field $L_0 = \mathbb{Q}$. For $a, b \in \widehat{L}$, $a \times b$ does $d(2d-1) = 2^k(2 \cdot 2^k - 1) = 2 \cdot 4^k - 2^k$ multiplication in $\mathbb{Q}$. The following table shows that multiplying in L verses $\widehat{L}$ requires a significantly more multiplications in $\mathbb{Q}$. The same is true when we compute in L mod p and $\widehat{L}$ mod p for a prime p.

k	1	2	3	4	6	8	10
$6^k/(2 \cdot 4^k - 2^k)$	1.00	1.28	1.80	2.61	5.74	12.84	28.85

In the modular GCD algorithm for $L[x]$, we want to compute over L mod p for a prime p. For this to work p must not divide any denominator in any minimal polynomial $m_i(z_i)$. Now let $L_p = L$ mod p and $\widehat{L}_p = \widehat{L}$ mod p. The rings L_p and $\widehat{L}_p$ are isomorphic. Furthermore, L_p and $\widehat{L}_p$ are finite rings with p^d elements which are either units, zero divisors or the 0 element.

For most choices of p, the minimal polynomial $M(z)$ for γ in Theorem 2 will factor over $\mathbb{Z}_p$ and then L_p and $\widehat{L}_p$ will have zero divisors. If we compute a GCD in $L_p[x]$, the monic Euclidean algorithm may try to invert a zero divisor. When

this happens, the modular GCD algorithm aborts this prime and simply restarts with a new prime. Provided the primes used satisfy $p \gg d$, the probability of encountering a zero divisor is low.

A second reason why computing in L is more expensive than computing in $\widehat{L}$ when $k > 1$ is that for every multiplication in L_i there are $d_i(2d_i^2 - 1)$ multiplications in L_{i-1}, each of which requires at least one piece of storage to store the product in $L_{i-1}[z_i]$ and a second piece of storage to store the remainder after division by m_i. This storage management overhead is high when d_1 is low, e.g., $d_1 = 2$, a case that occurs very often in practice.

We have implemented a C library for computing in $L_p[x]$ for primes $p < 2^{63}$ for $k \geq 1$. We eliminate the storage management overhead by preallocating working storage of size cd units for a small constant c. All field arithmetic in L_p runs in place, with no calls to the storage manager for space.

Another factor affecting the efficiency of computation in L_p and $\widehat{L}_p$ is the size of the primes. Ideally, we want to use the largest primes we can that are supported by the hardware. Let $a, b \in \mathbb{Z}_p$ with $0 \leq a, b < p$. For a GCD computation in $L_p[x]$, Maple uses 31.5 bit primes so that $a \times b < 2^{63}$ so that signed 64 bit integers can be used to multiply in $\mathbb{Z}_p$ in C using `a*b % p`. For $p < 2^{25}$ Magma uses floating point arithmetic to multiply in $\mathbb{Z}_p$. Our C library supports 63 bit primes. To multiply in $\mathbb{Z}_p$ we use Roman Pearce's implementation of Moller-Granlund [5]. The 63 bit primes means fewer primes are needed in the modular GCD algorithm.

We end with a timing benchmark which compares the cost of computing over L_p with $\widehat{L}_p$ using Pari, Magma, Maple and our C library. We used the number field $\mathbb{Q}(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ with minimal polynomials $m_1 = z_1^2 + z_1 + 1$, $m_2 = z_2^4 - z_1z_2 - 2$, $m_3 = z_3^2 - z_2z_3 - 4$, and $m_4 = z_4^2 - 3z_3z_4 - z_2$ for $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ respectively. So $d_1 = 2, d_2 = 4, d_3 = 2, d_4 = 2$ and $d = 32$.

We constructed GCD problems in $L[x]$ as follows. Let $a = x + 2z_1z_4 + 5z_3 + 4z_1z_2 + 3$, $b = x + z_2z_4 + 6z_3 + 7z_1z_2 + 8$ and $g = x + z_3z_4 + 9z_1z_4 + 2z_2 + 2$. For $n = 3, 7, 15, 31, 63, 127$ we set $f_1 = ga^n$ and $f_2 = gb^n$ and we compute $\gcd(f_1, f_2) = g$ in $L_p[x]$, and in $\widehat{L}_p[x]$, 100 times in a loop, for $p = 2^{25} - 855$ then for $p = 2^{62} - 923$. The two primes were chosen so that the ring $\widehat{L}_p$ is a field so that the Magma code works as Magma insists L mod p be a field.

In Table 1, there are two timings for Pari, Magma, Maple, and ipgcd (our C library). The first timing is for L_p, so for four extensions. The second timing in parenthesis is for $\widehat{L}_p$, so for one extension. The timings for one extension are much faster for all systems. The timings in the last column labeled phigcd include the cost of computing ϕ_γ, converting the coefficients of f_1, f_2 into $L_p = \mathbb{Q}(\gamma)$ mod p, computing the gcd in $L_p[x]$ and applying ϕ_γ^{-1}. Shown in parenthesis is the time for computing ϕ_γ, that is, computing A, A^{-1}, and $M(z)$ in Theorem 2.

The timings in Table 1 were generated using Magma V2.26-12, Pari 2.13.3, and Maple 2022. The Pari code, Magma code, and Maple code used is available at `www.cecm.sfu.ca/~mmonagan/code/CASC2026` . The hardware used was an Intel Xeon E5-2660 CPU running at 2.2 GHz (base) and 3.0 GHz (turbo) with 64 GB RAM.

$d = 32, p = 2^{25} - 855$

n	Pari	Magma	Maple	ipgcd	phigcd(%phi)
3	1.178(0.0301)	0.36(0.087)	0.11(0.045)	0.067(0.0155)	0.081(77.8%)
7	4.371(0.0691)	1.10(0.159)	0.30(0.101)	0.257(0.0484)	0.117(59.3%)
15	16.66(0.1821)	3.36(0.471)	0.97(0.275)	0.972(0.1617)	0.238(26.2%)
31	64.29(0.5507)	11.52(1.711)	3.25(0.799)	3.652(0.579)	0.672(9.3%)
61	239.1(1.754)	38.80(6.160)	11.1(2.616)	13.37(2.046)	2.197(2.9%)
127	STACK(6.734)	158.2(26.36)	45.28(10.23)	56.11(8.403)	8.717(0.7%)

$d = 32, p = 2^{62} - 923$

n	Pari	Magma	Maple	ipgcd	phigcd(%phi)
3	1.307(0.0722)	1.37(0.230)	1.535(BUG)	0.097(0.0234)	0.100(69.3%)
7	4.910(0.1870)	3.90(0.597)	5.184(BUG)	0.369(0.0760)	0.159(44.2%)
15	18.71(0.5320)	12.04(1.755)	18.34(BUG)	1.378(0.2609)	0.356(29.8%)
31	72.56(1.703)	40.26(5.830)	66.38(BUG)	5.264(0.948)	1.082(6.6%)
61	268.3(5.632)	148.4(19.85)	231.5(BUG)	19.12(3.375)	3.635(2.0%)
127	STACK(22.09)	662.0(79.40)	954.3(BUG)	77.84(13.95)	14.63(1.0%)

Table 1. Timings in CPU seconds. STACK means Pari ran out of stack space.

References

1. Ansari, M., Monagan, M. Computing GCDs of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions. *Proceedings of CASC 2023*, LNCS **14139**:1–20, Springer, 2023.
2. Geddes, K.O., Czapor, S., and Labahn, G. *Algorithms for Computer Algebra*, Kluwer Academic, 1992.
3. Encarnacion M.J. Computing GCDs of Polynomials over Algebraic Number Fields. *J. Symb. Comput.*, **20**:299-313, Elsevier, 1995.
4. van Hoeij, M., Monagan, M. A Modular GCD Algorithm over Number Fields Presented with Multiple Extensions. *Proceedings of ISSAC '02*, pp.109-116, ACM, 2002.
5. Moller, N., Granlund, T. Improved Division by Invariate Integers. *IEEE Transactions on Computers* **60**(2):165-175, IEEE, 2011.
6. Wang, P.S., Guy, M.J.T., Davenport, J.H.: P-adic reconstruction of rational numbers. SIGSAM Bulletin, **16**(2):2–3, ACM (1982)

Appendix: Pari code

```
? x;y;z;u;w;    /* Fixes x>y>z>u>w */
? p = 2^25-855; /* p = 2^62-923; */
? m1 = w^2+w+1;
? m2 = u^4-u*w-2;
? m3 = z^2-z*u-4;
? m4 = y^2-3*y*z-u;
/* K = Z/pZ[y,z,u,w]/<m1,m2,m3,m4>  */
? zero = Mod(Mod(Mod(Mod(Mod(0,p),m1),m2),m3),m4);
? g = x+2*u*y+5*z+4*w*u+3 + zero;
? a = x+y*w+6*z+7*w*u+8 + zero;
? b = x+y*z+9*y*u+2*w+2 + zero;
```

```
? n = 7; /* n = 3, 4, 7, 15, 31, 61, 127 */
? aa = g*a^n; bb = g*b^n;
? monicgcd = (a,b) -> {G = gcd(a,b); G/pollead(G)};
? for( i=1, 100, H=monicgcd(aa,bb) );
? ##
  ***   last result: cpu time 4,360 ms, real time 4,373 ms.
? liftall(H)
%276 = x + (2*u*y + (5*z + (4*w*u + 3)))
```

Appendix: Magma code

```
> p := 2^25-855; // p := 2^62-923;
> Fp := FiniteField(p);
> P1<z> := PolynomialRing(Fp); m1 := z^2+z+1; K1<z>,phi1 := quo<P1|m1>;
> P2<y> := PolynomialRing(K1); m2 := y^4-y*z-2; K2<y>,phi2 := quo<P2|m2>;
> P3<x> := PolynomialRing(K2); m3 := x^2-x*y-4; K3<x>,phi3 := quo<P3|m3>;
> P4<w> := PolynomialRing(K3); m4 := w^2-3*x*w-y; K4<w>,phi4 := quo<P4|m4>;
> P<u> := PolynomialRing(K4);
> g := u+2*w*z+5*x+4*y*z+3;
> a := u+w*y+6*x+7*y*z+8;
> b := u+w*x+9*w*z+2*y+2;
> n := 3; // 4, 7, 15, 31, 61, 127
> a := a^n*g;
> b := b^n*g;
> time for i := 1 to 100 do h := Gcd(a,b); end for;
Time: 0.480
> h;
u + 2*z*w + 5*x + 4*z*y + 3
```

Appendix: Maple code

```
> kernelopts(opaquemodules=false):
> RD := Algebraic:-RecursiveDensePolynomials:
> p := 2^25-855:  # p := 2^62-923;
> m1 := z^2+z+1:
> m2 := y^4-y*z-2:
> m3 := x^2-x*y-4:
> m4 := w^2-3*w*x-y:
> R := ( [u,w,x,y,z], [m4,m3,m2,m1], p ):
> g := RD:-rpoly( u+2*w*z+5*x+4*y*z+3, R ):
> aa := RD:-rpoly( u+w*y+6*x+7*y*z+8, R ):
> bb := RD:-rpoly( u+w*x+9*w*z+2*y+2, R ):
> n := 3:  # 3, 4, 7, 15, 31, 61, 127
> a := RD:-mulrpoly(g,RD:-powrpoly(aa,n)):
> b := RD:-mulrpoly(g,RD:-powrpoly(bb,n)):
> CodeTools[Usage]( to 100 do h := RD:-gcdrpoly(a,b) od ):
memory used=2.12MiB, alloc change=0 bytes, cpu time=40.00ms, real time=41.00ms, gc time=0ns
> RD:-rpoly(h); # check
                          2 w z + 4 y z + u + 5 x + 3
```