COMPUTING CLASS GROUPS AND UNIT GROUPS IN MAGMA

ANDREAS-STEPHAN ELSENHANS AND JOHN VOIGHT

Abstract. We describe the computation of class groups and unit groups of number fields as implemented in Magma (V2.29). After quickly reviewing the main algorithms based on factor bases, relation collection, and analytic class number evaluation, we distinguish their behavior across formalizable, rigorous, GRH-conditional, and heuristic regimes.

1. Introduction

A bit of motivation. In computational algebraic number theory, determining the class group and unit group of the ring of integers of a number field is a fundamental algorithmic task. Their computation remains of significant theoretical interest—many central questions in number theory concern properties of these groups, including their distribution—and they are indispensible in practice, underpinning explicit methods across arithmetic geometry. We might even consider these algorithms as one key benchmark for a computational algebra system used in number theory.

Takeaways. We consider five possible regimes of computing.

- *Formalizable*: rigorously proven in a manner that could be verified by a formalized proof assistant—in particular, unconditional. If numerical calculations are performed, they must use ball arithmetic.
- *Rigorous*: proven, but we allow numerical calculations if they include floating-point error analysis (but can ignore any possible accumulated round off errors).
- *GRH-conditional formalizable* and *GRH-conditional rigorous*: same but we assume the Generalized Riemann Hypothesis (GRH).
- *Heuristic*: good heuristics may be assumed, so it is very likely to be the correct answer, but it should not be considered proven. In particular, we allow good, reliable, nonrigorous numerical computations.

One step beyond formalizable would be *formalized*, where steps along the way are output with the intention of entering them into a formalized proof assistant.

In this article, we document class group and unit group algorithms in Magma [BCP97]. With the above in mind, the takeaways are as follows.

(1) By default (for example, calling `ClassGroup` with no optional arguments), the class group algorithm in Magma is formalizable. A "standard" approach is taken: generators are taken up to the Minkowski bound, relations are found, and then the class group is confirmed to be saturated. The latter steps are done using exact computations.

The bad news: a bug was uncovered (in fact, it has been a problem for a some time!) which turned off saturation of the class group. So the results returned were not formalizable. This has been fixed as of V2.29; and, for what it is worth, by

Key words and phrases. Class groups, unit groups.

the way that the relations are computed it is quite unlikely that a wrong answer was returned.

(2) By default, the unit group algorithm in Magma is rigorous. A lower bound of the regulator is computed and the class group is p-saturated with respect to all primes p smaller than the quotient of the regulator and the regulator bound. As all regulator computations use floating point numbers, this is not formalizable; but it could be made so, with ball arithmetic.

(3) When the GRH flag is added, in V2.28 (indeed going back many versions), a non-rigorous evaluation of the Euler product was used and so the class group and unit group were not rigorous or formalizable. In V2.29, this bug is fixed.

Calling `ClassGroup` with proof level `GRH` will result in a GRH-conditional rigorous computation. Further, the proof level `Subgroup` in combination with a GRH-bound for the generators of the class group will result in a GRH-conditional formalizable algorithm, as the check for saturation is performed.

Calling `UnitGroup` with the additional parameter `GRH` will skip the saturation and therefore result in a GRH-conditional rigorous computation. Without that parameter, the saturation check is done as well and the result will be rigorous.

(4) As it turns out, the bit of good news is that a non-rigorous Euler product evaluation turns out to be extremely close and very useful as a heuristic—so much so that again it seems extremely unlikely that a wrong answer was ever returned. We discuss at the end some possible heuristic algorithms that follow up on this observation.

Contents. This article gives a detailed account of how class groups and unit groups of number fields are computed in Magma V2.29, peppered with observations and questions throughout. We start in Section 2 with a brief history. Section 3 recalls the basic definitions and fundamental algorithmic problems, highlighting issues such as representation of number fields, maximal orders, and the Picard group. In section 4, we review the core factor base method, describing how generators and relations are obtained under various assumptions, how to decide when enough relations have been found, and how this connects to heuristics and conjectures. Then we turn to analytic techniques in section 5, detailing both rigorous GRH-conditional estimates and heuristic Euler product evaluations of the Dedekind zeta residue, with discussion of error analysis and stopping criteria. Section 6 focuses on unit group algorithms, including classical methods for quadratic fields, strategies for detecting and saturating missing units, and the use of discrete logarithms modulo primes. Then in section 7 we explain how the class group is saturated and verified, describing the detection of missing generators or relations and the structure of the saturation algorithm. We conclude in section 8 with a summary with benchmarks, along with observations on typical error patterns and some possible future directions.

Acknowledgements. We would like to thank Eran Assaf, Edgar Costa, Tim Dokchitser, Claus Fieker, Nicole Sutherland, Allan Steel, and Alain Chavarri Villarello for helpful conversations. Thanks also to the participants in the *Magma Meeting: Rational Points 2025* for their input including some suggestions for future work in the final section.

2. A bit of history

Many individuals have contributed to the current package in Magma. We would appreciate hearing of any corrections or additions to our attempt to reconstruct the history here. Since they often go together, we lump together code to compute the class group and unit group together, referring only to the former.

The class group code in Magma originates from code taken from the KANT package in 1994. This code was written in the early 1990s by Johannes Graf von Schmettow, then rewritten chiefly by Klaus Wildanger and Florian Hess in 1996 and then updated in Magma.

In the mid 1990s, code from Pari was installed and modified by Alexandra Flynn for class groups of quadratic fields. Although it was heavily used and supported during that time, it is no longer called (except through code for binary quadratic forms).

Claus Fieker built on the class group package by providing explicit algorithmic class field theory during the period 2000–2011 [F01].

Jürgen Klüners and Sebastian Pauli developed algorithms for computing the Picard group of non-maximal orders and for embedding the unit group of non-maximal orders into the unit group of the field in the mid 2000s. Also during this period, Mark Watkins implemented the fast algorithm of Wieb Bosma and Peter Stevenhagen for computing the 2-part of the ideal class group of a quadratic field.

In 2011, Jean-François Biasse implemented a quadratic sieve for computing the class group of a quadratic field in collaboration with Steve Donnelly and Nicole Sutherland. He also developed a generalisation of the sieve for number fields of degree greater than 2; it is reported to perform well for degrees up to 5 and to some extent in degree 6.

In the period 2012–2014, Steve Donnelly provided a new implementation of the general class group algorithm using a random walk on ideals with heavy use of LLL on ideal bases to find smooth relations—this is generally much faster than the old KANT code. Allan Steel also made significant improvements in this code.

Finally, Allan Steel implemented a distributed parallel version of the class group code in 2022.

3. Basic algorithms

We refer to the Magma handbook and the article by Claus Fieker [F06] for an overview. For general references, there is also an article by Lenstra [L92] and the books of Cohen [Co93, Co93]. Many of the results were originally obtained by Zassenhaus, Post, Cohen, Buchmann and many others, but this is not the right place to attempt to give detailed references.

Setup. In Magma, a number field $K = \mathbb{Q}(\alpha)$ is represented by the minimal polynomial $f(x) \in \mathbb{Q}[x]$ of a primitive element α. (In fact, it is represented in KANT by an equation order, which is why there are sometimes issues when the polynomial does not have integer coefficients.)

Question 3.1. We consider here only the absolute case. However, much of what is done can be made relative. This could be important if considering many extensions K of a given field F (to avoid recomputing data coming from the base field).

For example, what speed improvements could we hope for in the case of CM extensions K of a totally real field F (analogous to the case of imaginary quadratic fields)? As the rank of the unit group of a CM extension coincides with the unit rank over the totally real field, a specialized unit group algorithm may be helpful.

Remark 3.2. We could also work more broadly with étale algebras; this is explained and implemented in Magma by Stefano Marseglia. However, these algebras project onto their number field factors and a basic primitive used in these algorithms is the number field case.

Question 3.3. Do we gain any efficiency in working with number fields embedded in higher codimension? For example, $K = \mathbb{Q}[x,y]/(f(x,y), g(x,y))$? If a lot of time is spent mapping from a power basis, then perhaps. If everything is computed relative to a small (LLL-reduced?) integral basis, then this should be independent of the way that the algebra is represented. Perhaps we should lead with this way of presenting arithmetic in number fields—sure, the input is a minimal polynomial but most of the action is in finding a good basis for orders?

We suppose that a maximal order has been computed.

Question 3.4. The assumption that the order is maximal is unnecessary and could be quite expensive to prove. The algorithms work perfectly well without this assumption, working instead with the Picard group and when something goes wrong, we typically have produced a superorder. Is this difficult to implement?

Fundamental algorithms. We write $\operatorname{Cl}\mathcal{O}_K$ for the class group of $\mathcal{O}_K$, and write $\mathrm{h}(\mathcal{O}_K)$ for its order. Let r_1, r_2 be the number of real, complex places of K and let $d := \operatorname{disc}(\mathcal{O}_K)$ be the discriminant.

The class group is a finite abelian group by the geometry of numbers, and the unit group is a finitely generated abelian group by Dirichlet's unit theorem, of rank $r_1 + r_2 - 1$. Applying logarithms to the absolute values under the set of r archimedean places of K maps the unit group to a lattice of rank $r-1$. The kernel of this map are the roots of unity in $\mathcal{O}_K^\times$. The covolume of this lattice is the regulator.

Given as input the maximal order $\mathcal{O}_K$, there are two fundamental (and related) problems.

(1) Class group: give as output a computable isomorphism between an abstract finite abelian group and the class group $\operatorname{Cl}\mathcal{O}_K$ (as a group of classes of ideals)—so one can evaluate both the map (give an ideal which represents the given class) and its inverse (given an ideal, find its class).
(2) Unit group: give as output a computable isomorphism between an abstract finitely generated abelian group and the unit group $\mathcal{O}_K^\times$.

We allow the output of the unit group to be in a compact representation, as a product of elements with exponents. Calling `UnitGroup` with the optional parameter `Raw` gives the user access to this representation.

We can easily get roots of unity in the field, so we ignore this throughout. One way to obatain them is a search for elements of small T_2-norm, which is part of the saturation. But, usually they are obtained during the unit group computation.

The example

```
> _<x> := PolynomialRing(Rationals());
> K := NumberField(x^2 + 890232348011);
> ord := MaximalOrder(K);
> time c1, c2 := ClassGroup(ord : Proof := "GRH");
Time: 0.070
> 1.0 * Norm(c2(30000*c1.1));
6.022334006496979685501715642 31E172002
```

shows that the class group map can result in huge representatives, if the class group is a large cyclic group.

4. The factor base method: an overview

Basic idea. What better way to compute a finitely generated abelian group than with generators and relations! For the class group, we take a set of classes of prime ideals; we call this the factor base. We then find relations, meaning elements $\alpha \in K^\times$ which factor completely over the factor base. We can combine relations until they have the trivial factorization, which gives a unit.

Written out, let $S := \{\mathfrak{p}_1, \dots, \mathfrak{p}_m\}$ be a set of primes. Let

$$\mathcal{O}_{K,S}^\times := \{\alpha \in K^\times : \operatorname{ord}_\mathfrak{p}(\alpha) = 0 \text{ if } \mathfrak{p} \notin S\}$$

be the subgroup of elements which factor over the primes in S, i.e., the S-unit group. Then there is an exact sequence

$$1 \to \mathcal{O}_K^\times \to \mathcal{O}_{K,S}^\times \xrightarrow{\phi} \mathbb{Z}^m \xrightarrow{\pi} \operatorname{Cl} \mathcal{O}_K \tag{4.1}$$

where the first map is the natural inclusion, the map ϕ is factorization

$$\begin{aligned} \phi\colon \mathcal{O}_{K,S}^\times &\to \mathbb{Z}^m \\ \alpha &\mapsto (\operatorname{ord}_{\mathfrak{p}_i}(\alpha))_{i=1,\dots,m}, \end{aligned} \tag{4.2}$$

and π is taking the class

$$\begin{aligned} \pi\colon \mathbb{Z}^m &\to \operatorname{Cl} \mathcal{O}_K \\ (e_1, \dots, e_m) &\mapsto \prod_i^m [\mathfrak{p}_i^{e_i}]. \end{aligned} \tag{4.3}$$

The set S generates the class group if and only if π is surjective; in that case, if we can compute $\ker \pi$, we know the class group. So we search for enough elements in $\mathcal{O}_{K,S}^\times$ to map to a generating system of $\ker \pi \leq \mathbb{Z}^m$. As a byproduct, we can find candidate generators for the unit group from $\ker \phi$.

So over the next few subsections we will discuss generators, finding relations, and then how we know when we have found all relations. We break these up into the various regimes explained above.

Formalizable generators. The theorem of Minkowski provides that every ideal class of $\mathcal{O}_K$ has an integral representative $\mathfrak{a}$ with

$$\operatorname{Nm}(\mathfrak{a}) \leq \frac{n!}{n^n} \left(\frac{4}{\pi}\right)^{r_2} \sqrt{|d|} \tag{4.4}$$

where Nm denotes the absolute norm. (This is one way to prove that the class group is finite, but it is not necessarily an efficient way to represent elements of the class group.)

So for something formalizable, to be sure we have generators we take all prime ideals up to the Minkowski bound.

Remark 4.5. Sometimes the Minkowski bound is shortened to $O(\sqrt{|d|})$. But, this is only literally true when the degree is fixed, as it does not take the leading coefficient into account. Indeed, Stirling's approximation (which is quite good!) gives

$$\frac{n!}{n^n} \sim \frac{\sqrt{2\pi n}}{e^n} \approx 2.50\sqrt{n}e^{-n}$$

which decays exponentially in n. Already for $n = 10$, the constant is only 0.0012.

Although it is not meaningful to study the inequality for fixed discriminant, we might study number fields of small discriminant relative to the GRH lower bound, which is of the form

$$\sqrt{|d|} \gtrsim (60.84)^{r_1}(22.38)^{2r_2}$$

which suggests that a lower estimate for the Minkowski bound of the shape

$$2.50\sqrt{n}(22.38)^{r_1}(9.29)^{2r_2}$$

is possible. This expresses the Minkowski bound for number fields of large degree n whose discriminant is small relative to the degree: it at least indicates that the bound is larger when there are more complex embeddings, which makes sense.

The Zimmert bounds improved the Minkowski constants; there is an elegant reformulation due to Oseterlé. The best bound is that for every class C, there is an integral ideal $\mathfrak{a}$ either in C or $\mathfrak{d}C$ where $\mathfrak{d}$ denotes the different of $\mathcal{O}_K$ such that

$$\mathrm{Nm}(\mathfrak{a}) \leq (0.128 - o(1))^{r_1}(0.044 - o(1))^{r_2}\sqrt{|d|} \tag{4.6}$$

and

$$\mathrm{Nm}(\mathfrak{a}) \leq (0.14 - o(1))^{r_1}(0.051 - o(1))^{r_2}\sqrt{|d|} \tag{4.7}$$

for all ideal classes. Note how much better this is than Minkowski, which gives roughly

$$\mathrm{Nm}(\mathfrak{a}) \lesssim 2.50\sqrt{n}(0.36)^{r_1}(0.46)^{r_2}\sqrt{|d|}. \tag{4.8}$$

It is of course no trouble to include the class of the different (for example, it is trivial for monogenic orders).

Remark 4.9. The constants in Zimmert's inequality are known. Therefore, we have `ZimmertBound` in Magma. But, the improvement is not as big as one might hope for, as the following example shows:

```
> _<x> := PolynomialRing(Rationals());
> f := &*[x-i : i in [-6..6]] + 1;
> OK := MaximalOrder(f);
> 1.0* Discriminant(OK);
1.62088452500733828715108860464E88
> MinkowskiBound(OK);
261753666880391282721277871027153305290 2
> ZimmertBound(OK);
94965373777952522128015579012381435 03
> 1.0 * $2 / $1;
275.63063932378359883273396843 3
```

A constant factor, but in either case it is so large it is practically useless.

Remark 4.10. There is no known unconditional bound for the norms of a set of *generators* for the class group. The Minkowski bound allows us to represent every element of the class group this way, which is overkill.

Conditional generators. Conditional on the GRH, we can improve on Minkowski as follows.

Theorem 4.11 (Bach)**.** *Assuming GRH, the set*

$$\{\mathfrak{p} \subset \mathcal{O}_K : \mathrm{Nm}(\mathfrak{p}) \leq 12(\log|d|)^2\}$$

generates $\mathrm{Cl}\,\mathcal{O}_K$.

Proof. This is a direct consequence of [B96, Theorem 4]. □

Belabas–Diaz y Diaz–Friedman [BDF08] gave another GRH bound. In practice, one can use the minimum of the two. This is implemented in Magma via `GRHBound` and is used by the class group algorithm.

Heuristic generators. The Cohen–Lenstra–Martinet heuristics predict that the class group is "close to cyclic". More precisely, if G is the Galois group of K then the prime-to-$(\#\mu_K \# G)$ part of the class group has a matrix model in generators and relations like the above (with the size of matrices tending to infinity). For imaginary quadratic fields, the heuristic says that the odd part is cyclic with probability 97.7%. The odds only go up when considering higher degree fields, because they are modelled by a further quotient by units.

Using the Chebotarev theorem as a heuristic, we expect prime ideals to land randomly as elements in this cyclic group, so again the probability is very high that one small ideal, surely a few, will generate. From this perspective, we do not need to take all primes up to a given bound. But then we still have to find relations, a topic we turn to next.

Question 4.12. We also still have to worry about ℓ-Sylow subgroups with $\ell \mid \#\mu_K \# G$. For the 2-part of quadratic fields, we have genus theory available, and recent heuristics are theorems in this case. What does a generalization to arbitrary degree say heuristically?

Finding relations. By their construction as given above, the relations are elements of $\phi(\mathcal{O}_{K,S}^\times) \leq \mathbb{Z}^m$.

Many people have worked on different approaches to generate relations. Here, it is sufficient to know that we have some code that does it; and the longer we run it, the more relations we get. We may want at least to be sure that we loop over small elements to find small units, as in Equation (6.2).

In general, one can (and has to) view such code as a black box. First, some of the algorithms are randomized [Co93, section 6.5.2]. Second, if sieving is used to find relations, one might implement the optimizations listed in [Co93, section 10.4.3]. These result in a general speed up, but the search for relations is no longer exhaustive in a well-defined search range.

Question 4.13. What if we started by making relations, keeping those that satisfy some criterion, then form the generators based on the desired relations? This is the problem of finding a subset of r rows which are supported on r columns. Is this NP-complete?

Using linear algebra, given sufficiently many relations one can:

- compute a sublattice of $\ker(\phi)$.
- compute a subgroup of the unit group.

A parallel version of the relation search was implemented by Allan Steel in 2020–2022. It is based on the farmer-worker-model and is turned on once Magma is set to parallel mode via `SetNthreads`. As this allows to work with larger examples, the relation matrix is now represented as a sparse matrix.

When to stop searching? Given a method to find relations, we need a mechanism which indicates when to stop searching and proceed with simplifying the presentation.

For the unit group: if we miss units, the unit group does not have the expected rank or the discriminant of the unit lattice (regulator) is *larger* than it is supposed to (fewer units means larger covolume) by a positive integer factor.

For the class group: if we miss relations, given that we always take a set of generators we have only a submodule of $\ker(\pi)$. Thus, the class number found is a *multiple* of what it is supposed to be.

Remark 4.14. In the heuristic case, the factor base may also be too small and we may not have generators; but that is the point of the heuristic method, by design this is supposed to be very unlikely to happen.

In either case, because we are off by a positive integer factor, it would be enough to know the product $\mathrm{h}(K) \cdot \mathrm{Reg}(K)$ to a specified precision, so we turn to this now.

5. Analytic approaches

The analytic class number formula reads:

$$\operatorname*{res}_{s=1} \zeta_K(s) = \frac{2^{r_1}(2\pi)^{r_2} hR}{w\sqrt{|d|}}$$

where $w = \#O_{K,\mathrm{tors}}$ is the number of roots of unity in K, $h = \mathrm{h}(\mathcal{O}_K)$, and $R = \mathrm{Reg}(\mathcal{O}_K)$.

Formalizable evaluation. We could try to compute the residue by using the general package for working with L-series, implemented by Tim Dokchitser [D04]. The given complexity is $O(\sqrt{|d|})$, which makes it impractical unless the discriminant is small (in which case other methods are still faster). Also, the error terms are not worked out all the way to the end. In principle, this could be made formalizable using ball arithmetic.

An explanation for this: the Euler product actually converges for the Dedekind zeta function (see below), but for elliptic curves at $s = 1$ it does not—you need analytic continuation to push past $\mathrm{Re}(s) > 3/2$.

Currently, there is no formalizable method available in Magma that uses an analytic approach (in particular, we do not have ball arithmetic). However, we still use the heuristic evaluation below to give a *practical* guess as to when to stop looking for relations, as needed at the end of the previous section. We still have to do something else to certify (see below), and for theoretical purposes we could skip this—but in practice, it gives us a good indication of when we should stop searching.

Conditional evaluation. We now explain a conditional, rigorous method.

For $X > 0$, define

$$\begin{aligned} A_K(X) &:= \sum_{\mathfrak{p} \subset \mathcal{O}_K, N(\mathfrak{p}^m) < X} \frac{\log N(\mathfrak{p})}{N(\mathfrak{p})^{m/2}} \left(\frac{\sqrt{X}\log(X)}{N(\mathfrak{p})^{m/2}\log(N(\mathfrak{p})^m)} - 1 \right) \\ f_K(X) &:= \frac{3(A_K(X) - A_{\mathbb{Q}}(X) + A_{\mathbb{Q}}(X/9) - A_K(X/9))}{2\sqrt{X}\log(3X)} \end{aligned}$$

Theorem 5.1 (Belabas–Friedman, Bach)**.** *Assuming GRH, for all $X \geq 69$ we have*

$$|\log \zeta_K^*(1) - f_K(X)| \leq \frac{2.324 \log|d|}{\sqrt{X}\log(3X)} C$$

where

$$C := \left(1 + \frac{3.88}{\log(X/9)}\right)\left(1 + \frac{2}{\sqrt{\log|d|}}\right)^2 + \left(\frac{4.26(n-1)}{\sqrt{X}\log|d|}\right).$$

Proof. See [BF15, Theorem 1]. □

To estimate the numerical error in the evaluation of $\log \operatorname{res}_{s=1} \zeta_K(s)$ we use ϵ for the relative error of a single floating point operation. Further, we will use the inequalities listed in [SMC06, § VII.1] and [SMC06, § VII.28].

As a first step, we estimate the sum of absolute values. For $A_K(X)$ this is bounded by

$$\begin{aligned}
&\sum_{\mathfrak{p}\subset\mathcal{O}_K, N(\mathfrak{p}^m)<X} \frac{\log N(\mathfrak{p})}{N(\mathfrak{p})^{m/2}} \left(\frac{\sqrt{X}\log(X)}{N(\mathfrak{p})^{m/2}\log(N(\mathfrak{p})^m)} \right) \\
&\leq [K:\mathbb{Q}]\sqrt{X}\log(X) \left(\sum_{p<X} \frac{1}{p} + \sum_{p,m\geq 2} \frac{1}{p^m} \right) \\
&\leq [K:\mathbb{Q}]\sqrt{X}\log(X) \left(\log\log X + 0.2615 + \frac{1}{2(\log X)^2} + 0.77316 \right) .
\end{aligned}$$

To simplify the notation, we set

$$E(X) := \sqrt{X}\log(X) \left(\log\log X + 0.2615 + \frac{1}{2(\log X)^2} + 0.77316 \right) .$$

Assuming that the computation of one term in A_K uses k floating point operations, we can bound the sum of all errors of all the terms of $A_K(X)$ by

$$\epsilon k [K:\mathbb{Q}] E(X) .$$

As the total sum has less than $[K:\mathbb{Q}]X$ terms, the error of summing all the terms in a pairwise summation scheme is bounded by

$$\epsilon \frac{\log_2([K:\mathbb{Q}]X)}{1-\epsilon\log_2([K:\mathbb{Q}]X)} [K:\mathbb{Q}] E(X) .$$

This results in an error bound of

$$\epsilon 2(2+\log\log X)([K:\mathbb{Q}]+1)\frac{3}{2}\left(k+\log_2([K:\mathbb{Q}]X)+\log_2(X)\right)$$

for $\log \operatorname{res}_{s=1} \zeta_K(s)$.

If we use the standard data type `single`, one gets $\epsilon = 2^{-24}$. This results in an error bound of less than 0.01 for $X \leq 10^{10}$ and $[K:\mathbb{Q}] \leq 100$. We use the data type `double`, in which case $\epsilon = 2^{-52}$; therefore we end up with an error bound of less than 10^{-10} for $X \leq 10^{10}$ and $[K:\mathbb{Q}] \leq 100$.

To obtain a GRH-conditional rigorous algorithm, we compute the regulator to enough precision: estimates are worked out by Biasse–Fieker [BF14, Lemma 4.4].

The current Magma implementation for the GRH-conditional unit group and class group with proof level `GRH` rely on the correctness of the residue. However, setting only the bound for the class group generators to `GRH` will call the check of saturation and gives therefore a GRH-conditional formalizable result.

Heuristic evaluation. We can also just evaluate the Euler product "directly".

Theorem 5.2. *The infinite product*

$$\zeta_K^*(1) := \operatorname*{res}_{s=1} \zeta_K(s) = \lim_{s\to 1} \frac{\zeta_K(s)}{\zeta(s)} = \prod_p \frac{\zeta_{K,p}(1)}{\zeta_p(1)} = \prod_p \frac{1-1/p}{\prod_{\mathfrak{p}|p}(1-1/\operatorname{Nm}(\mathfrak{p}))}$$

over primes p converges.

Proof. The explicit bounds given in [GL22, Theorem 1] imply the convergence for any number field K. □

Example 5.3. We get

$$\left| \operatorname*{res}_{s=1} \zeta_K(s) - \prod_{p<X} \frac{1-1/p}{\prod_{\mathfrak{p}|p} 1-1/N\mathfrak{p}} \right| < 0.1461$$

for $X = \exp(10^{34})$ and $K = \mathbb{Q}(i)$.

Remark 5.4. For the approximation

$$A(X) := \prod_{p<X} \frac{1-1/p}{\prod_{\mathfrak{p}|p, N\mathfrak{p}<X} 1-1/N\mathfrak{p}},$$

the GRH based estimate

$$|\log(\operatorname*{res}_{s=1} \zeta_K(s)) - A(X)| < \frac{2\log|d| + (0.928n + 0.754)\log(X)}{\sqrt{X}} \text{ for } X > 1000$$

was worked out in [B94, Thm. 2, Tab. 2].

Magma computes the finite product

$$\operatorname*{res}_{s=1} \zeta_K(s) \approx \prod_{p<1000} \frac{\zeta_{K,p}(1)}{\zeta_p(1)}.$$

If this does not match the class and unit group found with an error of 5%, the bound is doubled, and this is repeated recursively. A test example with extra verbose printing is

```
> _<x> := PolynomialRing(Rationals());
> SetKantVerbose("CLASS_GROUP_CHECK",1);
> f := t^16 - 36*t^14 + 488*t^12 - 3186*t^10 + 10920*t^8
 - 19804*t^6 + 17801*t^4 - 6264*t^2 + 64;
> OK := MaximalOrder(f);
> ClassGroup(ord : Proof := "GRH");
Euler-Product bound: 1000
Euler-Product bound: 2000
Abelian Group of order 1
```

(Alas, `SetKantVerbose` only works on the development version.) Thus, the increase in factors works. Note that to *ensure* 5% error using the above GRH-based estimate would require a bound of 10^6 even in small examples.

The above example is the most extreme one within more than 1000 polynomials. Here, the relative error of the residue with the prime bound 1000 found is below 7.5%. If we consider the partial evaluation

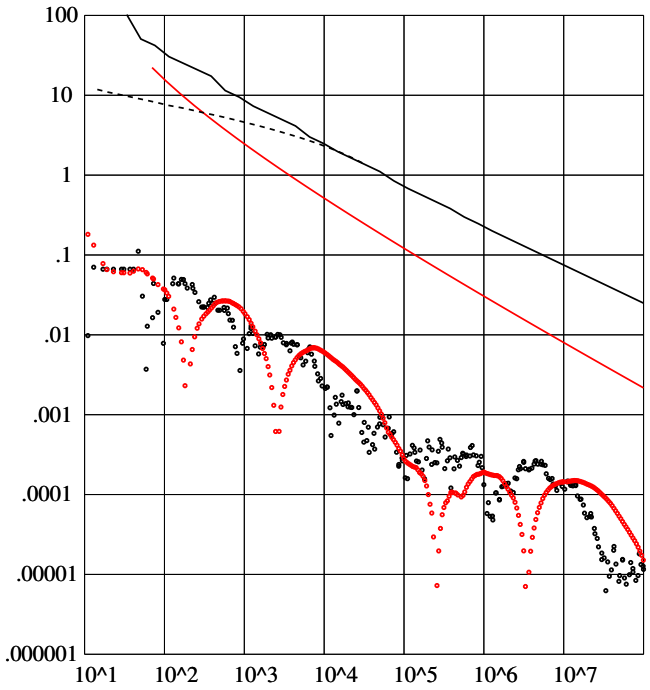
$$\frac{\prod_{p<1000} \zeta_{K,p}(1)/\zeta_p(1)}{\operatorname{res}_{s=1} \zeta_K(s)}$$

assuming GRH, we find in this experiment:

- mean value 1.012,
- standard deviation 0.015,
- skewness -0.0166, and
- kurtosis 3.114.

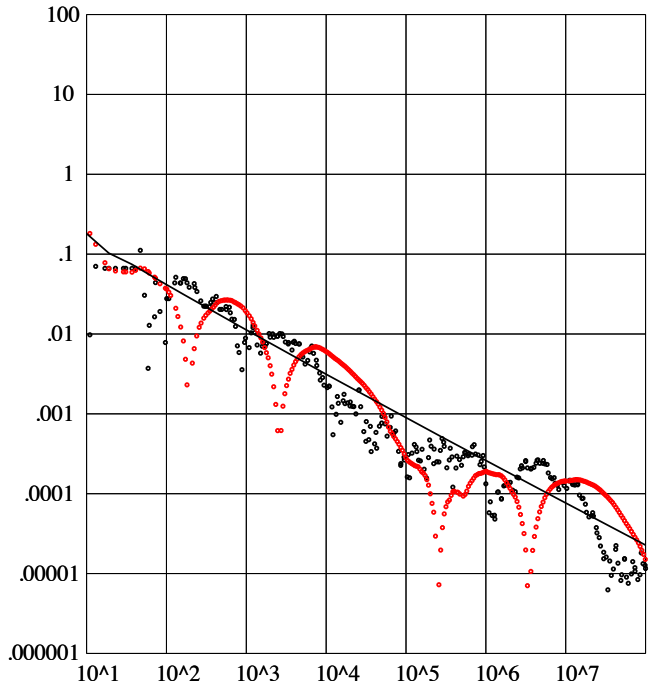
Because of the 5% error heuristic, the error has to be about 30 standard deviations to cause trouble.

Example 5.5. We picked the field given by the randomly chosen polynomial $x^8+9127x^3+9127x^2+9127x+18254$ for a closer inspection. It results in a maximal order with a 45-digit discriminant. We applied both methods described above to approximate the residual. The plot below shows the resulting approximation errors of the logarithm of the residue and the error estimates as a function of the used prime bound.



The color black is used for the Euler product approach and Bach's estimate. The red color is used for the weighted sum estimate and the Belabas-Friedman error bound. Each dot shows the maximal error that occures in the interval that it covers whereas the solid lines show the bounds. The dashed line shows a naive improvement of Bach's bound for small p. We can read off the plot that in practice both methods give about the same error, whereas the error estimates show a much bigger difference. Note that the downward spikes in the red dots indicate a change of sign in the error, whereas the error of the Euler product oscillates at a much higher frequence. Therefore, the changes of sign are not visible in his type of plot.

The next plot compares the two errors with the standard deviation of random Euler products.



The plot indicates that, a stochastic model gives an accurate representation of the errors. The article [GS03] may be interesting for a more theoretic perspective on this.

Question 5.6. Does this hold up for imaginary quadratic fields of large discriminant? How should 1000 actually scale with the parameters (degree, discriminant)?

Remark 5.7. The GRH is a perfectly good heuristic, and some helpful GRH estimates are given by Duke [Duk03, Proposition 5]. For example, we could add the assumption that the ratio $\zeta_K(s)/\zeta(s)$ is an entire Artin L-function—is this an additional assumption we should add to the hierarchy from the introduction?

In any case, this finishes the description of a heuristic algorithm for computing the class group and unit group.

6. Unit group algorithms

We now turn to the formalizable, rigorous, conditional, and heuristic algorithms focused on the unit group.

Quadratic fields. Imaginary quadratic fields have a finite unit group. The unit group of a real quadratic field can be determined by the continued fraction algorithm in quasi-linear time in the size of the output (which can be exponential in the input size). Another algorithm to compute a unit is to use Stark's method and evaluate the L-series.

Question 6.1. A quasi linear version of the continued fraction algorithm is implemented in magma `/package/Ring/FldQuad/fund_unit.m`. When not using a power product representation of units, this approach compares reasonably favorably in speed with the factor base approach.

Missing units. At this stage in the algorithm, we may suppose we have the roots of unity and (by continuing to generate relations) that we have a set of multiplicatively-independent units of rank equal to $r_1 + r_2 - 1$.

Then any missing unit would have an nth power in our given subgroup for some $n \geq 2$, so we need to *saturate* the lattice of units. Our first task is to give an upper bound for n.

Any unit outside the subgroup generated would have to have bounded logarithmic height (thinking of the lattice in the log Minkowski embedding), hence bounded exponential height (by the arithmetic-geometric mean inequality):

$$\log \mathrm{Nm}(x) \leq \log \prod_i (|x|_i)^2 = (\sum_i \log|x|_i)^2 \leq \sum_i (\log|x|_i)^2 \leq \lambda_1(\log \mathcal{O}_K^\times).$$

An approach *not* taken by Magma is to enumerate all elements up to that height.

Remark 6.2. In principle, this could be turned into a rigorous algorithm, at least once we have found (the roots of unity and) a set of linearly independent units.

Instead, we bound n as follows. We enumerate all elements that have at most twice the length of the last entry in the LLL bases of the given order. This gives lower bounds for the logarithmic norm of generators of the unit group. By Minkowski's inequalities this gives a lower bound on the regulator. As this function does an enumeration of small elements, it also determines all the roots of unity contained in the maximal order.

Using this, the index is bounded from above by the quotient

$$\frac{\text{regulator of known subgroup}}{\text{lower bound of regulator}}.$$

The check for saturation is performed for all prime indices up to this bound.

Saturating the unit group. We can now ensure that $\log \mathcal{O}_K^\times$ is ℓ-saturated for all primes $\ell \leq n$, with n computed as in the previous subsection.

In other words, given a prime ℓ we have to show that no ℓth root of a known unit gives a new unit.

A possible approach, not taken by Magma, is to just check if there is an ℓth root running over all possible elements in the lattice modulo ℓth powers. This will be expensive!

Instead, we use the following reduction modulo $\mathfrak{p}$ approach.

- Take sufficiently many (say m) prime ideals $\mathfrak{p}_i$ with $\ell \mid k_i^\times$. Here, we set $k_i := (\mathcal{O}/\mathfrak{p}_i)$.
- An ℓ-th power of $\mathcal{O}^\times$ reduces to an ℓ-th power in $k_i^\times$.
- A ℓ-th power is contained in an index ℓ^m subgroup of $\prod k_i^\times$.

This gives an algorithm, sketched as follows.

(1) Use discrete logarithms to map $k_i^\times/(k_i^\times)^\ell$ to an additive group.
(2) Use linear algebra in $(\mathbb{Z}/\ell\mathbb{Z})^m$ to determine all the known units that reduce to the index ℓ^m subgroup.
(3) If a unit that is not an ℓ-th power reduces to the index ℓ^m subgroup, redo the above computation with a larger value of m or terminate with an error message.
(4) Return that the unit group is ℓ-saturated.

All the above is implemented in KANT and called by Magma.

Remark 6.3. Currently, the discrete logarithm in $k_i^\times/(k_i^\times)^\ell$ is computed via the discrete logarithm of $k_i^\times$. To ensure that this does not get too hard, all the prime ideals $\mathfrak{p}_i$ used are chosen to be degree one primes.

In the range this is used, Pohlig–Hellman reduction in combination with a random walk algorithm for the discrete logarithm is a good choice. As this works by reducing to quotients of prime order, the code could be optimised, by computing only the discrete logarithm in the order ℓ quotient.

Generally, Magma caches the prime ideals, but for the discrete logarithm no caching is visible in the code.

Question 6.4. The Chebotarev density theorem is the heuristic for why this works. What does an optimistic version of this predict for the number of primes required?

7. Finishing the class group

In a similar way, we can saturate the class group. More precisely, we need to detect the following two problems.

- **Missing generators**. If we miss a generator, then a prime ideal below the Minkowski bound gives a new ideal class.
- **Missing relations**. A missing relation results in a class group larger than correct. Further, the decomposition of ideals into classes found is finer than the correct decomposition into ideal classes. As the result has a group structure, it will have elements of prime order that consist of principal ideals.

We do not have to take generators to start with, and this is the approach taken by Magma in the formalizable approach: we start with some plausible set of generators, then after we have relations we check that all of the remaining primes up to the Minkowski bound are covered by the known classes. This improves the speed in the linear algebra, because we can get organized in the class group doing linear algebra with smaller matrices.

In the conditional approach, we are guaranteed a (reasonably small) set of generators. (In a possible heuristic approach, we do not care.) Magma previously incorrectly used a heuristic Euler product evaluation, but now we use a rigorous one and we can stop (in proof level `GRH`) once we pass the analytic check explained in the previous section. If only the bounds for the generators of the class group are set to `GRH`, a check for saturation of the class group is done.

Saturating the class group. What remains then is a formalizable algorithm for the class group. We again do this with saturation. The basic idea is as follows: if $\mathfrak{a}$ is given and $\mathfrak{a}^n = (a)$ is known to be principal, then we have

- either $\mathfrak{a}$ is not principal, or
- there exists $u \in \mathcal{O}_K^\times$ such that $ua \in K^{\times n}$ is an nth power (in $\mathcal{O}_K$). (In this instance, an n-th root will generate $\mathfrak{a}$.)

As usual, we reduce to the case where $n = \ell$ is prime (by raising to the power n/ℓ). We compute a GRH-conditional unit group. The implemented check will confirm that the class group and the unit group are in fact ℓ-saturated or terminate with an error message.

We have to saturate the entire class group and not just a cyclic subgroup. Therefore, the approach is a bit more elaborate.

We use the notation $\mathrm{Cl}(\mathcal{O}_K) \cong C_1 \times \cdots \times C_k$ to decompose the class group into a product of cyclic groups. For each factor C_i we chose an ideal $\mathfrak{a}_i$ representing a generator. Now, the function `ClassGroupCyclicFactorGenerators` returns a generator g_i for each principal ideal $\mathfrak{a}_i^{\#C_i}$. Further, we denote by $u_0, \ldots, u_r$ generators of the unit group. We assume that u_0 is the only generator of finite order.

Assuming the prime ℓ divides $\#C_1, \ldots, \#C_j$ then the class and the unit group are ℓ-saturated if and only if the subgroup $\langle u_0, \ldots, u_r, g_1, \ldots, g_j \rangle$ does not contain an ℓth power aside from 1. If $\ell \nmid \mathrm{ord}(u_0)$, then we can remove it from the subgroup. The search for ℓ-th powers is done in the same way as the saturation of the unit group by a reduction modulo $\mathfrak{p}$ approach and linear algebra over $\mathbb{Z}/\ell\mathbb{Z}$.

Remark 7.1. The current C-level implementation of the class group saturation is switched off and it is no longer compatible with the current C code. Therefore, this part is reimplemented in package code.

As the above computation can be done modulo h-th powers, this allows to replace `ClassGroupCyclicFactorGenerators` by a more efficient function that gives the results of the linear algebra steps only modulo h (resp. the final result modulo h-th powers). Further, it is not necessary to be compatible with the class group map, as it is not used in the saturation algorithm. As the computation is done by reduction modulo $\mathfrak{p}$, all elements are represented as power products and multiplied out after reduction modulo $\mathfrak{p}$.

Even in the case of imaginary quadratic fields with large discriminants, the time spent on discrete logarithms is negligible. Other types of fields result in much smaller class groups and the discrete logarithms are trivial to compute. The only step whose computation time is not negligible is the computation of the Smith form (with transformation matrix) of the relation matrix.

The above saturation of the class group is implemented in KANT, but unfortunately, it was switched off. As of V2.29, a new implementation is available.

8. Conclusions

Summary. To summarize, as of V2.29, the algorithms run as follows.

- To compute a formalizable class group, we use a heuristic set of generators and a heuristic evaluation of the Euler product, compute a set of relations until the analytic class number matches, and then (using exact methods) saturate the class group. Finally it is checked that the heuristically chosen set of generators is in fact a generating set for the class group.
- To compute a rigorous unit group, we saturate using the regulator of the known subgroup. (This could be made formalizable with ball arithmetic.)
- We have a GRH-conditional formalizable algorithm for the class group, but again only GRH-conditional rigorous algorithm for the unit group. For a GRH-conditional rigorous algorithm, we use the GRH-bound, evaluate the analytic class number, and declare we are done with relations when we match.
- We don't yet have a fast heuristic algorithm.

The computation of $\operatorname{res}_{s=1}\zeta_K(s)$ and error estimates are implemented in C. The saturation proof for the class group is implemented in package level. The description of the proof levels `GRH, UserBound, Subgroup, Full` in the handbook has been updated. By an early abort of the relation search (before the Euler product matches), we can test the saturation algorithms.

Benchmarks. The table below shows the timinings for the steps performed.

Polynomial	max order Euler prod residue	$\mathrm{Cl}(\mathcal{O}_K)$	$\mathcal{O}_K^\times$	$\mathrm{Cl}(\mathcal{O}_K)$ saturation
x^2+3	0.000	0.030	0.000	0.000
$x^2+10^{30}+57$	0.010	2.090	0.000	5.620
$x^2-10^{30}-57$	0.010	0.960	2.010	0.000
x^3-17	0.010	0.020	0.000	0.000
$x^3-x+2015993900449$	0.010	0.450	1.800	0.200
$x^6+4x^3+30x^2+4$	0.020	0.070	0.010	0.010
$x^8+4346x^4+169$	0.050	0.370	0.280	0.070
$x^8+8932x^4+17161$	0.050	0.640	1.650	0.120
$x^{12}+4x^6+270x^4+4$	0.080	0.640	0.130	0.110
$x^{12}+422x^6+5070x^4+44521$	0.130	3.410	18.820	0.720

The time to saturate the unit group is proprtional to the upper index bound and the time to fully check the class group is proportional to the Minkowski bound. Thus, both is only possible in small examples.

Polynomial	Conditional results	Verify $\mathrm{Cl}(\mathcal{O}_K)$	$\mathcal{O}_K^\times$ saturation
$x^2 - 13$	0.010	0.000	0.000
$x^2 - 113$	0.030	0.000	0.000
$x^2 - 1013$	0.010	0.000	0.000
$x^2 - 10103$	0.010	0.000	0.000
$x^2 - 10^5 - 103$	0.020	0.000	0.000
$x^2 - 10^6 - 1003$	0.030	0.000	0.000
$x^2 - 10^7 - 1009$	0.030	0.000	0.010
$x^2 - 10^8 - 10017$	0.030	0.020	0.020
$x^2 - 10^9 - 10029$	0.040	0.050	0.070
$x^2 - 10^{10} - 100003$	0.090	0.850	0.280
$x^2 - 10^{11} - 100009$	0.080	1.270	0.560
$x^2 - 10^{12} - 1000021$	0.180	10.190	2.620
$x^2 - 10^{13} - 1000029$	0.450	81.710	8.580
$x^2 - 10^{14} - 10000003$	1.990	—	89.170

The next larger example was terminated when using more than 20 GB of memory.

Testing. Among other test, the following was performed with repeated runs of more than 10000 number fields. The relation search was terminated early, once the relation matrix and the unit group reached full rank, but without a match in the analytic class number formula.

The resulting class groups and unit groups were compared with GRH-conditional results. In case the results did not coincide, a rigorous computation was requestet. This called the proof algorithms, as the result was viewed as conditional. The test found:

- Corrected indices of unit groups (by a search for small units) range from 2 to 648 and include various primes such as 2, 3, 7, 11, 13, 17, 23.
- Detected indices of unit groups (resulting in error messages) range from 2 to 648 (including 101 and other odd primes).
- Detected indices of class groups (resulting in error messages) range from 2 to 24. Prime factors other than 2 and 3 did not show up.

In total an incorrect class group occurred about 4000 times, an incorrect unit group occurred about 9000. For further improvements of the algorithm we can learn from this test that a full rank relation matrix will give the correct class group with higher probability than the unit group. An incorrect class group will most likely be off by a factor of the shape $2^i 3^j$. Further, the unit group derived from an incomplete relation matrix can have large index in the full unit group. In case the unit group is incorrect, the index is 2 with a probability of about 2/3.

Possible future work.

(1) The algorithm to generate relations used in the class group computation would be useful in other contexts. Could we have access to these functions at the user level, perhaps with some parameters to control the factor base and how many relations to generate? This would be potentially useful in Diophantine contexts, e.g. in the Brauer–Manin obstruction . For this application it is not needed that the kernel

of the relation matrix gives the whole unit group. More precisely, we would like to find elements whose norm is an ℓth power, without necessarily completeness results; relation finding code provides an important input for that.

(2) It would be useful to be able to work with unexpanded power products for ideals and units as a well-defined type.

(3) For abelian fields, can we do better in class group computations: for example, computing class numbers using Bernoulli numbers?

(4) Is there any potential speedup to compute just $(\mathrm{Cl}\,\mathcal{O}_K)/(\mathrm{Cl}\,\mathcal{O}_K)^\ell$? In principle, the linear algebra is faster (no awfulness with Smith form), if we have generators we only need to do ℓ-saturation which is easier. The case $\ell = 2$ shows up frequently in applications.

(5) Are there other heuristics or computational observations that would lead to faster heuristic algorithms? Sometimes one just wants to know the answer as quickly as possible without rigor, then after the experimental phase is over certifying it.

References

[B94] Eric Bach, *Improved approximations for Euler products*, in *Number theory (Halifax, NS, 1994)*, 13–28, CMS Conf. Proc., 15, Amer. Math. Soc., Providence, RI. 10

[B96] Eric Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), no. 191, 355–380. 7

[BDF08] Karim Belabas, Francisco Diaz y Diaz and Eduardo Friedman, *Small generators of the ideal class group*, Math. Comp. **77** (2008), no. 262, 1185–1197. 7

[BF15] Karim Belabas and Eduardo Friedman, *Computing the residue of the Dedekind zeta function*, Math. Comp. **84** (2015), no. 291, 357–369. 8

[BF14] Jean-François Biasse and Claus Fieker, *Subexponential class group and unit group computation in large degree number fields*, LMS J. Comput. Math., **17**(A) (2014), 385–403. 9

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), 235–265. 1

[Co93] Henri Cohen, *A course in computational algebraic number theory*, Grad. Texts in Math., vol. 138, Springer, Berlin, 1993. 3, 7

[Co93] Henri Cohen, *Advanced topics in computational number theory*, Grad. Texts in Math., vol. 193, Springer-Verlag, New York, 2000. 3

[D04] Tim Dokchitser, *Computing special values of motivic L-functions*, Experiment. Math. **13** (2004), no. 2, 137–149. 8

[Duk03] W. Duke, *Extreme values of Artin L-functions and class numbers*, Compositio Math. **136** (2003), 103–115. 12

[F01] Claus Fieker, *Computing class fields via the Artin map*, Math. Comp. **70** (2001), no. 235, 1293–1303. 3

[F06] Claus Fieker, *Applications of the class field theory of global fields*, Discovering mathematics with Magma, Algorithms Comput. Math., vol. 19, Springer, Berlin, 2006, 31–62. 3

[GL22] Stephan Ramon Garcia and Ethan Simpson Lee, *Unconditional explicit Mertens' theorems for number fields and Dedekind zeta residue bounds*, Ramanujan J. **57** (2022), no. 3, 1169–1191. 9

[GS03] A. J. Granville and K. Soundararajan, *The distribution of values of* $L(1, \chi_d)$, Geom. Funct. Anal. **13** (2003), no. 5, 992–1028. 12

[L92] H. W. Lenstra Jr., *Algorithms in algebraic number theory*, Bull. Amer. Math. Soc. (N.S.) **26** (1992), no. 2, 211–244 3

[SMC06] József Sándor, Dragoslav S. Mitrinović,and Borislav Crstici, *Handbook of number theory. I*, Springer, Dordrecht, 2006. 9

Institut für Mathematik, Universität Würzburg, Emil-Fischer-Strasse 30, D-97074 Würzburg, German y
Email address: stephan.elsenhans@mathematik.uni-wuerzburg.de
URL: https://www.mathematik.uni-wuerzburg.de/institut/personal/elsenhans.html

School of Mathematics and Statistics, University of Sydney, NSW, 2006, Australia
Email address: jvoight@gmail.com
URL: https://jvoight.github.io/