Computing class groups and unit groups in Magma

Andreas-Stephan Elsenhans
School of Mathematics and Statistics, University of Sydney
Sydney, NSW, Australia
Institut für Mathematik, Universität Würzburg
Würzburg, Germany
stephan.elsenhans@mathematik.uni-wuerzburg.de

John Voight
School of Mathematics and Statistics, University of Sydney
Sydney, NSW, Australia
jvoight@gmail.com

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.

Keywords

class groups, unit groups, computational number theory, Magma

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: heuristic generators are taken, relations are found, and then the class group is confirmed to be saturated and complete. The latter steps involve the Minkowski bound and are done using exact computations.
The bad news: a bug was uncovered (in fact, it has been a problem for 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 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 for the regulator is computed, and the unit 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 some final comments, then provide in section A a summary of some tests, observations on typical error patterns, and a brief comment on comparisons with implementations in other systems.

Acknowledgements

We would like to thank Eran Assaf, Edgar Costa, Brendan Creutz, Tim Dokchitser, Claus Fieker, Sam Frengley, Henri Johnston, Nicole Sutherland, Allan Steel, and Alain Chavarri Villarello for helpful conversations. Many thanks go to the anonymous referees for their feedback. 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. The authors were supported by a grant from the Simons Foundation (SFI-MPS-Infrastructure-00008650, JV).

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 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. The original 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) in the current version.

Claus Fieker built on the class group package by providing explicit algorithmic class field theory during 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.

In 2022, Allan Steel developed a parallel version of Donnelly's random walk algorithm: this spawns independent Magma worker jobs which each do distinct random walks on ideals to generate independent relations, while the manager job then collects all the relations from each worker and processes all the relations until there are enough (at the same time, the relation handling code had to be rewritten to use the sparse matrix type to handle the potentially huge dimension of the relation matrices now involved). This parallel sieving often does yield a speedup reasonably close to k when there are k separate workers. The linear algebra phase also uses parallel pthreading methods which speeds up the computation of abelian group structure of the relation matrix, etc. Some of this may be reported on in future work.

Our contributions to the code, discussed in this article, are twofold. First, we include some updates that include the latest results on number field L-functions. Second, we performed systematic tests that led us to uncover and fix problems in the proof algorithms.

3 Basic algorithms

We refer to the Magma handbook and more generally the article by Claus Fieker [F06] for an overview. For general background, there is also an article by Lenstra [L92] and the books of Cohen [Co93, Co00]. 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.)

Remark 3.1. We consider here only the absolute case, following the current Magma implementation. However, much of what is done can be made relative, as partially explored by Cohen [Co00, Chapter 7]. 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 quite 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—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 denote by O_K the maximal order of a number field K. Further, we write $\mathrm{Cl}(O_K)$ for the class group of O_K, and $\mathrm{h}(O_K)$ for its order. Let r_1, r_2 be the number of real, complex places of K and let $d := \mathrm{disc}(O_K)$ be the discriminant. Finally, μ_K denotes the roots of unity contained in K.

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 $r = r_1 + r_2$ 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 $O_K^\times$. The covolume of this lattice is the regulator $\mathrm{Reg}(O_K)$.

Given as input the maximal order 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 $\mathrm{Cl}\, O_K$ (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 $O_K^\times$.

We allow the output of the unit group to be in a compact representation, as a product of elements of $K^\times$ 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 obtain them is a search for elements of small T_2-norm, which is part of the saturation. But, usually they are automatically obtained during the unit group computation anyway.

Example 3.5. 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 $S := \{\mathfrak{p}_1, \ldots, \mathfrak{p}_m\}$ be a set of primes. Let

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

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

$$1 \to O_K^\times \to O_{K,S}^\times \xrightarrow{\phi} \mathbb{Z}^m \xrightarrow{\pi} \mathrm{Cl}\, O_K \tag{4.1}$$

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

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

and π is taking the class

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

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 $O_{K,S}^\times$ to map to a generating system of $\ker \pi \le \mathbb{Z}^m$. 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. For a discovery of the algorithm inside Magma, we suggest to use `SetVerbose("ClassGroup",1); SetVerbose("UnitGroup",1);` or a highter level to watch the progress of the computation.

Formalizable generators

The theorem of Minkowski provides that every ideal class of O_K has an integral representative $\mathfrak{a}$ with

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

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 4.1. 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 Oesterlé. 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 O_K such that

$$\mathrm{Nm}(\mathfrak{a}) \le (0.128 - o(1))^{r_1}(0.044 - o(1))^{r_2}\sqrt{|d|} \tag{4.5}$$

and

$$\mathrm{Nm}(\mathfrak{a}) \le (0.14 - o(1))^{r_1}(0.051 - o(1))^{r_2}\sqrt{|d|} \tag{4.6}$$

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

$$\mathrm{Nm}(\mathfrak{a}) \lesssim 2.50\sqrt{n}(0.36)^{r_1}(0.46)^{r_2}\sqrt{|d|}. \tag{4.7}$$

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

Remark 4.2. 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
```

We see a constant factor improvement, but in either case it is so large it is practically useless.

Remark 4.3. 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 4.4 (Bach). *Assuming GRH, the set*

$$\{\mathfrak{p} \subset O_K : \mathrm{Nm}(\mathfrak{p}) \le 12(\log|d|)^2\}$$

generates $\mathrm{Cl}\, O_K$.

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.

Remark 4.5. The constant 12 in the above theorem of Bach has recently been improved by Grenié–Molteni [GM25] to a value slightly below 4. This result has not yet been incorporated into the code.

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 4.6. 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 $\phi(O_{K,S}^\times) \le \mathbb{Z}^m$.

Many people have worked on different approaches to generate relations. Currently, the relation search is done by factoring short elements in randomly generated ideals. Further, a relation search by sieving is available for fields of small degree.

For this article, 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 Theorem 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 4.7. 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, implemented by Allan Steel, 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 4.8. 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 $\mathrm{h}(O_K) \cdot \mathrm{Reg}(O_K)$ 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 and we abbreviate $h = \mathrm{h}(O_K)$ and $R = \mathrm{Reg}(O_K)$.

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

$$A_K(X) := \sum_{\substack{\mathfrak{p}\subset O_K \\ \mathrm{Nm}(\mathfrak{p}^m)<X}} \frac{\log \mathrm{Nm}(\mathfrak{p})}{\mathrm{Nm}(\mathfrak{p})^{m/2}} \left(\frac{\sqrt{X}\log(X)}{\mathrm{Nm}(\mathfrak{p})^{m/2}\log(\mathrm{Nm}(\mathfrak{p})^m)} - 1 \right)$$

$$f_K(X) := \frac{3(A_K(X) - A_{\mathbb{Q}}(X) + A_{\mathbb{Q}}(X/9) - A_K(X/9))}{2\sqrt{X}\log(3X)}$$

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 evaluating $\log \mathrm{res}_{s=1}\, \zeta_K(s)$ 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

$$\sum_{\mathfrak{p}\subset O_K, \mathrm{Nm}(\mathfrak{p}^m)<X} \frac{\log \mathrm{Nm}(\mathfrak{p})}{\mathrm{Nm}(\mathfrak{p})^{m/2}} \left(\frac{\sqrt{X}\log(X)}{\mathrm{Nm}(\mathfrak{p})^{m/2}\log(\mathrm{Nm}(\mathfrak{p})^m)} \right)$$

$$\leq [K:\mathbb{Q}]\sqrt{X}\log(X)\left(\sum_{p<X}\frac{1}{p} + \sum_{p,m\geq 2}\frac{1}{p^m}\right)$$

$$\leq [K:\mathbb{Q}]\sqrt{X}\log(X)\left(\log\log X + 0.2615 + \frac{1}{2(\log X)^2} + 0.77316\right).$$

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

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

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

For 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*

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

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

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

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

Remark 5.4. For the approximation

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

the GRH based estimate

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

for $X > 1000$ was worked out in [B94, Thm. 2, Tab. 2].

Magma (V2.28) 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. Note that to *ensure* 5% error using the above GRH-based estimate would require a bound of 10^6 even in small examples.

If we consider the partial evaluation

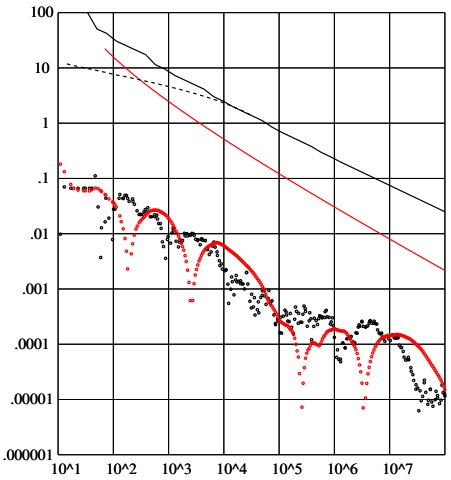
$$\frac{\prod_{p<1000} \zeta_{K,p}(1)/\zeta_p(1)}{\operatorname{res}_{s=1} \zeta_K(s)}$$

assuming GRH, we find in an experiment:

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

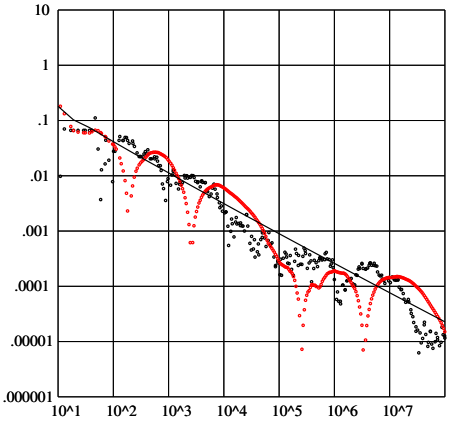
Because of the 5% error heuristic, the error has to be about 30 standard deviations to cause trouble.

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 residue. 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 occurs 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 frequency. 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.

Remark 6.1. A quasi linear version of the continued fraction algorithm is implemented in `/package/Ring/FldQuad/fund_unit.m`. When not using a power product representation of units, this approach is slightly faster compared to 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):

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

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. Further details on lower regulator bounds are given in [FP08, BM21, BM25].

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 $\log O_K^\times$ 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 $k_i := (O/\mathfrak{p}_i)$.
- An ℓ-th power of $O^\times$ 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 principal ideals of prime order.

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 (based on the GRH bound), 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`, saturation of the class group is checked.

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 $u \in O_K^\times$ such that $ua \in K^{\times n}$ is an nth power (in 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 $\mathrm{Cl}(O_K) \simeq C_1 \times \cdots \times C_k$ 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. 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, with u_0 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 ℓ-th power that is not an ℓ-th power of an element of the subgroup. 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—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. We provide a new implementation in V2.29.

8 Conclusions

In V2.29 and later, Magma's default class group computation is formalizable: it uses a heuristic generator set and a heuristic Euler-product evaluation to guide relation collection, then saturates and verifies the resulting group using exact methods. A GRH-conditional variant replaces the heuristic stopping rule by the GRH bound and the analytic class number formula. For units, Magma computes a full-rank subgroup and saturates using the regulator; making this fully formalizable would require rigorous (ball) arithmetic in the analytic estimates. Most analytic computations for $\mathrm{res}_{s=1}\, \zeta_K(s)$ and associated error bounds are implemented in C, while the saturation proofs are implemented at the package level and surfaced through the proof levels documented in the handbook.

Future work could include exposing the relation-finding machinery at user level, representing ideals and units as unexpanded power products, and developing specialized routines (e.g. for abelian fields or for $\mathrm{Cl}(O_K)/\mathrm{Cl}(O_K)^\ell$).

A Experimental results

Benchmarks

The table below shows the timings for the steps performed.

Polynomial	max order Euler prod residue	$\mathrm{Cl}(O_K)$	$O_K^\times$	$\mathrm{Cl}(O_K)$ 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 examples indicate that even in small cases, most time is spent on relation search and evaluation, not on initialisation. The time to saturate the unit group is proportional 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. The limitations are illustrated by the next examples.

Polynomial	Conditional results	Verify $\mathrm{Cl}(\mathcal{O}_K)$	$\mathcal{O}_K^\times$ saturation
$x^2 - 10^8 - 10017$	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.

Further testing

To test the rigorous algorithms separately, we performed the following runs on 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 then compared with GRH-conditional results. In case the results did not coincide, a rigorous computation was requested. This called the proof algorithms, as the result was viewed as conditional. We 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.

Comparision with other systems

In very broad terms, our experiments indicate that Magma has roughly comparable performance to Pari [P25] and OSCAR [O26], giving two other major implementations of class groups and unit groups. But we also found individual fields that seemed to be outliers for one system or the other. That being said, the algorithms have many implementation-dependent stages and these stages are exposed in different ways; accordingly, small changes in what is timed can produce large apparent differences, especially on individual examples. In particular, a detailed comparison of runtime and complexity of different parts of the algorithm as well as an estimate of the overall complexity is unfortunately out of scope of this review. In particular, a fair comparison of the performance between systems will require a detailed understanding what is exactly computed by each system and if further initializations are done. Computing different presentations of the unit group or initializing the computation of a generator for principal ideals is in many cases far more expensive than the raw class group and unit group computation, for example. So we leave it as future work to establish common benchmarks for a future comparison.

References

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

[B96] Eric Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), no. 191, 355–380. DOI: https://doi.org/10.1090/S0025-5718-1990-1023756-8.

[BM21] F. Battistoni and G. Molteni, *Generalization of a Pohst's inequality*, J. Number Theory **228** (2021), 73–86. DOI: https://doi.org/10.1016/j.jnt.2021.04.014.

[BM25] F. Battistoni and G. Molteni, *Generalized Pohst inequality and small regulators*, Math. Comp. **94** (2025), no. 351, 475–504. DOI: https://doi.org/10.1090/mcom/3954.

[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. DOI: https://doi.org/10.1090/S0025-5718-07-02003-0.

[BF15] Karim Belabas and Eduardo Friedman, *Computing the residue of the Dedekind zeta function*, Math. Comp. **84** (2015), no. 291, 357–369. DOI: https://doi.org/10.1090/S0025-5718-2014-02843-3.

[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. DOI: https://doi.org/10.1112/S1461157014000345.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), 235–265. DOI: https://doi.org/10.1006/jsco.1996.0125.

[Co93] Henri Cohen, *A course in computational algebraic number theory*, Grad. Texts in Math., vol. 138, Springer, Berlin, 1993. DOI: https://doi.org/10.1007/978-3-662-02945-9.

[Co00] Henri Cohen, *Advanced topics in computational number theory*, Grad. Texts in Math., vol. 193, Springer-Verlag, New York, 2000. DOI: https://doi.org/10.1007/978-1-4419-8489-0.

[D04] Tim Dokchitser, *Computing special values of motivic L-functions*, Experiment. Math. **13** (2004), no. 2, 137–149. DOI: https://doi.org/10.1080/10586458.2004.10504528.

[Duk03] W. Duke, *Extreme values of Artin L-functions and class numbers*, Compositio Math. **136** (2003), 103–115. DOI: https://doi.org/10.1023/A:1022695505997.

[F01] Claus Fieker, *Computing class fields via the Artin map*, Math. Comp. **70** (2001), no. 235, 1293–1303. DOI: https://doi.org/10.1090/S0025-5718-00-01255-2.

[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. DOI: https://doi.org/10.1007/978-3-540-37634-7_2.

[FP08] Claus Fieker and Michael E. Pohst, *A lower regulator bound for number fields*, J. Number Theory **128** (2008), no. 10, 2767–2775. DOI: https://doi.org/10.1016/j.jnt.2008.04.005.

[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. DOI: https://doi.org/10.1007/s11139-021-00435-6.

[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. DOI: https://doi.org/10.1007/s00039-003-0438-3.

[GM25] L. Grenié and G. Molteni, *Breaking the 4 barrier for the bound of a generating set of the class group*, Math. Comp. **94** (2025), no. 356, 3145–3163. DOI: https://doi.org/10.1090/mcom/4114.

[L92] H. W. Lenstra Jr., *Algorithms in algebraic number theory*, Bull. Amer. Math. Soc. (N.S.) **26** (1992), no. 2, 211–244. DOI: https://doi.org/10.1090/S0273-0979-1992-00284-7.

[O26] OSCAR – Open Source Computer Algebra Research system, Version 1.7.2, The OSCAR Team, 2026. (https://www.oscar-system.org) DOI: https://doi.org/10.5281/zenodo.19472862.

[P25] The PARI Group, PARI/GP version 2.17.3, Univ. Bordeaux, 2025, http://pari.math.u-bordeaux.fr/.

[SMC06] József Sándor, Dragoslav S. Mitrinović,and Borislav Crstici, *Handbook of number theory. I*, Springer, Dordrecht, 2006. DOI: https://doi.org/10.1007/1-4020-3658-2.