Computing roots of polynomials over number fields using complex embeddings

Andrea Lesavourey, Thomas Plantard, Willy Susilo

COMPUTING ROOTS OF POLYNOMIALS OVER NUMBER FIELDS USING COMPLEX EMBEDDINGS

ANDREA LESAVOUREY *

Univ Rennes, CNRS, IRISA

THOMAS PLANTARD

WILLY SUSILO

School of Computing & Information Technology, Institute of Cybersecurity and Cryptology, University of Wollongong, Australia

Abstract. We explore a fairly generic method to compute roots of polynomials over number fields through complex embeddings. Our main contribution is to show how to use a structure of a relative extension to decode in a subfield. Additionally we describe several heuristic options to improve practical efficiency. We provide experimental data from our implementation and compare our methods to the state of the art algorithm implemented in Pari/Gp.

1. Introduction

In the past few years, the rise of cryptographic protocols using the structure of number fields has driven attention to computational number theory. Indeed, in order to obtain efficient implementations of such cryptosystems as well as testing their security, fast computations over number fields are required. In particular, number field units and S-units are the main objects used to retrieve short generators of principal ideals [16, 2, 25] and short elements of general ideal lattices [15, 32, 6, 7] respectively. Their computation commonly requires a saturation process [35, 17, 10], which includes a necessary step of n-th roots computations. This task is a special case of solving a polynomial equation with coefficients in a number field.

We are therefore interested in the following problem. Given K a number field and $f(T) \in K[T]$, find the roots of $f(T)$ in K i.e. find $Z_K(f) = \{x \in K \mid f(x) = 0\}$. As a matter of fact we will only consider the cases such that all the roots can be expressed as integral combinations of a known basis of K i.e.

E-mail addresses: andrea.lesavourey@irisa.fr, thomas.plantard@gmail.com, wsusilo@uow.edu.au.

2020 *Mathematics Subject Classification.* 11R09,11Y40.

Key words and phrases. polynomial roots, number fields, relative extensions, LLL, complex embeddings.

* *Corresponding author.*

Andrea Lesavourey is funded by the Direction Générale de l'Armement (Pôle de Recherche CYBER), with the support of Région Bretagne.

such that: $\exists (b_i)_i \in K^n, \forall x \in Z_K(f), \exists (x_i)_i \in \mathbb{Z}^n \mid x = x_1 b_1 + \cdots + x_n b_n$. We believe that we are not too restrictive as this is often the case in standard computations, such as S-units computations or the Number Field Sieve [22, 41].

The state of the art algorithm to solve this problem – that we will call the *algebraic* or *p-adic* method – is implemented in softwares such as Magma or Pari/GP [11, 31]. It follows the ideas used to factorise polynomials over number fields developed for example in [3, 17, 36, 37] and uses finite places. More precisely, the procedure is as follows:

(1) pick a prime ideal $\mathfrak{p}$ which defines an isomorphism between $\mathcal{O}_K/\mathfrak{p}$ and $\mathbb{F}_{p^s}$ for some prime number $p \in \mathbb{Z}$;
(2) compute the roots modulo $\mathfrak{p}$ in $\mathbb{F}_{p^s}[X]$;
(3) lift these to elements to K modulo a power $\mathfrak{p}^k$;
(4) if $\mathfrak{p}^k$ is large enough compared to the size of the solutions and it is given by a LLL-reduced basis then we can recover a solution x given $y \equiv x \bmod \mathfrak{p}^k$.

In practice, the method requires three main computational operations:

(1) compute first an "approximation" of the solutions;
(2) compute a reduced basis of a lattice;
(3) retrieve the solutions from the approximation using the lattice.

The third step – called the *decoding* phase – can be seen as solving instances of the Bounded Distance Decoding (BDD) problem.

However, it is possible to use complex embeddings instead of prime ideals. Indeed, it is known at least since the seminal paper introducing the LLL algorithm [23] how to use lattices in order to find short integral relations between algebraic numbers or find the irreducible polynomial of a number field element. As an example, assume one knows approximations of real numbers $\alpha_1, \ldots, \alpha_n$. Now define the embedding

$$\begin{array}{ccc} \mathbb{Z}^n & \longrightarrow & \mathbb{R}^{n+1} \\ (\lambda_i)_{i\in[\![1,n]\!]} & \longmapsto & (C\sum_{i=1}^{n}\lambda_i\alpha_i, \lambda_1, \ldots, \lambda_n) \end{array}$$

which gives a lattice of $\mathbb{R}^{n+1}$ represented by the row matrix

$$\begin{bmatrix} C\alpha_1 & 1 & 0 & \ldots & 0 \\ C\alpha_2 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ C\alpha_n & 0 & \ldots & 0 & 1 \end{bmatrix}.$$

Here C is a large coefficient used to ensure the shortness of the solution. Reducing this basis would allow us to retrieve a short vector $(\lambda_i)_{i\in[\![1,n]\!]}$ such that $C\sum_{i=1}^{n}\lambda_i\alpha_i$ is small. Note that this strategy can be seen as solving an instance of the Shorstest Vector Problem (SVP). Similarly it is common to use LLL algorithm to solve knapsack problems. Assume we are given $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}^n$ and $S \in \mathbb{Z}$, and that we want to find a short vector $(\lambda_1, \ldots, \lambda_n)$ such that $\sum_{i=1}^{n}\lambda_i\alpha_i = S$. Then one can consider the similar

matrix

$$\begin{bmatrix} -C\alpha_1 & 1 & 0 & \dots & 0 \\ -C\alpha_2 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -C\alpha_n & 0 & \dots & 0 & 1 \\ CS & 0 & \dots & \dots & 0 \end{bmatrix}.$$

Again a LLL reduction would yield a small solution of the linear equation.

Our work. One can combine both approaches to retrieve the coefficient of $x \in \mathbb{Z}[\mathcal{B}]$ from an embedding $\sigma_i(x)$, where $\mathcal{B} = (b_1, \dots, b_n)$ is a $\mathbb{Q}$-basis of K. This is done through solving an instance of BDD. For all $j \in [\![1, n]\!]$, denote by β_j an approximation of $\sigma_i(b_j)$ and S an approximation if $\sigma_i(x)$. Now consider $(x_i)_{i \in [\![1,n]\!]} \in \mathbb{Z}^n$ the coefficients of x in the basis $\mathcal{B}$. Then one can expect $S - \sum_{i=1}^{n} x_i\beta_i$ to be close to 0. Thus solving a BDD instance with input the row matrix

$$\begin{bmatrix} -\beta_1 & 1 & 0 & \dots & 0 \\ -\beta_2 & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\beta_n & 0 & \dots & 0 & 1 \end{bmatrix}$$

and target vector $[S, 0, \dots, 0]$ is expected to yield the solution $[\epsilon, x_1, \dots, x_n]$ if approximations are computed with high enough precision.

In this article, we study this last method and apply it to the problem of retrieving polynomial roots in number fields. In particular, under the conjecture that the lattices we consider follow the Gaussian heuristic, we establish a *heuristic* precision for which one is ensured to retrieve an element, see Theorem 3.17. We note that a similar study has been done in [33] when considering the version solving a SVP instead of a BDD. Following their analysis, we provide in Theorem 3.11 a *proven* precision ensuring correctness of the decoding, yet larger than the heuristic one.

Additionally, one of our main contribution is to show how to take advantage of a number field extension L/K to replace the decoding of an element in L by the decoding of $[L : K]$ elements in K. We will call it the *relative method*. When applied to the problem of solving polynomial equations, applying this method comes with the cost of searching through a set of size $d^{[L:K]}$, where d is the degree of the equation considered.

In order to improve the *practical* efficiency of our algorithms, we develop several heuristic strategies and observations allowing for substantial speed-ups.

- We describe a modification of the LLL algorithm – called SpecLLL – which improves its running time when $\mathcal{B}$ is of the form $(1, \alpha, \dots, \alpha^{n-1})$: we pre-reduce each new basis vector of index $i + 1$ using the relation used to size-reduce the previous vector of index i with respect to the lattice generated by the $(i - 1)$-th first rows, see Algorithm 9. Note that this modification was inspired by [34], which is a version of LLL using the specific structure of ideal lattices. Asymptotically, we gain between 20% and 50% in the execution time, depending on which implementation of LLL is tested, as shown in Figure 2 and 3.
- We provide a tighter evaluation of the precision required to retrieve the coefficients of a targetted element – see Section 5.2.

- We use a early abort strategy in the decoding phase (with Babaï's nearest plane algorithm [1]) to avoid unnecessary computations, see Algorithm 10. This is particularly impactful when using our relative method.

Using our implementation in Gp [31][1], we study the generic behaviour of our algorithms, how our improvements impact their running time, and compare them to the function nfroots implemented in Pari/Gp [31]. In most of our experiments we consider number fields defined by polynomials whose coefficients are randomly drawn in a given segment. Respectively to this definition of "random number fields", we are then able to study the *average behaviour* of the aforementioned algorithms.

Concerning the absolute method we can observe the following facts.

- We first remark that our absolute method is *more stable* than the p-adic algorithm implemented in Pari/Gp. Indeed, there can be as much as 10% of fields over which the execution time of nfroots explodes. From experiments, these are fields for which nfroots do not find a suitable inert prime. Consequently, our algorithm in its certified version has similar running time *in average*, depending on the parameters, despite being slower than nfroots at its best.
- It is more influenced than Pari/Gp nfroots function by several parameters : the dimension $[K : \mathbb{Q}]$ and the size of the roots. This is mostly linked to the fact that the lattices we use to decode take longer to be reduced than powers of prime ideals as in nfroots. Our heuristic improvements have significant impact on the time efficiency of our algorithm (10 times faster than the certified version in dimension 100 for example, see Figure 5).

Then when we consider relative extensions L/K and our relative method we have the following main observations:

- the heuristic observations allow us to obtain a more efficient algorithm as well in the relative case: between 10 and 100 times faster for relatively small parameters chosen for our experiments, see Figure 7;
- when the relative degree $[L : K]$ and the degree of the equation are relatively small, our relative algorithm can offer significant speed-ups compared to the absolute one: again for small parameters, our relative algorithm can already be up to 10 times faster than the absolute one, see Figure 8 and Figure 9 for example.

Finally we study specific families of number fields, namely cyclotomic fields and real Kummer extensions of the form $\mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_r})$ with $(m_1, \ldots, m_r) \in \mathbb{N}^r$ and p a prime integer. We can observe the following facts:

- over cyclotomic fields, our relative method applied to the extension K_m/K_m^+ – it has relative degree $[K_m : K_m^+] = 2$ – does not outperforms Pari/Gp nfroots function, which highlights the fact that cyclotomic fields are "good" fields for this method;
- over real Kummer extensions, which are "bad" fields for nfroots, the situation is completely different, and our relative algorithm can be up to 500 times faster than Pari/Gp nfroots function, see Section 6.3.4.

Future work. First we plan to study the possibility of using the structure of relative extensions within the p-adic method. If we were to be successful, it might be adapted to general global fields as well, which are a natural generalisation of number fields [5].

[1]Code publicly available at https://github.com/AndLesav/nf_polynomial_roots

Since the main bottleneck of our method is the reduction of lattices to decode, we plan to improve SpecLLL. One direction would be to follow the strategy described in [21] for knapsack-like lattices which could lead to smaller running times. Then when there is no subfield, it is possible to adapt the strategy by using several embeddings. Intuitively, adding relations should allow us to decode more easily.

Outline. The rest of this article is as follows. We give necessary recalls on number fields and Euclidean lattices are in Section 2. In Sections 3 and 4 we describe the generic methods considered. Then Section 5 is used to detail the improvements that we developed to speed-up the running time of our algorithms. Finally, we provide experimental results obtained from our implementation in Section 6. We also give additional data in Appendix B.

2. Recalls on number fields and lattices

In this section, we recall some notions and facts that we will use in the rest of the article. First we will quickly mention number fields, then describe Euclidean lattices and reduction algorithms.

Notations 2.1. The inner product is denoted by $(\cdot \mid \cdot)$. Given $x \in \mathbb{R}$, we denote by $\{x\}$ its fractional part $x - \lfloor x \rceil$.

Algebraic closures of number field extensions considered will be designated by Ω. Given a ring morphism $\sigma : A \to B$ and $a \in A$, a^σ can be used to design $\sigma(a)$. This extends to any object f which can be identified to a vector with coefficients in A, such as polynomials in $A[X]$. Additionally, if $A \subset B$ and $f(X) \in A[X]$ we write $Z_B(f)$ the set of roots of $f(X)$ in B.

Given two ordered sets A and B we will denote by $A \otimes B$ the tensor product of A and B (when it makes sense), i.e. $A \otimes B = \{ab, b \in B \mid a \in A\}$. If A and B are not ordered $A \otimes B$ will be the collection of all the products ab such that $a \in A$ and $b \in B$. Finally, we also use this notation for the tensor product of vectors and matrices.

Given a matrix M, we will denote by $M[i]$ its i-th row and M^T its transpose. Given a vector v we will write $s(v)$ for $\log_2 \|v\|_\infty$. This notation extend to any object that can be expressed as a vector, such as polynomials.

2.1. Number fields.

We refer the reader to [4, 12, 13, 28, 38] for anything related to number fields and computational number theory.

Definition 2.2. A *number field* K is a field which is a finite extension of $\mathbb{Q}$, i.e. a finite dimensional $\mathbb{Q}$-vector space.

Notation 2.3. Given an extension L/K of number fields, we will call the dimension of L over K the *degree* of L/K. It will denoted by $[L : K]$.

Proposition 2.4. *Let L/K be an extension of number fields. Then there is an* irreducible polynomial *$P(X) \in K[X]$ such that*

$$L \cong \frac{K[X]}{(P(X))}.$$

and $[L : K] = \deg P(X)$. Moreover $P(X)$ has $[L : K]$ distinct roots in an algebraic closure Ω of K containing L. These roots define $[L : K]$ distinct $K-$isomorphisms of L into Ω. If α is such a root,

then the corresponding isomorphism σ_α is the following,

$$\begin{array}{rccc} \sigma_\alpha : & \dfrac{K[X]}{(P(X))} & \longrightarrow & K[\alpha] \subset \Omega \\ & \displaystyle\sum_{i=0}^{[L:K]-1} c_i X^i & \longmapsto & \displaystyle\sum_{i=0}^{[L:K]-1} c_i \alpha^i. \end{array}$$

Notation 2.5. We call any such polynomial a *defining polynomial of L over K*, and use denote it by $P_K(X)$.

An element $\sigma_\alpha \in \mathrm{Hom}(K, \mathbb{C})$ is a field embedding of K into $\mathbb{C}$ corresponding a root α of $P_K(X)$ in $\mathbb{C}$. There are r_1 real embeddings and r_2 pairs of (strictly) complex embeddings. The two elements of a given pair are conjugates one from each other. It is usual to write $\sigma_1, \ldots, \sigma_{r_1}$ the real embeddings and to consider that $\sigma_{j+r_2} = \overline{\sigma_j}$ for all $j \in [\![r_1 + 1, r_1 + r_2]\!]$.

Definition 2.6 (Minkowski embedding)**.** Given a number field K defined by a degree n irreducible polynomial $P(X)$, the *canonical embedding* or *Minkowski embedding* is defined as

$$\begin{array}{rccc} \sigma_K : & K & \longrightarrow & \mathbb{R}^{r_1} \times \mathbb{C}^{r_2} \cong \mathbb{R}^n \\ & x & \longmapsto & (\sigma_i(x))_{i \in [\![1, r_1+r_2]\!]}. \end{array} \tag{2.1}$$

Then K can be seen as embedded in $\mathbb{R}^n$. We will relax this classical definition by considering any algebraic closure Ω instead of $\mathbb{C}$. This way σ_K defines an embedding of K into Ω. Then one can define a relative version of the Minkowski embedding: for a field extension L/K we define

$$\sigma_{L/K} : x \longmapsto (\sigma(x))_{\sigma \in \mathrm{Hom}_K(L, \Omega)}.$$

We call this map the *Minkowski embedding relative to L/K*.

2.2. **Euclidean lattices.** We refer the reader interested in more in-depth presentations on Euclidean lattices to [27, 14].

Gram-Schmidt orthogonalisation. An important and classical computation in linear algebra is the Gram-Schmidt orthogonalisation (GSO). It allows transforming a free family of a vector space (with a scalar product) into in an orthogonal family. The naive algorithm can be found in Algorithm 1.

Algorithm 1 `GSO`

Require: A free family $\mathcal{B} = \{b_1, .., b_r\}$ of $\mathbb{R}^n$
Ensure: A family $\tilde{\mathcal{B}} = \{\tilde{b}_1, .., \tilde{b}_n\}$ with $\tilde{b}_1 = b_1$ and $(\tilde{b}_i \mid \tilde{b}_j) = 0$ for $i \neq j$

$\tilde{\mathcal{B}} \leftarrow \mathcal{B}$
for $i = 1$ to r **do**
  $\tilde{b}_i \leftarrow b_i$
  for $j = 1$ to $i - 1$ **do**
    $\tilde{b}_i \leftarrow \tilde{b}_i - \dfrac{(b_i \mid \tilde{b}_j)}{(\tilde{b}_j \mid \tilde{b}_j)} \tilde{b}_j$
  end for
end for
return $\tilde{\mathcal{B}}$

Definition 2.7 (GSO)**.** Given a basis $\mathcal{B}$, we will call the basis $\widetilde{\mathcal{B}}$ outputted by Algorithm 1 *the GSO* of $\mathcal{B}$.

For the GSO computation, one can follow a QR decomposition as described in [29] for example which gives an algebraic complexity of $O(n^3)$.

Definition and first properties.

Definition 2.8. A *Euclidean lattice* is a discrete subgroup of $\mathbb{R}^n$ where n is a positive integer. We say a lattice is an *integral* lattice when it is a subgroup of $\mathbb{Z}^n$ or *rational* when it is in $\mathbb{Q}^n$. A *basis* of a lattice $\mathcal{L}$ is a basis of $\mathcal{L}$ as a $\mathbb{Z}$-module. The cardinal of said basis is called the *rank* of the lattice.

Notation 2.9. Given a matrix B we will write $\mathcal{L}(B)$ the lattice generated by its row vectors.

Definition 2.10. Consider a Euclidean lattice $\mathcal{L}$ with basis matrix B. Then the *determinant* of $\mathcal{L}$, denoted by $\det(\mathcal{L})$, is the value $\sqrt{\det(BB^\mathsf{T})}$.

Notation 2.11. The determinant of a lattice $\mathcal{L}$ is also called its *volume* because it is the volume of the fundamental domain defined by the vectors of one of its bases. Thus it is also written $\mathrm{vol}(\mathcal{L})$.

One of the most important problem over lattices is to compute short vectors. In particular, an important quantity is the length of one of the shortest non-zero vector. It is usually denoted by $\lambda_1(\mathcal{L})$. We can give a minimal bound of λ_1 using the Gram-Schmidt orthogonalisation of a basis we have at hand.

Theorem 2.12. *Let $B = (b_i)_{i\in[\![1,r]\!]}$ be a basis of a lattice $\mathcal{L}$. Then $\lambda_1(\mathcal{L}) \geqslant \min\{\|\tilde{b_i}\| \mid i \in [\![1, r]\!]\}$.*

Then one can analyse further the geometry to obtain approximate values on λ_1.

Heuristic 1 (Gaussian heuristic)**.** *Given a lattice $\mathcal{L}$ of rank r, an approximate value for $\lambda_1(\mathcal{L})$ is*

$$\lambda_1(\mathcal{L})_{\mathrm{gauss}} = \sqrt{\frac{r}{2\pi e}} \times \sqrt[r]{\mathrm{vol}(\mathcal{L})}. \tag{2.2}$$

In our context, the problem that we will be interested in is the Bounded Distance Decoding with it approximate version.

Definition 2.13 (**BDD**: Bounded Distance Decoding)**.** Given a basis B of a lattice $\mathcal{L}$ and a point x such that $d(x, \mathcal{L}) < \lambda_1(B)/2$, find the lattice vector $v \in \mathcal{L}$ closest to x.

Definition 2.14 (**BDD**$_\gamma$: γ-Approximate Bounded Distance Decoding)**.** Given a basis B of a lattice $\mathcal{L}$, a point x and a approximation factor γ ensuring $d(x, \mathcal{L}) < \gamma\lambda_1(B)$ find the lattice vector $v \in \mathcal{L}$ closest to x.

In practice, one considers BDD_γ for $\gamma < \frac{1}{2}$. This ensures that there is only one lattice vector v satisfying $d(x, v) < \gamma\lambda_1(B)$.

Algorithms for lattices. Intuitively if a basis $\mathcal{B}$ is composed by vectors globally more orthogonal to each other than the vectors of another basis matrix $\mathcal{B}'$ are, then the vectors of $\mathcal{B}$ will be globally shorter than the vectors of $\mathcal{B}'$. Therefore, given a basis it is natural to orthogonalise it as much as possible to solve problems on lattices. Since it likely results in a basis with shorter vectors, we call such a process a *reduction process* or *reduction algorithm*.

First let us define a weak notion of basis reduction due to Hermite [19].

Definition 2.15 (Size-reduce)**.** A basis $\mathcal{B} = (b_1, \ldots, b_r)$ of a lattice is said to be *size-reduced* if its GSO satisfies the following condition: $\forall i \in [\![1, r]\!], \forall 1 \leqslant j < i, |\mu_{i,j}| \leqslant \frac{1}{2}$.

One can find the simple algorithm computing a size-reduced basis in Algorithm 2, which is essentially an approximation of the GSO algorithm (Alg. 1).

Algorithm 2 SizeReduce

Require: A free family $\mathcal{B} = \{b_1, .., b_k\}$ of $\mathcal{L}$ such that $|\mu_{i,j}| \leqslant 1/2$ for all $i \neq j$, an element $b \notin \mathcal{L}$.
Ensure: An element b_{k+1} such that $b_{k+1} \equiv b \bmod \mathcal{L}$ and $|\mu'_{k+1,i}| \leqslant 1/2$ for $i \neq k+1$

1: $b_{k+1} \leftarrow b$
2: $\tilde{\mathcal{B}}, G \leftarrow \texttt{GSO}(\mathcal{B})$
3: **for** $i = k$ to 1 **do**
4: $\quad b_{k+1} \leftarrow b_{k+1} - \left\lfloor \frac{(b_{k+1} \mid \tilde{b}_i)}{\|\tilde{b}_i\|^2} \right\rceil b_j$
5: $\quad$ Update G
6: **end for**
7: **return** $\mathcal{B} \cup \{b_{k+1}\}$, G

A fairly natural way of reducing a basis would be to follow Algorithm 1, and replace all coefficients $\mu_{i,j}$ by their closest integers. This way one could obtain a basis close to the GSO of the starting matrix, i.e. which is size-reduced. In order to obtain a polynomial time algorithm outputting a basis which is proven to be reduced for varying dimension, one has to introduce a new reduction condition. This is the result of the ground breaking work by A. K. Lenstra, H. W. Lenstra and L. Lovász in [23]. They define a new notion of *reduced basis*, that is then called *LLL-reduced*.

Definition 2.16. Consider a lattice $\mathcal{L}$ defined by a basis $\mathcal{B} = (b_1, \ldots, b_r)$ and $\delta \in]\frac{1}{4}, 1[$. Then $\mathcal{B}$ is called LLL-reduced with parameter δ (or δ-LLL reduced) if it satisfies the following conditions:

(1) It is *size-reduced*: $\forall i \in [\![1, r]\!], \forall 1 \leqslant j < i, |\mu_{i,j}| \leqslant \frac{1}{2}$;
(2) It satisfies the *Lovász conditions*: $\forall i \in [\![2, r]\!], \delta \|\tilde{b}_{i-1}\| \leqslant \|\tilde{b}_i + \mu_{i,i-1}\tilde{b}_{i-1}\|^2 = \|\tilde{b}_i\|^2 + \mu_{i,i-1}^2 \|\tilde{b}_{i-1}\|^2$.

The LLL algorithm shown in Algorithm 3 essentially consists of applying `SizeReduce` to new vector basis incrementally, verify if Lovász condition is true and continue if so. Otherwise we swap the two last vectors and reduce again.

Theorem 2.17. *Consider $\mathcal{L}$ a lattice of rank r, $\mathcal{B} = (b_1, \ldots, b_r)$ a δ-LLL reduced basis of $\mathcal{L}$ for $\delta = 3/4$, and $\tilde{\mathcal{B}}$ the GSO of $\mathcal{B}$. Then the following properties are true.*

(1) $\forall i \in [\![1, r]\!], \forall 1 \leqslant j \leqslant i, \|b_j\| \leqslant 2^{(i-1)/2} \|\tilde{b}_i\|$
(2) For any $x \in \mathcal{L} \setminus \{0\}$, $\|b_1\| \leqslant 2^{(r-1)/2} \|x\|$.

Theorem 2.17 shows that a LLL reduced basis has good properties. It provides upper bounds related to the norms of the basis vectors. In particular, one can remark that the norm of shortest basis vectors cannot be too large compared to $\lambda_1(\mathcal{L})$. Indeed, the second point shows that the LLL algorithm solves in deterministic polynomial time SVP_γ, for $\gamma = 2^{(r-1)/2}$. Many improvements and versions were developed since the original version of LLL. Among many others, one can consider the use of floating-point arithmetic [29].

Algorithm 3 LLL

Require: $\mathcal{B}$, a basis of $\mathcal{L}$ of rank r, and a constant $\delta \in]\frac{1}{4}, 1[$.
Ensure: $\mathcal{B}'$, a $\delta-$LLL reduced basis of $\mathcal{L}$.

$i \leftarrow 2$
$\mathcal{B}' \leftarrow \{b_1\}$
while $i \leqslant r$ **do**
  $\mathcal{B}' \leftarrow (b'_1, \ldots, b'_{i-1})$
  $\mathcal{B}', G \leftarrow \texttt{SizeReduce}(\mathcal{B}', b_i)$
  if Lovász condition is satisfied for δ, i **then** $i \leftarrow i + 1$
  else $\texttt{Swap}(b'_i, b'_{i-1})$ and $i \leftarrow \max\{2, i-1\}$
  end if
end while
return $\mathcal{B}$

Theorem 2.18 (Complexity of LLL [30])**.** *Consider $\mathcal{L} = \mathcal{L}(\mathcal{B}) \subseteq \mathbb{R}^n$ a lattice of rank r, and $M = \max\{\log_2 \|b\|_2 \mid b \in \mathcal{B}\}$. Then for input $\mathcal{B}$, one can compute a LLL-reduced basis of $\mathcal{L}$ in time complexity*

$$O\left(r^2 n(r+M)M\mathcal{M}(r)\right). \tag{2.3}$$

Remark 2.19. When considering a full-rank lattice, i.e. $r = n$, we get an algebraic complexity $O(n^4M + n^3M^2)$.

Babaï's nearest plane algorithm. In order to solve BDD instances, one can use an algorithm due to [1], and called the *nearest plane algorithm*. It is an inductive technique, described in Algorithm 4, which is essentially a process of size-reduction (Alg. 2)

Algorithm 4 Babaï's Nearest Plane Algorithm – NearestPlane

Require: $t \in \mathbb{R}^n$, $\mathcal{B} = (b_1, \ldots, b_r)$ a basis of a lattice $\mathcal{L}$, $\tilde{\mathcal{B}}$ the GSO of B
Ensure: $v \in \mathcal{L}$ a close vector of t

$v \leftarrow t$
for $i = r$ down to 1 **do**
  $v \leftarrow v - \left\lfloor \frac{(v|\tilde{b}_i)}{\|\tilde{b}_i\|^2} \right\rceil b_i$ ▷ Make t more orthogonal to b_i
end for
return $v - t$

Again the output depends on the quality of the basis given as input. We can prove that the output is not too far from the target given the basis is LLL-reduced.

Proposition 2.20. *Consider a rank r lattice $\mathcal{L}$, given by a basis $\mathcal{B}$. Then for input t and $\mathcal{B}$, Algorithm 4 outputs $v \in \mathcal{L}$ lying in $t + \mathcal{F}(\tilde{\mathcal{B}})$. Thus it solves BDD$_\gamma$ for $\gamma \leqslant \frac{1}{2}\min\{\|\tilde{b}_i\| \mid i \in [\![1, r]\!]\}$.*

Following [30], the algebraic complexity of NearestPlane is at most $O\left(n^3 + n^2 M\right)$.

3. Absolute method

In this section, we study the methods to recover number field elements from complex embeddings, and how to apply them for solving polynomial equations.

Definition 3.1. If x is in $\mathbb{R}$, the *approximation of x up to l-bits* for $l \in \mathbb{N}$ is the integer $\lfloor 2^l x \rceil$. If $x \in \mathbb{C}$ then its *approximation up to l-bits* is $\lfloor 2^l \Re(x) \rceil + i \lfloor 2^l \Im(z) \rceil$. In both cases it is be denoted by $[x]_l$.

Remark 3.2. We commonly identify a complex number x with the pair of real numbers $(\Re(x), \Im(x))$. We extend this identification to approximations.

3.1. **Recovering elements from complex embeddings.** First we describe how we compute the coefficients of an element given one of its complex embeddings.

Definition 3.3. Consider a number field K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\sigma \in \mathrm{Hom}(K, \mathbb{C})$, and $l \in \mathbb{N}$. We call a *basis lattice of K up to precision l relative to $\mathcal{B}$ and σ* and denote by $\mathcal{L}(\mathcal{B}, \sigma, l)$ the lattice generated by the matrix $B(\mathcal{B}, \sigma, l)$ defined as follows:

$$B(\mathcal{B}, \sigma, l) = \begin{bmatrix} -[\sigma(b_1)]_l & C & 0 & \dots & 0 \\ -[\sigma(b_2)]_l & 0 & C & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -[\sigma(b_n)]_l & 0 & \dots & 0 & C \end{bmatrix} \tag{3.1}$$

The matrix $B(\mathcal{B}, \sigma, l)$ is called the *basis matrix* of $\mathcal{L}(\mathcal{B}, \sigma, l)$.

Remark 3.4.
- When there is no ambiguity regarding K, $\mathcal{B}$ or σ, we will simply denote $\mathcal{L}(\mathcal{B}, \sigma, l)$ by $\mathcal{L}_l$ and call it the *lattice basis of K up to precision l*. Similarly the matrix $B(\mathcal{B}, \sigma, l)$ will be written B_l.
- The constant C is used to ensure the validity of the decoding, and acceleratethe reduction algorithm on B_l. It will be determined latter.
- If the embedding σ is not a real embedding, then the first column of the matrix in Equation 3.1 is in fact two columns, containing respectively the real and imaginary parts of the corresponding complex numbers.

Notation 3.5. If $\sigma(K) \subset \mathbb{R}$, we denote by $\sigma(\mathcal{B})_l$ the first column vector of a basis matrix $B(\mathcal{B}, \sigma, l)$. If $\sigma(K) \not\subset \mathbb{R}$, we write $\sigma(\mathcal{B})_l = [\Re(\sigma(\mathcal{B})_l), \Im(\sigma(\mathcal{B})_l)]$ for the matrix composed of the two first column vectors of $B(\mathcal{B}, \sigma, l)$.

As before, consider K a number field, $\mathcal{B} = (b_1, \dots, b_n)$ a $\mathbb{Q}$-basis of K, and $x \in K$ such that there is $(x_i)_{i \in [\![1,n]\!]} \in \mathbb{Z}^n$ with $x = x_1 b_1 + \cdots + x_n b_n$. Now assume that $[\sigma(x)]_l$ is known for some $\sigma \in \mathrm{Hom}(K, \mathbb{C})$ and $l \in \mathbb{N}$. Then one can use Babaï's nearest plane algorithm (see Algorithm 4) using $\mathcal{L}(\mathcal{B}, i, l)$ and $t = ([\sigma(x)]_l, 0, \dots, 0)$. It is also better to reduce $B(\mathcal{B}, \sigma, l)$ with LLL first, and use Algorithm 4 with input this reduced basis and t. This leads to Algorithm 5.

Notation 3.6. Given K a number field, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\sigma \in \mathrm{Hom}(K, \mathbb{C})$ and a precision l, we denote by $L(\mathcal{B}, \sigma, l)$ the LLL-reduced basis of $B(\mathcal{B}, \sigma, l)$. Similarly, we will write L_l when there is no ambiguity.

The output of Algorithm 5 is a vector of coefficients which are expected to be the coefficients of x in the basis $\mathcal{B}$. The correctness of the outcome depends on the parameters chosen, i.e. the precision l and the constant C. We will determine a lower bound on the precision in Section 3.2, then under a reasonnable conjecture on lattices $\mathcal{L}_l$ we will exhibit a slightly better lower bound in Section 3.3.

Given $x \in \mathcal{O}_K$, if one denote by t_x the target vector $\left[\lfloor 2^l x^\sigma \rceil, 0, \dots, 0\right]$ we wish to evaluate a precision l for which $e_x := \mathsf{NearestPlane}(\mathcal{L}_l, t_x)$ is equal to $[\epsilon, Cx_1, \dots, Cx_n]$. It is known that Babaï's nearest plane

Algorithm 5 TestDecode

Require: An integer l, the matrix L_l of a reduced basis of $\mathcal{L}(\mathcal{B}, \sigma, l)$, $[\sigma(x)]_l$ for some $x \in K$.
Ensure: A candidate $y = (y_1, \ldots, y_n)$ for the vector of coefficients of x expressed in $\mathcal{B}$.
1: $r \leftarrow [K_\sigma : \mathbb{R}]$ ▷ $r = 1$ if $\sigma(K) \subset \mathbb{R}$, and 2 otherwise
2: $t \leftarrow ([\sigma_i(x)]_l, 0, \ldots, 0) \in \mathbb{Z}^{n+r}$
3: $v \leftarrow \mathsf{NearestPlane}(t, L_l)$ ▷ (Alg. 4)
4: **return** $[v_{r+1}, .., v_{n+r+1}]/C$

algorithm outputs a vector with norm smaller than half the norm of the shortest vector of the GSO. Thus, obtaining a lower bound on the first minimum of $\mathcal{L}_l$ will allow us to obtain a bound on the norm of the output of $\mathsf{NearestPlane}(\mathcal{L}_l, t_g)$. Coupled with an upper bound on the norm of e_g, this will give a condition for a correct decoding, corresponding to

$$\|e_g\|_2 \leqslant \frac{\lambda_1(\mathcal{L}_l)}{2^n}. \tag{3.2}$$

Proposition 3.7. *Consider K a number field, $\mathcal{L}_l = \mathcal{L}(\mathcal{B}, \sigma, l)$ a basis matrix of K, and $x \in \mathbb{Z}[\mathcal{B}]$. Let m be an integer such that $m = 1$ if $\sigma(K) \subset \mathbb{R}$ and $m = 2$ otherwise. Then Algorithm 5 outputs the correct vector of coefficients $(x_1, \ldots, x_n)$ of x in $\mathcal{B}$ if the following holds:*

$$\frac{2^{m-1}\left(1 + \|x\|_1\right)^2}{4} + C^2 \|x\|_2^2 \leqslant \frac{\lambda_1(\mathcal{L}_l)^2}{2^{2n}}. \tag{3.3}$$

Proof. In TestDecode we wish to solve the BDD with respect to the lattice $\mathcal{L}(\mathcal{B}, \sigma, l)$ and target vector $t = ([\sigma(x)]_l, 0, \ldots, 0)$. As Proposition 2.20 states, the output is known to be correct if the distance between the target and the lattice is smaller than $\frac{1}{2}\min\left\{\left\|\tilde{b}_i\right\|_2 \mid i \in [\![1, r]\!]\right\}$. The lattice vector v which is expected to be the closest to t is $v = (\sum_{i=1}^n x_i[\sigma(b_i)], -Cx_1, \ldots, -Cx_n)$. The error vector $e = t - v$ is then equal to

$$\left([\sigma(x)]_l - \sum_{i=1}^n x_i[\sigma(b_i)], Cx_1, \ldots, Cx_n\right).$$

Now let us consider the case where $m = 1$ and look at the first coordinate of e. First write $(\eta, \epsilon_1, \ldots, \epsilon_n) \in [-\frac{1}{2}, \frac{1}{2}]^{n+1}$ the vector of errors due to the approximations, i.e. $[\sigma(x)]_l = 2^l\sigma(x) + \eta$ and for all $i \in [\![1, n]\!]$, $[\sigma(b_i)]_l = 2^l\sigma(b_i) + \epsilon_i$. Then we have

$$e_1 = 2^l\sigma(x) + \eta - \sum_{i=1}^n 2^l x_i\sigma(b_i) + x_i\epsilon_i = \eta - \sum_{i=1}^n x_i\epsilon_i$$

which gives $e_1^2 \leqslant \left(1 + \sum_{i=1}^n |x_i|\right)^2/4 = \left(1 + \|x\|_1\right)^2/4$. If $m = 2$, or equivalently $\sigma(K) \not\subset \mathbb{R}$, one needs to consider the real and imaginary parts. Thus we get $e \in \mathbb{R}^{n+2}$ and $(\eta, \epsilon_1, \ldots, \epsilon_n) \in [-\frac{1}{2}, \frac{1}{2}]^{n+2}$ with $\eta = (\eta_1, \eta_2)$. Following the previous analysis we obtain $e_1^2 \leqslant \left(1 + \sum_{i=1}^n |x_i|\right)^2 \times 2/4 = \left(1 + \|x\|_1\right)^2/2$. Thus we obtain the following upper bound for $\|e\|_2^2$:

$$\|e\|_2^2 \leqslant \frac{2^{m-1}}{4}\left(1 + \|x\|_1\right)^2 + C^2\|x\|_2^2.$$

Finally remark that Theorem 2.17 tells us that a LLL-reduced basis $(b_1, \ldots, b_n)$ of a lattice $\mathcal{L}$ (with $\delta = \frac{3}{4}$) satisfies $\left\|\tilde{b}_i\right\|_2^2 \geqslant \left\|\tilde{b}_1\right\|_2^2/2^{n-1} \geqslant \lambda_1(\mathcal{L})/2^{n-1}$. □

Notation 3.8. Given $x \in \mathbb{Z}[\mathcal{B}]$ and $C \in \mathbb{N}^*$, we will write m_x for the upper bound of $\|e_x\|_2^2$, i.e. such that $m_x^2 := \frac{2^{m-1}\left(1+\|x\|_1\right)^2}{4} + C^2\|x\|_2^2$

Now if one has a lower bound for $\lambda_1(\mathcal{L}_l)$ depending on the parameters, one can deduce a condition for the correctness of the output of TestDecode. In particular it is possible to obtain a lower bound of the precision l for which the computation is correct.

3.2. **Proven precision and correctness.** Let us determine for which precision Algorithm 5 outputs the correct vector of coefficients.

Notations 3.9. Consider a number field K and $\mathcal{B}$ a $\mathbb{Q}$-basis of K. Let us write $D_{\mathcal{B}}$ for a denominator of $\mathcal{B}$, i.e. such that for all $i \in [\![1,n]\!]$, $D_{\mathcal{B}} b_i \in \mathbb{Z}[X]$ where $K \cong \mathbb{Q}[X]/(P_K(X))$. Additionnaly we will denote $\max_{i\in[\![1,n]\!]} T_2(b_i)$ by $m(\mathcal{B}, T_2)$.

Lemma 3.10. *Consider K a number field, $\mathcal{L}_l = \mathcal{L}(\mathcal{B}, \sigma, l)$ a basis matrix of K, and $x \in \mathbb{Z}[\mathcal{B}]$. For any integral vector $\lambda \in B(0, 2^n m_x/C)$ we get the following lower bound for $\|\lambda B_l\|_2$*

$$\|\lambda B_l\|_2 \geqslant \frac{2^l C^{n-1}}{n\sqrt{n}^n 2^{n(n-1)} m_x^{n-1} m(\mathcal{B}, T_2)^{n-1} D_{\mathcal{B}}^n (1 + \|P_K\|_2)^{2n-1}}. \tag{3.4}$$

Proof. We will focus on the case where σ is a real embedding, the proof in the case where $\sigma(K) \not\subset \mathbb{R}$ being almost identical. We follow the proof of [33, Lemma A.7], where similar lattices are considered. A given $\lambda \in \mathbb{Z}^n$ generates the vector of $\mathcal{L}_l$ of the form

$$[-\sum_{i=1}^{r} \lambda_i (2^l b_i^\sigma + \epsilon_{b_i}), C\lambda_1, \ldots, C\lambda_n].$$

If we write $s(\lambda, l) := \sum_{i=1}^n \lambda_i (2^l b_i + \epsilon_{b_i})$ then $|s(\lambda, l)| \geqslant 2^l |\sum_{i=1}^n \lambda_i b_i^\sigma| - \sum_{i=1}^n \frac{|\lambda_i|}{2} \geqslant 2^l |\lambda^\sigma| - \frac{\|\lambda_i\|_1}{2}$, where λ is seen as an element of K. Then we can write $\|\lambda B_l(\mathcal{B})\|_2 \geqslant \|\lambda B_l(\mathcal{B})\|_1 / \sqrt{n} \geqslant 2^l |\lambda^\sigma| / \sqrt{n}$. Let us find a lower bound for $|\lambda^\sigma|$. If $\mu = D_{\mathcal{B}}\lambda$ then from [33, Lemma A.8] we have

$$|\mu^\sigma| \geqslant \frac{1}{n(1 + \|P_K\|_2^{2n-1}) T_2(\mu)^{n-1}}. \tag{3.5}$$

Then $T_2(\mu) = T_2(D_{\mathcal{B}} \sum_{i=1}^n \lambda_i b_i) \leqslant D_{\mathcal{B}} \sum_{i=1}^n |\lambda_i| T_2(b_i) \leqslant D_{\mathcal{B}} m(\mathcal{B}, T_2) \|\lambda\|_1$, so Equation (3.5) gives

$$|\lambda^\sigma| \geqslant \frac{1}{n \|\lambda\|_1^{n-1} D_{\mathcal{B}}^n m(\mathcal{B}, T_2)^{n-1} (1 + \|P_K\|_2)^{2n-1}}.$$

Finally, recalling that we consider $\lambda \in B(0, 2^n m_x/C)$ and that $\|\lambda\|_1 \leqslant \sqrt{n} \|\lambda\|_2$ we get the following lower bound for $\lambda_1(\mathcal{L}_l)$

$$\lambda_1(\mathcal{L}_l) \geqslant \frac{2^l C^{n-1}}{n\sqrt{n}^n 2^{n(n-1)} m_x^{n-1} D_{\mathcal{B}}^n m(\mathcal{B}, T_2)^{n-1} (1 + \|P_K\|_2)^{2n-1}}.$$

□

Theorem 3.11 (Proven precision for correctness)**.** *Consider K a number field, $\mathcal{B}$ a $\mathbb{Q}$-basis of K and $x \in \mathbb{Z}[\mathcal{B}]$. Additionally fix $\sigma \in \mathrm{Hom}(K, \mathbb{C})$, $C \in \mathbb{N}$, and $m \in \mathbb{N}$ such that $m = 1$ if $\sigma(K) \subset \mathbb{R}$ and $m = 2$ otherwise. If $l \in \mathbb{N}$ with $l \geqslant 1$ satisfies*

$$\begin{aligned} l \geqslant\ & n^2 + (n+1)\log_2 C + n\left(\log_2 \|x\|_2 + \frac{3\log_2(n)}{2} + \log_2 D_{\mathcal{B}} + 2\right) \\ & + (n-1)\log_2 m(\mathcal{B}, T_2) + (2n-1)\log_2 \|P_K\|_2 \end{aligned} \tag{3.6}$$

then Algorithm 5 outputs the vector of coefficients of x in the basis $\mathcal{B}$. In particular, the coefficients of x given one of its embedding can be recovered in time polynomial with respect to the size of the entries.

Proof. Using Equations (3.2) and (3.5) we can conclude that the decoding is correct if

$$2^n m_x \leqslant \frac{2^l C^{n-1}}{n\sqrt{n}^n 2^{n(n-1)} m_g^{n-1} D_{\mathcal{B}}^n m(\mathcal{B},T_2)^{n-1}(1+\|P_K\|_2)^{2n-1}}$$

which is equivalent to

$$2^l \geqslant \frac{2^n m_x^n n\sqrt{n}^n 2^{n(n-1)} D_{\mathcal{B}}^n m(\mathcal{B},T_2)^{n-1}(1+\|P_K\|_2)^{2n-1}}{C^{n-1}}$$

which is verified as soon as

$$l \geqslant n^2 + n(\log_2(m_x) + \log_2(n)/2 + \log_2 D_{\mathcal{B}}) + (n-1)(\log_2 m(\mathcal{B},T_2) - \log_2(C)) + (2n-1)\log_2(1+\|P_K\|_2).$$

Now we can replace m_x by its value. However let us first rewrite it for simplicity, using $\|x\|_2$ only. Since $\|x\|_1 \leqslant \sqrt{n}\,\|x\|_2$, one can safely replace m_x^2 by $\frac{2^{m-1}\left(1+\sqrt{n}\|x\|_2\right)^2}{4} + C^2\|x\|_2^2 = \|x\|_2^2\,(C^2 + \frac{2^{m-1}}{4}(n + 2\sqrt{n}/\|x\|_2 + 1/\|x\|_2^2)) \leqslant \|x\|_2^2\,(C^2+2n)$ Replacing m_x^2 by this last value we get that e_g is equal to some $[\epsilon, Cg_1, \ldots, Cg_n]$ as soon as

$$\begin{aligned} l \geqslant\ & n^2 + n\left(\log_2\|x\|_2 + \frac{3\log_2(n)}{2} + \log_2(C) + \log_2 D_{\mathcal{B}} + 2\right) \\ & + (n-1)\log_2 m(\mathcal{B},T_2) + (2n-1)\log_2\|P_K\|_2 + \log_2 C, \end{aligned}$$

which gives the claimed inequality. □

Remark 3.12. If $\mathcal{B}$ is an integral basis of $\mathcal{O}_K$ then one can bound $D_{\mathcal{B}}$ by $n^n\sqrt{n}\,\|P_K\|_2^{2(n-1)}$[26]. Additionally if $\mathcal{B}$ is LLL-reduced for the T_2-norm, we have that $m(\mathcal{B},T_2) \leqslant D_K^{O(1)}$ [33] so Equation (3.6) becomes

$$\begin{aligned} l \geqslant n^2 + n\left(\log_2\|g\|_1 + \frac{\log_2(n)}{2} + (n+1/2)\log_2 n\right) & \\ + (n-1)O(1)\log_2 D_K + (n+1)(2n-1)\log_2\|P_K\|_2 + \log_2 C. & \end{aligned}$$

In the case where $\mathcal{B}$ is a power basis, i.e. $\mathcal{B} = (1,\alpha,\ldots,\alpha^{n-1})$ we have $D_{\mathcal{B}} \leqslant D_\alpha^{n-1}$ and $m(\mathcal{B},T_2) = \sqrt{n}\max_\sigma|\alpha^\sigma|^{n-1}$ so Equation (3.6) can be replaced by

$$\begin{aligned} l \geqslant\ & n^2 + (n+1)\log_2 C + n\left(\log_2\|x\|_2 + \frac{3\log_2(n)}{2} + (n-1)\log_2 D_\alpha + 2\right) \\ & + (n-1)^2\log_2\max_\sigma|\alpha^\sigma| + (n-1)\frac{\log_2 n}{2} + (2n-1)\log_2\|P_K\|_2. \end{aligned}$$

3.3. **Heuristic precision and correctness.** Let us now determine a lower bound under a conjecture regarding lattices of the form $\mathcal{L}_l$ we handle. First we determine the determinant of said lattices.

Lemma 3.13 (Matrix determinant lemma [18])**.** *Let A be a ring, $M \in \mathrm{M}_n(A)$ be an invertible matrix and $U, V \in \mathrm{M}_{n,m}(A)$. Then the following is true:*

$$\det(M + UV^\mathsf{T}) = \left(\mathrm{Id}_m + V^\mathsf{T}M^{-1}U\right)\det(M). \tag{3.7}$$

Lemma 3.14. *Let K be a number field and $\mathcal{L}_l = \mathcal{L}(\mathcal{B},\sigma,l)$ be a basis lattice of K. Also let m be an integer equal to 1 if σ is real, and 2 otherwise. Then*

$$\mathrm{vol}(\mathcal{L}_l)^2 = C^{2n}\det\left(\mathrm{Id}_m + \frac{1}{C^2}\sigma(\mathcal{B})_l^\mathsf{T}\sigma(\mathcal{B})_l\right) \tag{3.8}$$

Proof. By definition $\mathrm{vol}(\mathcal{L}_l)^2$ is the determinant of the matrix $B(\mathcal{B},\sigma,l)B(\mathcal{B},\sigma,l)^\mathsf{T} = C^2\mathrm{Id}_n + \sigma(\mathcal{B})_l \times \sigma(\mathcal{B})_l^\mathsf{T}$. Then Lemma 3.13 gives the claimed identity. □

One can deduce from Lemma 3.14 a lower bound on the volume of $\mathcal{L}(\mathcal{B},\sigma,l)$ which depends on the precision l and the size of $\sigma(\mathcal{B})$.

Notation 3.15. Consider a number field K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\sigma \in \mathrm{Hom}(K,\mathbb{C})$ and $l \in \mathbb{N}$. Let us define the value $\Delta(\mathcal{B},\sigma,l)$. If $\sigma(K) \subset \mathbb{R}$ we set $\Delta(\mathcal{B},\sigma,l) = \|\sigma(\mathcal{B})\|_2^2 - \|\sigma(\mathcal{B})\|_1/2^l$, and if $\sigma(K) \not\subset \mathbb{R}$ we set $\Delta(\mathcal{B},\sigma,l) = \|\Re(\sigma(\mathcal{B}))\|_2^2 + \|\Im(\sigma(\mathcal{B}))\|_2^2 - \|\Re(\sigma(\mathcal{B}))\|_1/2^l - \|\Im(\sigma(\mathcal{B}))\|_1/2^l$.

Proposition 3.16. *Let K be a number field and $\mathcal{L}_l = \mathcal{L}(\mathcal{B},\sigma,l)$ be a basis lattice of K. Also let m be an integer equal to 1 if σ is real, and 2 otherwise. Then the following is true,*

$$\mathrm{vol}(\mathcal{L}_l)^2 \geqslant C^{2n}\left(1 + \frac{2^{2l}}{C^2}\Delta(\mathcal{B},\sigma,l)\right). \tag{3.9}$$

Proof. From Equation 3.8 we can write for $m = 1$

$$\mathrm{vol}(\mathcal{L}_l)^2 = C^{2n}\left(1 + \frac{1}{C^2}\|\sigma(\mathcal{B})_l\|_2^2\right) = C^{2n}\left(1 + \frac{1}{C^2}\sum_{i=1}^{n}[\sigma(b_i)]_l^2\right).$$

Then for each $i \in [\![1,n]\!]$, there is $\epsilon_i \in [-\frac{1}{2},\frac{1}{2}]$ such that $[\sigma(b_i)]_l = 2^l\sigma(b_i) + \epsilon_i$. Thus we obtain

$$\begin{aligned}\mathrm{vol}(\mathcal{L}_l)^2 &= C^{2n}\left(1 + \frac{1}{C^2}\sum_{i=1}^{n}(2^l\sigma(b_i) + \epsilon_i)^2\right)\\ &= C^{2n}\left(1 + \frac{2^{2l}}{C^2}\sum_{i=1}^{n}(\sigma(b_i)^2 + 2\sigma(b_i)\frac{\epsilon_i}{2^l} + \frac{\epsilon_i^2}{2^{2l}})\right).\end{aligned}$$

Since for all $i \in [\![1,n]\!]$, one has $|\epsilon_i| \leqslant \frac{1}{2}$, we obtain the inequality

$$\mathrm{vol}(\mathcal{L}_l)^2 \geqslant C^{2n}\left(1 + \frac{2^{2l}}{C^2}\sum_{i=1}^{n}(\sigma(b_i)^2 - 2\times\frac{1}{2}\times\frac{|\sigma(b_i)|}{2^l})\right).$$

If $\sigma(K) \not\subset \mathbb{R}$ then from Equation 3.8 we have $\mathrm{vol}(\mathcal{L}_l)^2 = C^{2n}\det\left(\mathrm{Id}_2 + \frac{1}{C^2}\sigma(\mathcal{B})_l^{\mathsf{T}}\sigma(\mathcal{B})_l\right)$. If we write $M = \mathrm{Id}_2 + \frac{1}{C^2}\sigma(\mathcal{B})_l^{\mathsf{T}}\sigma(\mathcal{B})_l$, then

$$M = \begin{bmatrix} 1 + \dfrac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2}{C^2} & \dfrac{(\Re(\sigma(\mathcal{B})_l) \mid \Im(\sigma(\mathcal{B})_l))}{C^2} \\ \dfrac{(\Re(\sigma(\mathcal{B})_l) \mid \Im(\sigma(\mathcal{B})_l))}{C^2} & 1 + \dfrac{\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^2} \end{bmatrix}$$

so we get

$$\begin{aligned}\det(M) =& 1 + \frac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2}{C^2} + \frac{\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^2} + \frac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^4}\\ &- \frac{(\Re(\sigma(\mathcal{B})_l) \mid \Im(\sigma(\mathcal{B})_l))^2}{C^4}.\end{aligned}$$

By Cauchy-Schwarz inequality we have

$$\frac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^4} - \frac{(\Re(\sigma(\mathcal{B})_l) \mid \Im(\sigma(\mathcal{B})_l))^2}{C^4} \geqslant 0,$$

therefore

$$\mathrm{vol}(\mathcal{L}_l)^2 \geqslant C^{2n}\left(1 + \frac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2}{C^2} + \frac{\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^2}\right).$$

Following the same reasoning that we did for the real case, we can conclude that

$$\frac{\|\Re(\sigma(\mathcal{B})_l)\|_2^2}{C^2} \geqslant \frac{2^{2l}}{C^2}\left(\|\Re(\sigma(\mathcal{B}))\|_2^2 - \frac{\|\Re(\sigma(\mathcal{B}))\|_1}{2^l}\right)$$

and

$$\frac{\|\Im(\sigma(\mathcal{B})_l)\|_2^2}{C^2} \geqslant \frac{2^{2l}}{C^2}\left(\|\Im(\sigma(\mathcal{B}))\|_2^2 - \frac{\|\Im(\sigma(\mathcal{B}))\|_1}{2^l}\right),$$

which gives the desired result. □

It is now possible to certify the correctness of the decoding. The Gaussian heuristic provides an *estimation* of $\lambda_1(\mathcal{L}_l)$, which can be used to obtain a *heuristic* condition for the correctness of the algorithm, as in Theorem 3.17.

Theorem 3.17 (Correctness of decoding)**.** *Consider K a number field, $\mathcal{B}$ a $\mathbb{Q}$-basis of K and $x \in \mathbb{Z}[\mathcal{B}]$. Additionally fix $\sigma \in \mathrm{Hom}(K, \mathbb{C})$, $C \in \mathbb{N}$, and $m \in \mathbb{N}$ such that $m = 1$ if $\sigma(K) \subset \mathbb{R}$ and $m = 2$ otherwise. If $l \in \mathbb{N}$ with $l \geqslant 1$ satisfies*

$$l \geqslant n^2 + n\log_2\|x\|_2 + \frac{n}{2}\log_2(2\pi e(\frac{1}{n}+2)) + \log_2(C) - \frac{\log_2\Delta(\mathcal{B},\sigma,l)}{2} \tag{3.10}$$

then Algorithm 5 outputs the vector of coefficients of x in the basis $\mathcal{B}$, if $\mathcal{L}(\mathcal{B},\sigma,l)$ satisfies the Gaussian heuristic (eq. 2.2).

Proof. Let us fix $l \in \mathbb{N}$. The Gaussian heuristic states $\lambda_1(\mathcal{L}) \sim \sqrt{\frac{n}{2\pi e}}\det\mathcal{L}^{1/n}$. Then one can combine Equation (3.3) to obtain a conditional inequality. In order to simplify the expression, we will replace m_x^2 by $\|x\|_2^2(C^2+2n)$ as explained in the proof of Theorem 3.11. This leads to the following inequality:

$$\|x\|_2^2(C^2+2n) \leqslant \frac{1}{2^{2n}}\left(\frac{n}{2\pi e}C^2\sqrt[n]{\left(1+\frac{2^{2l}}{C^2}(\Delta(\mathcal{B},\sigma,l))\right)}\right).$$

We can obtain the condition

$$\|x\|_2^2(C^2+2n) \leqslant \frac{n}{2\pi e}C^2\sqrt[n]{\left(\frac{2^{2l}}{C^2}(\Delta(\mathcal{B},\sigma,1))\right)}. \tag{3.11}$$

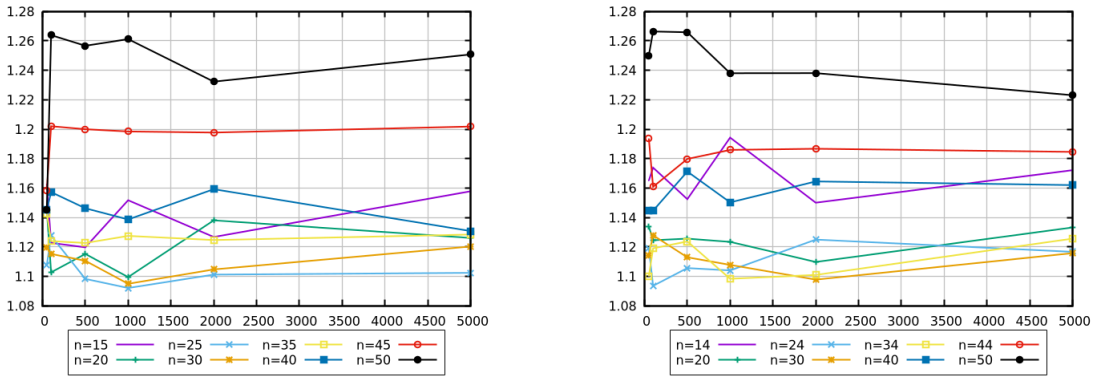
Under the logarithmic map, the left side of Equation (3.11) gives $2\log_2\|x\|_2 + 2\log_2(C) + \log_2(1+\frac{2n}{C^2})$ while the right side becomes $\log_2(\frac{n}{2\pi e}) - 2n + \frac{1}{n}\times(2(n-1)\log_2(C) + 2l + \log_2(\Delta(\mathcal{B},\sigma,l)))$, and combining them allows us to retrieve the claimed condition. □

On can remark that Theorem 3.17 gives a better condition than the theoretical one given by Theorem 3.11.

Remark 3.18. As a matter of fact, correctness of decoding is ensured by Equation (3.10) as long as $\mathcal{L}_l$ satisfies $\lambda_1(\mathcal{L}_l) \geqslant \lambda_1(\mathcal{L}_l)_{\mathrm{gauss}}$.

In order to estimate the correctness of the value given by Theorem 3.17, we verified if the Gaussian heuristic holds for lattices $\mathcal{L}_l$. We computed the average value of the quotient $\lambda_1(\mathcal{L}_l)/\lambda_1(\mathcal{L}_l)_{\mathrm{gauss}}$ for increasing values of l, over random number fields K with fixed degrees. Here "random" means defined by integral polynomials whose coefficients are drawn uniformly in $[\![-2^s, 2^s]\!]$ for randomly chosen $1 \leqslant s \leqslant 10$. The results can be found in Figure 1. One can see that $\lambda_1(\mathcal{L}_l)$ is larger than the value predicted by the Gaussian heuristic. This tends to show that asymptotically, one can safely consider that Theorem 3.17 provides a correct value certifying the correctness of the output of TestDecode. Note that running this kind of experiment for large dimensions (say larger than 120) would be intractable. Indeed, even approximating $\lambda_1(\mathcal{L})$ is a hard problem.

Heuristic 2 (Gaussian heuristic for bases lattices)**.** *Consider K a number field, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\sigma \in \mathrm{Hom}(K,\mathbb{C})$ and $l \in \mathbb{N}$. Then $\lambda_1(\mathcal{L}(\mathcal{B},\sigma,l)) \geqslant \lambda_1(\mathcal{L}_l)_{\mathrm{gauss}}$.*



(A) $\sigma(K) \subset \mathbb{R}$ (B) $\sigma(K) \not\subset \mathbb{R}$

FIGURE 1. Average value of $\lambda_1(\mathcal{L}_l)/\lambda_1(\mathcal{L}_l)_{\text{gauss}}$ plotted against the precision l, for several $n = [K : \mathbb{Q}]$.

3.4. **Computing polynomial roots.** We described which lattice we use, and the decoding method. Obviously the reduced matrix $L(\mathcal{B}, \sigma, l)$ is computed once and used to retrieve all roots. Therefore we have the following main steps:

(1) fix an embedding $\sigma : K \hookrightarrow \mathbb{C}$;
(2) compute a precision $l \in \mathbb{N}$ ensuring that the decoding is correct;
(3) compute a LLL-reduced basis L_l of the lattice generated by the basis of K;
(4) using σ, compute approximations of the roots of $f(T)$ up to a precision l;
(5) use L_l to retrieve the roots of $f(T)$ using TestDecode.

We will call PrecisionEvaluation the function returning the needed precision for input $f(T) \in K[T]$. Following Theorem 3.17, it depends on the Euclidean norms of the roots of $f(T)$. In order to evaluate an upper bound of these norms, we can follow [3]. We will denote by FloatPolynomialRoots the procedure computing the real (resp. complex roots) of a real (resp. complex) polynomial. We obtain Algorithm 6 describing the method we implemented to compute $Z_K(f)$.

Following the results from the previous section, one can state the following theorem.

Theorem 3.19. *Consider a number field K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K and $f(T) \in K[T]$ such that $Z_K(f) \subset \mathbb{Z}[\mathcal{B}]$. Then for input $(K, \mathcal{B}, f(T))$, Algorithm 6 outputs $Z_K(f)$ in polynomial time.*

Finally we give a broad estimate of the algebraic complexity in function of the precision required to ensure correctness. We will only count the complexity linked to lattices operations, as other steps are negligeable. Note as well that we express these complexities in terms of the precision at which computations are made instead of the size of the roots since the latter is unknown. Additionnaly this allows us to take into account different choices for PrecisionEvaluation.

Proposition 3.20. *Consider a number field K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K and $f(T) \in K[T]$ such that $Z_K(f) \subset \mathbb{Z}[\mathcal{B}]$. Then for input $(K, \mathcal{B}, f(T))$, the algebraic complexity of lattice operations of Algorithm 6 (reduction of a basis lattice $\mathcal{L}_l$, GSO computation and decoding) are respectively in $O(n^4 l + n^3 l^2)$, $O(n^3)$ and $O(dn^2)$, where n is the dimension $[K : \mathbb{Q}]$.*

Proof. Recall by Theorem 2.18 the algebraic complexity of LLL is in $O(n^4 M + n^3 M^2)$, where M is an upper-bound on the bit size of basis elements. A broad estimate of M for $B(\mathcal{B}, \sigma, l)$ is l. For the GSO

Algorithm 6 AbsoluteRoots

Require: A number field K, $f(T) \in K[T]$, $\mathcal{B}$ a $\mathbb{Q}$-basis of K such that $Z_K(f) \in \mathbb{Z}[\mathcal{B}]$.
Ensure: The set $Z_K(f)$

```
σ ← ChooseEmbedding(K)
l ← PrecisionEvaluation(f(T))
L_l ← LLL(B(B, σ, l))
Z ← FloatPolynomialRoots(f^σ, l)
S ← ∅
for z ∈ Z do
    y ← TestDecode(L_l, z)
    if f(y) = 0 then
        S ← S ∪ {y}
    end if
end for
return S
```

computation, we can follow a QR decomposition as described in [29] for example which gives a complexity of $O(n^3)$. Finally Algorithm 6 makes $d = \deg f(T)$ calls to Algorithm 4, adding up to $O(dn^2)$. □

4. Relative method

Let us now describe a method to recover the roots of a polynomial in an extension of number fields L/K, which is one of our main contribution.

4.1. **Decoding in a subfield.** First let us describe how one can reduce knowledge of embeddings related to the extension L/K to decodings in K. Fix Ω an algebraic closure of L/K. We will use the relative Minkowski embedding $\sigma_{L/K} : x \in L \mapsto (\sigma(x))_{\sigma \in \mathrm{Hom}_K(L,\Omega)}$. It defines a K-linear embedding of $L \cong K^n$ into Ω^n where n is the degree of L/K. More precisely, let us fix $\mathcal{E} = (e_1, \dots, e_n)$ a K-basis of L and let $x = x_1 e_1 + \cdots + x_r e_n \in L$. We assume that the action of each $\sigma \in \mathrm{Hom}_K(L, \Omega)$ on $\mathcal{E}$ is known. Then $\sigma_{L/K}$ sends K^n into Ω^n,

$$\begin{array}{rccc} \sigma_{L/K} : & K^n & \longrightarrow & \Omega^n \\ & (x_i)_{i \in [\![1,n]\!]} & \longmapsto & (\sum_{i=1}^n x_i \sigma(e_i))_{\sigma \in \mathrm{Hom}_K(L,\Omega)}. \end{array}$$

Thus given knowledge of $\sigma_{L/K}(x)$ one can apply $\sigma_{L/K}^{-1}$ and obtain recover $(x_i)_i$. For computational purposes, we need to consider embeddings into $\mathbb{C}^n$. In order to do so, one needs to fix τ an embedding of K into $\mathbb{C}$ then consider $\sigma_{L/K}$ as the collection of complex embeddings of L extending τ. This leads to Algorithm 7 which retrieves coefficients of x knowing its Minkowski embedding relative to L/K.

Lemma 4.1. *Algorithm 7 runs in polynomial time and outputs the correct vector of coefficients provided l is large enough.*

Proof. The polynomial running time is clear. For any $\sigma : L \hookrightarrow \mathbb{C}$ extending τ, its action on $x \in L$ can be seen as $\sum_{i=1}^n \tau(x_i)\sigma(e_i)$. This way, K is identified with a subfield of $\mathbb{C}$ and L is identified with $\tau(K)^n$. The action of $\sigma_{L/K}$ can be expressed as an action from $\tau(K)^n$ into $\mathbb{C}^n$ as a matrix in $\mathrm{M}_n(\mathbb{C})$, that is $\Sigma_{L/K}$. In particular we have $\Sigma_{L/K}[i] = \sigma_{L/K}(e_i)$, thus $XM = X\Sigma_{L/K}^{-1} = (x_1^\tau, \dots, x_{[K:\mathbb{Q}]}^\tau)$. Thus knowing approximations of $\sigma_{L/K}(x)$ and $\Sigma_{L/K}$ up to $l + s$, it is possible to find the approximations of $(\tau(x_i))_i$

Algorithm 7 Mink2coeff

Require: An extension L/K, given by $\mathcal{B}$ a basis of K and $\mathcal{E}$ a K-basis of L, $M = \Sigma_{L/K}^{-1}$ the matrix of $\sigma_{L/K}^{-1}$ expressed with respect to $\tau(\mathcal{E})$ up to a precision $l+s$, the matrix L_l from a reduced basis of $\mathcal{L}(\mathcal{B}, \tau, l)$ for some $\tau \in \mathrm{Hom}(K, \mathbb{C})$ and $X = [\sigma_{L/K}(x)]_{l+s}$ for some $x \in L/K$.

Ensure: A candidate $y = (y_1, \ldots, y_N)$ for the vector of coefficients of x expressed in $\mathcal{B} \otimes \mathcal{E}$.

1: $Y \leftarrow XM$
2: $y = [\ \ ]$
3: **for** $i = 1$ to n **do**
4: $\quad y \leftarrow [y \mid \mathsf{TestDecode}(L_l, Y_i)]$ ▷ Concatenation of row vectors
5: **end for**
6: **return** y

up to l. Then by the results of Sections 3.2 and 3.3 we know that $\mathsf{TestDecode}(L_l, [x_i]_l)$ is the vector of coefficients of x_i with respect to $\mathcal{B}$ provided that l is large enough. □

4.2. **An algorithm for polynomial roots.** We can apply the previous strategy to compute polynomial roots by decoding in the subfield K. Again, we fix some objects. The extension of number fields L/K is given by a $\mathbb{Q}$-basis $\mathcal{B}$ and $\mathcal{E}$ a K-basis of L. Then consider a polynomial $f(T) \in L[X]$ such that $Z_L(f) \subset \mathbb{Z}[\mathcal{E} \otimes \mathcal{B}]$. From what we described previously, in order to retrieve the coefficients of $x \in Z_L(f)$, one can compute $\sigma_{L/K}$ and use Mink2coeff with the following main steps:

(1) compute a precision l certifying the correctness of the computation;
(2) compute L_l a LLL reduced basis of $\mathcal{L}(\mathcal{B}, \tau, l)$;
(3) compute $M = \Sigma_{L/K}^{-1}$ up to precision $l+s$ for some s;
(4) compute $Z = \prod_{\sigma \in \mathrm{Hom}_K(L,\mathbb{C})} Z_{\mathbb{C}}(f^\sigma)$ up to precision $l+s$;
(5) For each $x \in Z$, use Mink2coeff to obtain a root candidate.

This leads to Algorithm 8.

Remark 4.2. The set Z is the cartesian product of the sets $Z(f^\sigma)$ with σ in $\mathrm{Hom}_K(L, \mathbb{C})$. Each of such set $Z(f^\sigma)$ has at most $d = \deg f$ elements. Therefore, Z is a set of at most d^n complex numbers. Moreover one cannot tell a priori if an element $z \in Z(f^\sigma)$ is of the form $[\sigma(x)]_{l+s}$ for some $x \in Z_L(f)$, or even if a vector $(z_1, \ldots, z_n) \in Z$ corresponds to a root x of $f(T)$. Thus one has to call Mink2coeff d^n times in the worst case. Even if $f(T)$ splits in L, this leads to a search of d vectors in a set of size d^n. In order to improve slightly this search, a simple observation can be made. Let us write $Z = \prod_\sigma Z_{\mathbb{C}}(f^\sigma)$ where σ ranges over $\mathrm{Hom}_K(L, \mathbb{C})$. Then one has

$$\forall x \in Z_K(f), \forall \sigma \in \mathrm{Hom}_K(L, \mathbb{C}), \exists! z \in Z_{\mathbb{C}}(f^\sigma) \mid z = [\sigma(x)]_l, \tag{4.1}$$

which implies that once we found a correct vector in Z we can remove from the search tree all the nodes where any of its coordinates appears. A more precise study of the average cost of the search (number of vectors of Z tested) can be found in Appendix A.

Notation 4.3. We denote by UpdateTree the procedure updating the search space as described.

The absolute method AbsoluteRoots requires at most $d = \deg f(T)$ decodings, while RelativeRoots_naive requires enumerating through d^n vectors, corresponding to n decodings each. The mere enumeration of d^n elements shows that RelativeRoots_naive has an exponential cost when n increases. In addition, several operations on vectors and matrices are done for each of the d^n possibilities. Thus it is quickly impractical.

Algorithm 8 RelativeRoots_naive

Require: An extension L/K, given by $\mathcal{B}$ a basis of K and $\mathcal{E}$ a K-basis of L, and $f(T) \in L[T]$ such that $Z_L(f) \in \mathbb{Z}[\mathcal{E} \otimes \mathcal{B}]$

Ensure: $Z_L(f)$

$l \leftarrow \mathsf{PrecisionEvaluation}(f(T))$
$L_l \leftarrow \mathrm{LLL}(B(\mathcal{B}, \tau, l))$
$M \leftarrow \Sigma_{L/K}^{-1}$ ▷ Up to precision $l + s$
$Z \leftarrow \prod_{\sigma \in \mathrm{Hom}_K(L,\mathbb{C})} \mathsf{FloatPolynomialRoots}(f^\sigma, l + s)$
$S \leftarrow \emptyset$
for $z \in Z$ **do**
    $y \leftarrow \mathsf{Mink2coeff}(z, M, L_l)$ ▷ Compute the inverse and decode
    if $f(y) = 0$ **then**
        $S \leftarrow S \cup \{y\}$
        $\mathsf{UpdateTree}(Z, z)$
    end if
end for
return S

However, for small n, Algorithm 8 can considerably speed-up the computation of $Z_K(f)$. Indeed one has to remember that an important part of the computation time is dedicated to the reduction of the lattice used to decode, as is also the case for the algebraic method. Algorithm 6 requires the reduction of a lattice of rank $[L : \mathbb{Q}]$, while the rank of the lattice reduced in Algorithm 8 is $[L : \mathbb{Q}]/n$. In addition, the precision needed to certify the computation shown in Theorem 3.17 involves the dimension. Therefore, dividing the dimension allows us to do computations at a smaller precision, which also leads to smaller coefficients in the matrix B_l which is reduced.

Theorem 4.4. *Consider an extension L/K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\mathcal{E}$ a K-basis of L and $f(T) \in L[T]$ such that $Z_L(f) \subset \mathbb{Z}[\mathcal{E} \otimes \mathcal{B}]$. Then for input $(L/K, \mathcal{E}, \mathcal{B}, f(T))$, Algorithm 8 outputs $Z_L(f)$ in polynomial time, except eventually for the enumeration due to the **for** loop.*

Proof. Since the required precision to ensure correctness of decoding has been proved to be polynomial with respect to entry sizes, the polynomial time complexity of each intermediate operations is clear. For the correctness, remark that for any $x \in Z_L(f)$ and any $\sigma \in \mathrm{Hom}_{\mathbb{Q}}(L, \mathbb{C})$, $\sigma(x)$ is a root of f^σ. Thus, $\sigma_{L/K}(x) \in \prod_{\sigma \in \mathrm{Hom}_K(L,\mathbb{C})} Z_{\mathbb{C}}(f^\sigma)$, so for each $x \in Z_L(f)$ there is one vector of Z in Algorithm 8 which is of the form $([\sigma(x)]_{l+s})_\sigma = [\sigma_{L/K}(x)]_{l+s}$. This ensures that S is the correct set at the end of Algorithm 8. □

Proposition 4.5. *Consider an extension L/K, $\mathcal{B}$ a $\mathbb{Q}$-basis of K, $\mathcal{E}$ a K-basis of L and $f(T) \in L[T]$ such that $Z_L(f) \subset \mathbb{Z}[\mathcal{E} \otimes \mathcal{B}]$. Then for input $(L/K, \mathcal{E}, \mathcal{B}, f(T))$, the algebraic complexity of lattice operations of Algorithm 8 (reduction of a basis lattice $\mathcal{L}_l$, GSO computation and decoding) are respectively in $O(n_K^4 l + n_K^3 l^2)$, $O(n_K^3)$ and $n_{L/K} d^{n_{L/K}} O(n_K^2)$, where l is the required precision.*

Proof. The proof is similar to the one of Proposition 3.20, noting that the lattice used to decode is linked to K, that there are $d^{n_{L/K}}$ elements in Z and that Mink2coeff makes $n_{L/K}$ calls to TestDecode for each of these elements. □

4.3. **Trade-off.** Now let us express some broad trade-off between our absolute and relative methods, considering only computations linked to lattice reduction and decoding. We forget about the GSO computation since it is done only once and is negligeable with respect to the lattice reduction. Consider L/K a number field extension and write n_L, n_K and $n_{L/K}$ for the degrees $[L:\mathbb{Q}]$, $[K:\mathbb{Q}]$ and $[L:K]$ respectively. Following Propositions 3.20 and 4.5 we can conclude that the complexity of Algorithm 6 due to lattice reduction and decoding is $O(n_L^4 l + n_L^3 l + d n_L^2)$ while these operations amount to $O(n_K^4 l + n_K^3 l + n_{L/K} d^{n_{L/K}} n_K^2)$ for Algorithm 8. Thus, we obtain a condition for a trade-off :

$$\begin{aligned} & n_K^2(n_K^2 l + n_K l + n_{L/K} d^{n_{L/K}}) = n_L^2(n_L^2 l + n_L l + d) \\ \iff\ & n_{L/K} d(d^{n_{L/K}-1} - n_{L/K}) = n_K l(n_{L/K}^3(n_L+1) - (n_K+1)). \end{aligned} \tag{4.2}$$

Using Equation (5.6) allows us to obtain a broad indication on which algorithm to run. For example, fixing $[L:K] = n_{L/K} = 2$ we obtain $2d(d-2) \leqslant n_K l(15n_K+7)$, therefore our relative algorithm would be advantageous for d up to (at least) $\sqrt{\frac{n_K l(15n_K+7)}{2}}$. Note that following both theoretical and heuristic bounds from Theorems 3.11 and 3.17, l is larger than $n_L^2 + n_L \log_2 \|x\|_2$ so $\sqrt{\frac{n_K l(15n_K+7)}{2}}$ can be replaced by

$$\sqrt{\frac{n_K(2n_K(2n_K + \log_2 \|x\|_2))(15n_K+7)}{2}} = n_K \sqrt{(2n_K + \log_2 \|x\|_2)(15n_K+7)}.$$

5. Improvements and heuristic observations

In order to improve the naive algorithms presented in Sections 3 and 4, several directions can be explored. The main bottleneck of our absolute method (Alg. 6) is the reduction of a basis lattice $\mathcal{L}(\mathcal{B}, \sigma, l)$. It can be hasten by improving the reduction process itself or by evaluating the required precision l more accurately. Our relative method potentially requires many decodings, i.e. calls to a BDD solver such as NearestPlane (Algorithm 4). Therefore, improving this step can speed-up considerably the running time of Algorithm 8.

5.1. **Basis reduction in the case of power-bases.** The first heuristic improvement that we develop concern the reduction of $B(\mathcal{B}, \sigma, l)$, in the case where $\mathcal{B}$ is a power-basis, i.e there is $\alpha \in K$ such that $\mathcal{B} = (1, \alpha, \ldots, \alpha^{n-1})$. The idea behind the improvement comes from the following observation. First, denote by $B^{(i)}$ the top left $i \times (i+1)$ submatrix of B, then assume that B is reduced up to the i-th vector, i.e. we found $U^{(i)}$ such that $L^{(i)} = U^{(i)} B^{(i)}$ is reduced. Then we found $u^{(i)} \in \mathbb{Z}^i$ such that $u^{(i)} B^{(i)} = L^{(i)}[i]$. Consider now the $(i+1)$-th vector of the matrix. Then

$[0, u^{(i)}] B^{(i+1)} \approx \alpha u^{(i)} B^{(i)} = \alpha L^{(i)}[i]$. Thus applying the transformation operated to reduce the i-th vector should pre-reduce the $(i+1)$-th vector. This leads to Algorithm 9, which computes a LLL-reduced basis L_l. Note that pre-reduction strategies have already been mentioned or used in the literature [39, 21]. To the best of our knowledge these apply to general knapsack-like matrices, while our observation tries to take advantage of the regularity of the basis vectors, and can be compared to [34].

We compared SpecLLL with LLL for available implementations of Magma [11], Pari/Gp [31] and Fplll [40]. The results can be found in Figure 2 for the real case and Figure 3 for the complex one.

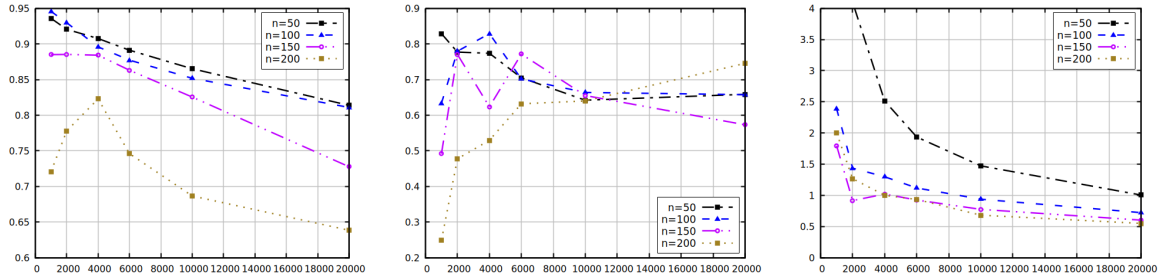
The first observation is that SpecLLL is asymptotically faster than LLL, regardless of the software used. Then one can remark that the ratio is decreasing when both the precision or the dimension is increasing.

SpecLLL is always more efficient than LLL when used in Pari/Gp or Magma, with a asymptotic gain of 25% and 35% respectively. For Fplll, using Algorithm 9 results in larger running times for small

Algorithm 9 SpecLLL

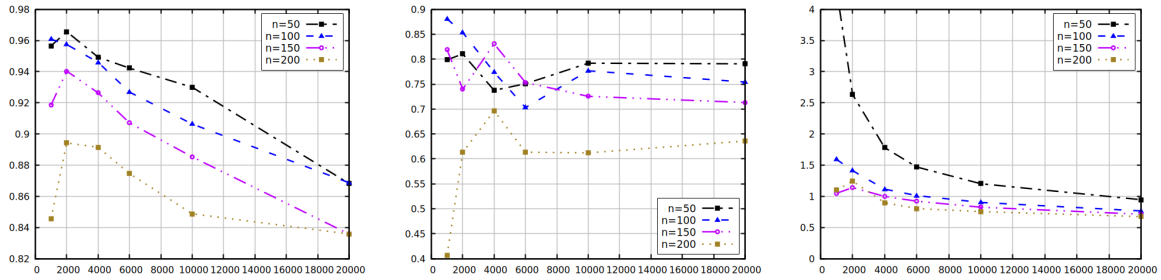
Require: $\mathcal{B}$, a basis of a lattice $\mathcal{L}$ of rank n.
Ensure: $\mathcal{B}'$, a LLL reduced basis of $\mathcal{L}$.

$\mathcal{B}' \leftarrow \{b_1\}$
for $i = 2$ to n **do**
  $b'_i \leftarrow b_i + \left\|b'_{i-1}\right\|_2^2 e_{i+1}$ ▷ M is a large constant to ensure reduction
  $b'_i \leftarrow b'_i - [0, u^{(i-1)}]B^{(i)}$ ▷ Pre-reduction using the previous vector
  $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{b'_i\}$
  $\mathcal{B}' \leftarrow \mathrm{LLL}(\mathcal{B}')$ ▷ Size-reduction
  $b'_i \leftarrow b'_i + (C - \|b'_{i-1}\|_2^2)e_{i+1}$
end for
$\mathcal{B}' \leftarrow \mathrm{LLL}(\mathcal{B}')$
return $\mathcal{B}'$



(A) Pari/Gp (B) Magma (C) Fplll

FIGURE 2. Ratios of running times of SpecLLL to LLL plotted against the precsion l, for $\sigma(K) \subset \mathbb{R}$ and several dimensions n



(A) Pari/Gp (B) Magma (C) Fplll

FIGURE 3. Ratios of running times of SpecLLL to LLL plotted against the precsion l, for $\sigma(K) \not\subset \mathbb{R}$ and several dimensions n

precision. However, it becomes more efficient as precision increases and offers speed-ups between 25 and 50% for $l = 20000$, depending on the dimension.

Moreover, as expected the speed-up given by SpecLLL is larger when $\sigma(K) \subset \mathbb{R}$, however one still gains up to 20% when $\sigma(K) \not\subset \mathbb{R}$.

Note that the gain really depends on the version of LLL used. However, a better gain does not imply that the corresponding software should be used. Indeed, one should also take into account the initial

running time of LLL. For example, FPLLL is more efficient than the two other implementation. We refer the reader to the timings plots presented in Appendix B.1, where are shown the corresponding timings.

Finally we verified that our modified algorithm really takes advantage of the special shape of the initial matrix. To do so, we launched SpecLLL over matrices of the shape

$$\begin{bmatrix} \beta_1 & C & 0 & \dots & 0 \\ \beta_2 & 0 & C & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \beta_n & 0 & \dots & 0 & C \end{bmatrix},$$

where β_i are random integers with the same size as the entries of $B(\mathcal{B}, \sigma, l)$, and computed the same timings ratios. Results can be found in Table 6, and one can remark that it is always close to 1, sometimes larger.

5.2. **Heuristic precision evaluation.** The second observation concern the precision needed to retrieve the roots. First we study which precision is required to retrieve an element depending on its norm, then we present an improvement concerning the evaluation of the norm of the roots.

5.2.1. *Precision from the norm.* The precision given by Equation (3.10) allows us to certify the correctness and the polynomial complexity of the method we described. However, it is possible to find a smaller value which is experimentally sufficient to retrieve an element.

Fix K a number field given by an irreducible polynomial $P_K(X) \in \mathbb{Z}[X]$. Given $\mathcal{B} = (b_1, \dots, b_n)$ a $\mathbb{Q}$-basis of K, we generated random elements $x \in \mathbb{Z}[\mathcal{B}]$ such that for all $i \in [\![1, n]\!], x_i \in [\![-2^s, 2^s]\!]$ for $s \in \{1, 25, 50, 75, 100\}$. Then we computed the quotient q_t of the precision required to retrieve the coefficients of x by $l_t = [K : \mathbb{Q}] \log_2 \|x\|_2$.

We chose to test the experimental precision against l_t for several reasons. First, when K and $\mathcal{B}$ are fixed, it is the term of Equation 3.10 which is asymptotically relevant. Moreover the algebraic method of [3] requires the norm $\mathrm{N}_{K/\mathbb{Q}}(\mathfrak{p}^k)$ to be greater than a value which is essentially l_t, so that the decoding is certified. Finally, this value was also suggested by experiments when we first used this method to compute cube roots in multicubic fields [25]. When comparing this value to Equation (3.10), we can remark that we gain essentially $n(n - \log_2(n))$.

Experimentally, the quotient q_t seems to be influenced by the size of the elements to decode and the dimension. More precisely, the maximum of the retrieved quotients in our experiments approaches 1 from above when the precision is increasing. Moreover, it seems to increase with the dimension. With these requirements in mind, we observed that it is always smaller than $1 + \ln(n) \ln\ln(n)/(s \ln(2) + \ln(n)/2)$ – see Figure 16 in appendix. Remark that the quotient in the previous value is the norm of an element with all coefficients equal to 2^s.

If Prec is the precision required to retrieve the coefficients of an element x, it seems reasonable that

$$\mathsf{Prec}(x) \leqslant \left(1 + \frac{\ln(n) \ln\ln(n)}{s + \ln(n)/2}\right) n \ln \|x\|_2 \leqslant n \ln \|x\|_2 + n \ln(n) \ln\ln(n). \tag{5.1}$$

5.2.2. *Norm of the roots.* Fix the factorisation of $f(T)$ over K as $f(T) = g(T)h(T)$ such $Z_K(g) = Z_K(f)$, $Z_K(h) = \emptyset$. We can follow [17, 3] to bound $\|x\|_2^2$ for $x \in Z_K(f)$ given $f(T)$:

(1) find B_{f,T_2} such $\left\|\sigma_{K/\mathbb{Q}}(x)\right\|_2^2 \leqslant B_{f,T_2}$ for all $x \in Z_K(f)$;
(2) compute $B_{\sigma_K^{-1}}$ the matrix norm of σ_K^{-1}, expressed relatively to the canonical basis of $\sigma_{K/\mathbb{Q}}(K)$ and $\mathcal{B}$;

(3) the value $B_f = B_{f,T_2} B^2_{\sigma_K^{-1}}$ satisfies the desired property.

This bound can be quite large compared to $M(f) := \sup\{\|x\|_2^2 \mid x \in Z_K(f)\}$. Without extra-information this is the best one can do. However, one can use heuristic evaluations for $\|x\|_2^2$, giving smaller results than B_f. Consider $\sigma \in \mathrm{Hom}(K, \mathbb{C})$ and $x \in K$. Then one has

$$|\sigma(x)|^2 = \left|\sum_{i=1}^{n} x_i \sigma(b_i)\right|^2 \leqslant \left(\sum_{i=1}^{n} |x_i \sigma(b_i)|\right)^2 \leqslant \|x\|_2^2 \, \|\sigma(\mathcal{B})\|_2^2 .$$

Therefore for any $x \in K$ and any $\sigma \in \mathrm{Hom}(K, \mathbb{C})$, $\|x\|_2^2 \geqslant \frac{|\sigma(x)|^2}{\|\sigma(\mathcal{B})\|_2^2}$. We define the function $\mathsf{HeuristicNorm}$ as the maximum of such quotients, as follows. For any $\sigma \in \mathrm{Hom}(K, \mathbb{C})$, let $B_{f,\sigma}$ be such that

$$\forall x \in Z_K(f), |\sigma(x)|^2 \leqslant B_{f,\sigma}. \tag{5.2}$$

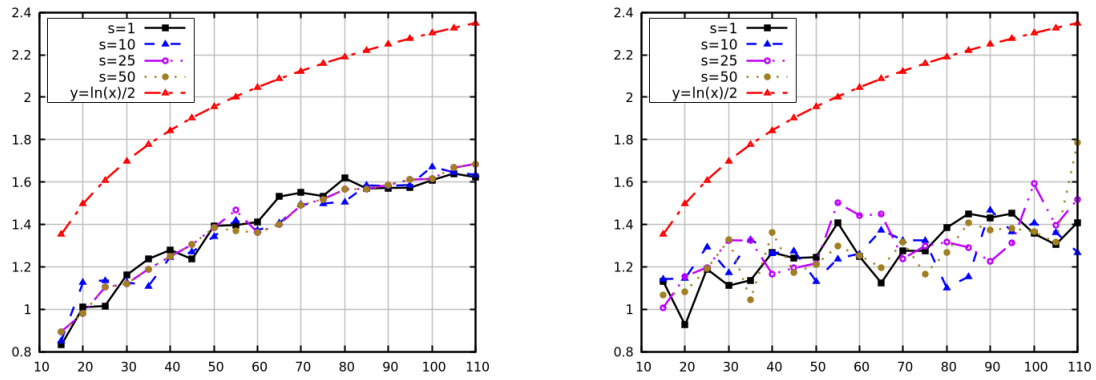
Then we fix

$$\mathsf{HeuristicNorm}(f) := \max\left\{ \frac{B_{f,\sigma}}{\|\sigma(\mathcal{B})\|_2^2} \mid \sigma \in \mathrm{Hom}(K, \mathbb{C}) \right\}. \tag{5.3}$$

Even if $\mathsf{HeuristicNorm}$ gives a value which is a lower bound on $\|x\|_2^2$ instead of an upper bound, it allows a first precision to be obtained. This evaluation is usually smaller than B_f and gives a good starting point, i.e. we do not need to increase much the precision to retrieve at least *some roots*.

We evaluated the generic difference between the proper maximal norm of $x \in Z_K(f)$ and the value given by $\mathsf{HeuristicNorm}$, and plotted the results in Figure 4. These expiremental values tend to show

$$\ln M(f) \leqslant \mathsf{HeuristicNorm}(f) + \ln(n)/2. \tag{5.4}$$



(A) $s(P_K) = 1$ (B) $s(P_K) = 5$

FIGURE 4. Logarithms of maximum ratios of $M(f)$ by $\mathsf{HeuristicNorm}(f)$ plotted against $n = [K; \mathbb{Q}]$ for several s, and $s(P_K) \in \{1, 5\}$

Heuristic precision. Finally, given K and $f(T) \in K[T]$ as above, one can compute a conjectured precision $\mathsf{HeuristicPrecision}(f)$ combining Equations (5.1) and (5.4) as

$$\mathsf{HeuristicPrecision} := [K : \mathbb{Q}] \log_2 \mathsf{HeuristicNorm}(f) + n \ln(n)/2 + n \ln(n) \ln\ln(n). \tag{5.5}$$

Algorithm 10 Babaï's Nearest Plane Algorithm with early abort– EarlyAbortNP

Require: $t \in \mathbb{R}^n$, $\mathcal{B} = (b_1, \ldots, b_r)$ a basis of a lattice $\mathcal{L}$, $\tilde{\mathcal{B}}$ the GSO of $\mathcal{B}$
Ensure: $v \in \mathcal{L}$ a vector close to t and b a boolean expressing whether t has been rejected or accepted.

1: $w \leftarrow t$
2: **for** $i = r$ down to 1 **do**
3: $\quad \lambda = (w \mid \tilde{b}_i) / \left\|\tilde{b}_i\right\|_2^2$
4: $\quad$ **if** $|\{\lambda\}| > 1/4$ **then**
5: $\quad\quad$ **return** w, false
6: $\quad$ **else**
7: $\quad\quad w \leftarrow w - \lfloor \lambda \rceil \, b_i$
8: $\quad$ **end if**
9: **end for**
10: **return** w, true

5.3. **Solution testing through finite fields.** An important step – especially when using heuristic observations – is to verify that a candidate is indeed a solution (steps 8 of both Algorithm 6 and Algorithm 8). When manupilating large elements, evaluating the polynomial expression $f(x)$ can be costly. Thus we chose to use a probabilistic method, which is summed up as follows:

- choose prime integers $p_1, \ldots, p_r$
- check whether $f(x) = 0 \bmod p_i$ for all $i \in [\![1, r]\!]$

If the p_i are uniformly drawn as 32-bits integers and $f(x)$ is uniformly distributed in $\mathbb{F}_{p_i}$, then the probability x of obtaining a false positive is $1/2^{32r}$, which is rapidly small. This choice is particularly useful for our relative method, since it potentially requires to test a huge amount of elements.

5.4. **Early abort in decoding.** The final observation that we use to speed-up computations consists in improving the decoding step, especially when the tested complex number y does not correspond to any solution. Here is the rationale behind our technique. If the vector Y given to NearestPlane correspond to a solution, then it is expected to be close to a vector of the lattice $\mathcal{L}_l$. However, if the vector is not a solution vector, then it should be "far" from a lattice vector. It can even be expected to be uniformly distributed modulo $\mathcal{L}_l$. Given a vector t and $\mathcal{B}$ a basis, if we denote $|(t \mid \tilde{b}_i) / \left\|\tilde{b}_i\right\|_2^2|$ by λ_i, then the distance between v and $\mathsf{NearestPlane}(t, \mathcal{B})$ is $\sum_{i=1}^{r} |\{\lambda_i\}| \cdot \left\|\tilde{b}_i\right\|^2$. The quantity $\{\lambda_i\} \in [-1/2, 1/2]$, thus we distinguish between two cases : if $|\{\lambda_i\}| \leqslant 1/4$ we deem λ_i to be acceptable and we continue the algorithm, while if $|\{\lambda_i\}| > 1/4$ we deem t to be rejected. This gives Algorithm 10.

Additionally to this strategy, we will reject a vector of complex numbers during the search of Algorithm 8 as soon as one of the decoding fails.

We checked the experimental maximum number of steps done in Algorithm 10 over bad vectors t of the form $[z, 0 \ldots, 0]$, when $z \in Z_{\mathbb{C}}(f)$ in the run of Section 5.5, for several root sizes $s_Z = \max_{x \in Z_K(f)} s(x)$, field dimesion $[K : \mathbb{Q}]$ and coefficient sizes of the defining polynomial $s(P_K)$. For fields dimension between 25 and 125 and various root sizes, the maximum that we find is 7. Over all parameters this maximum does not seem to vary much and is mainly in $[\![4, 7]\!]$, see Table 1 for example.

Algorithm 11 AbsoluteRootsHeur

Require: A number field K, $f(T) \in K[T]$, $\mathcal{B}$ a $\mathbb{Q}$-basis of K such that $Z_K(f) \in \mathbb{Z}[\mathcal{B}]$.
Ensure: The set $Z_K(f)$

1: $\sigma \leftarrow \mathsf{ChooseEmbedding}(K)$
2: $l \leftarrow \mathsf{HeuristicPrecision}(f(T))$ ▷ Heuristic norm evaluation and precision formula
3: $L_l \leftarrow \mathrm{LLL}(B(\mathcal{B}, \sigma, l))$ ▷ Use of SpecLLL when $\mathcal{B}$ is a power-basis
4: $Z \leftarrow \mathsf{FloatPolynomialRoots}(f^\sigma, l)$
5: $S \leftarrow \emptyset$
6: **for** $z \in Z$ **do**
7: $\quad y, b \leftarrow \mathsf{TestDecodeHeur}(L_l, z)$ ▷ Early abort nearest plane
8: $\quad$ **if** b **and** $f(y) = 0$ **then** ▷ Verification through finite fields
9: $\quad\quad S \leftarrow S \cup \{y\}$
10: $\quad$ **end if**
11: **end for**
12: **return** S

Dimension		25	50	75	100	125
$s_Z = 1$	min	1	1	1	1	1
	max	6	7	5	6	6
	average	2.7	2.4	2.5	2.2	2.8
$s_Z = 50$	min	1	2	1	1	1
	max	4	7	5	4	6
	average	2.5	2.8	2.7	2.3	2.8

TABLE 1. Number of steps before early aborts, for random targets in random number fields with $s(P_K) = 10$.

One can remark that the average value is always smaller than 3.

5.5. **Heuristic algorithms.** All the observations and strategies mentioned above lead to heuristic versions of the algorithms computing polynomial roots. First is the heuristic version of the decoding method TestDecode using EarlyAbortNP. We will design it by TestDecodeHeur. Using this version of the decoding method, together with the heuristic precision evaluation given by Equation (5.5) leads to a heuristic version of Algorithm 6 described in Algorithm 11.

Finally, regarding the relative method, we get a heuristic variant of Algorithm 7 and Algorithm 8. Remark that in Algorithm 12 (thus in Algorithm 13 as well) the use of early abort for the decoding of each coordinate respectively to L/K (step 8 of Algorithm 12) leads generally to decode only one of them when the vector of complex numbers X is not of the form $\sigma_{L/K}(x)$ (as a matter of fact, the decoding will typically abort only after a few steps).

5.5.1. *Trade-off.* Now let us express some broad trade-off between our heuristic absolute and relative methods, again considering only computations linked to lattice reduction and decoding. Compared to the previous analysis, we need to take into account the influence of early abort. This still gives $O(n_L^4 l + n_L^3 l + d n_L^2)$ in the worst-case for Algorithm 11 while these operations amount to $O(n_K^4 l + n_K^3 l +$

Algorithm 12 EarlyAbortMink2Coeff

Require: An extension L/K, given by $\mathcal{B}$ a basis of K and $\mathcal{E}$ a K-basis of L, an integer l, the matrix L_l from a reduced basis of $\mathcal{L}(\mathcal{B}, \tau, l)$, $M = \Sigma_{L/K}^{-1}$ up to a precision $l+s$, and $X = [\sigma_{L/K}(x)]_{l+s}$ for some $x \in L/K$

Ensure: A candidate $y = (y_1, \ldots, y_N)$ for the vector of coefficients of x expressed in $\mathcal{B} \otimes \mathcal{E}$

$Y \leftarrow XM$
$y = [\ \ ]$ ▷ void vector
for $i = 1$ **to** n **do**
  $v, b \leftarrow \mathsf{TestDecodeHeur}(L_l, Y_i)$
  if $b =$ false **then**
    return X, b
  end if
  $y \leftarrow [y \mid v]$
end for
return y, b

Algorithm 13 RelativeRootsHeur

Require: An extension L/K, given by $\mathcal{B}$ a basis of K and $\mathcal{E}$ a K-basis of L, and $f(T) \in L[X]$ such that $Z_L(f) \in \mathbb{Z}[\mathcal{E} \otimes \mathcal{B}]$

Ensure: $Z_L(f)$

$l \leftarrow \mathsf{HeuristicPrecision}(f(T))$
$L_l \leftarrow \mathrm{LLL}(B(\mathcal{B}, \tau, l))$ ▷ Use of SpecLLL when $\mathcal{B}$ is a power-basis
$M \leftarrow \Sigma_{L/K}^{-1}$ ▷ Up to precision $l+s$
$Z \leftarrow \prod_{\sigma \in \mathrm{Hom}_K(L,\mathbb{C})} \mathsf{FloatPolynomialRoots}(f^\sigma, l+s)$
$S \leftarrow \emptyset$
for $z \in Z$ **do**
  $y, b \leftarrow \mathsf{EarlyAbortMink2coeff}(z, M, L_l)$ ▷ Early abort
  if b **and** $f(y) = 0$ **then** ▷ Verification through finite fields
    $S \leftarrow S \cup \{y\}$
    $\mathsf{UpdateTree}(Z, z)$
  end if
end for
return S

$(n_{L/K} d + d^{n_{L/K}} - d) n_K^2)$ for Algorithm 13. This gives us the following condition for a trade-off :

$$d^{n_{L/K}} + d n_{L/K} n_K^2 - d n_L^2 - d = n_K^3 l (n_{L/K}^3 (n_L + 1) - (n_K + 1)). \tag{5.6}$$

When $n_{L/K} = 2$ for example, this becomes $d^2 - 2dn_K^2 - d = n_K^3 l(15 n_K + 7)$. Thus following the analysis for the naive versions of our algorithm we obtain that the relative version with early abort is advantageous compared to the absolute one for d up to

$$\sqrt{2} n_K^2 \sqrt{(n_K + 2 \log_2 \|x\|_2)(15 n_K + 7)}$$

which is better than the naive trade-off by a factor of $O(n_K)$.

6. Experimental results

We compare in these section the practical performances of the methods described previously with the generic algebraic methods implemented in Pari/Gp [31], which is the function nfroots. Our implementation is (mainly) in Gp, and is publicly available.[2]

Recall that we assume that $Z_K(f) \subset \mathbb{Z}[\mathcal{B}]$, with $\mathcal{B} = (b_1, \dots, b_n)$ being some $\mathbb{Q}$-basis of K. Given $x \in K$ we will keep denoting by $x_1, \dots, x_n$ its coefficients relative to $\mathcal{B}$. In almost all our experiments, we chose $\mathcal{B}$ to be $\mathbb{Z}[\theta]$ with $\theta = X \bmod P_K(X)$, where $P_K(X)$ is a fixed polynomial defining the field K.

Number fields are drawn randomly through the choice of their defining polynomial following a given distribution. When we consider general number fields, meaning not from a special family such as cyclotomic number fields, $P_K(X)$ will be drawn with random coefficients in a given interval of the form $[\![-2^s, 2^s]\!]$ with $s \in \mathbb{N}$. This somewhat arbitrary choice has been used in the literature [4] and allows us to study the average behaviour of our algorithms, for a well defined notion of "average". Finally, we consider two specific classes of number fields, namely *cyclotomic fields* in Section 6.3.3 and *real Kummer fields* in Section 6.3.4, because they showcase nicely the good and bad behaviours of our algorithms (especially our relative method) and Pari/Gp nfroots.

Remark 6.1. We checked the number of roots retrieved by our algorithms, and all roots were retrieved by both methods (absolute and relative).

6.1. Instability of nfroots.

We observed experimentally that over generic number fields, nfroots is not stable, and that fields could be divided into two groups. We call *bad fields* those number fields for which nfroots behave poorly, and *good fields* the rest of them. In order to obtain better observations, we split them by minimising the sum of their medians, that we denote by m_1 and m_2 respectively. To showcase this unstable behaviour we give some statistical data regarding the running time of nfroots, AbsoluteRoots and AbsoluteRoots in Table 2.

[2] https://github.com/AndLesav/nf_polynomial_roots

TABLE 2. Statistical data regarding the running time of nfroots, AbsoluteRoots and AbsoluteRoots over random number fields such that $s(P_K) = 5$.

(A) nfroots

Dimension $[K : \mathbb{Q}]$	25	50	75	100
# bad fields (%)	49	13	10	12
m_1 (s)	2.37	20.41	55.51	125.92
m_2 (s)	3.10	174.00	2189.93	16081.07
average (s)	2.76	40.08	269.67	1989.31

(B) AbsoluteRoots

Dimension $[K : \mathbb{Q}]$	25	50	75	100
# bad fields (%)	65	77	68	67
m_1 (s)	1.59	39.53	318.24	1637.11
m_2 (s)	2.69	66.44	539.42	2646.72
average (s)	2.31	60.44	467.41	2285.76

(C) AbsoluteRootsHeur

Dimension $[K : \mathbb{Q}]$	25	50	75	100
# bad fields (%)	57	71	72	61
m_1 (s)	0.42	3.86	15.95	54.84
m_2 (s)	0.61	5.31	21.72	67.19
average (s)	0.53	4.88	19.98	62.33

In Table 2 one can see that there are about 10% of fields for which the running time of nfroots explodes. The corresponding median m_2 is up to 127 times larger than the median m_1 corresponding to the rest of the fields. Algorithm 6 and Algorithm 13 are more stable. For these algorithms m_2 never exceeds $2m_1$. The instability of nfroots has the following consequence. For a majority of number fields, nfroots clearly outperforms AbsoluteRoots (it is more than 10 times faster for $[K : \mathbb{Q}] = 100$) and this difference is growing with the dimension. However, *in average*, the timings obtained for both algorithms are similar.

Potential explanation. The determining parameter seems to be the possibility of retrieving an inert prime. Following the search for good primes implemented in PARI/GP, we evaluated the proportion of fields for which an inert prime will be found, see Table 3.

Dimension $[K : \mathbb{Q}]$	23	51	75	83
# bad fields (%)	5	7	17	10

TABLE 3. Proportion of fields for which no inert prime has been found

These proportions tend to follow the ones found previously, such has presented in Table 2. Additionnally we recorded the running times over the two kind of fields, as showcased in Table 3.

TABLE 4. Running time of nfroots, AbsoluteRoots and AbsoluteRoots over random number fields for which inert primes can be found or not.

(A) nfroots

Dimension $[K:\mathbb{Q}]$	25	50	75
Good fields (s)	3.1	17.8	55.5
Bad fields (s)	4.9	164.1	2177.9

(B) AbsoluteRoots

Dimension $[K:\mathbb{Q}]$	25	50	75
Good fields (s)	2.3	52.5	437.1
Bad fields (s)	2.3	52.2	403.1

(C) AbsoluteRootsHeur

Dimension $[K:\mathbb{Q}]$	25	50	75
Good fields (s)	0.5	4.7	21.7
Bad fields (s)	0.5	4.7	20.7

The results presented in Tables 2 and 3 are very similar, which confirms the fact that the determining parameter regarding the instability of nfroots is the use of inert primes.

Remark 6.2. In the following, for nfroots, we will systematically present the average value of the execution times over all fields, over "good" fields and over "bad" fields, in order to have a more complete picture of its behaviour. We will write m_1 and m_2 the two last quantities.

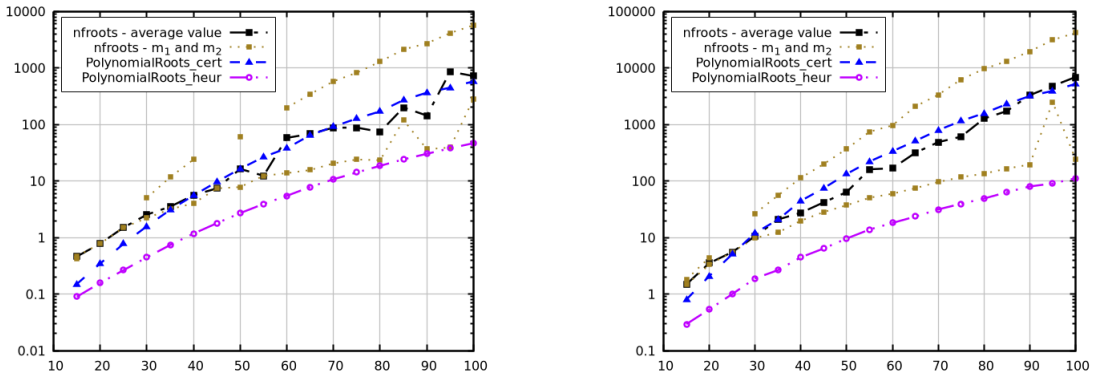
6.2. **Absolute method.** We study the impact of the different parameters of the problems which are the size of the polynomial $P_K(X)$, the number of roots $|Z_K(f)|$, and the size of the roots $\log_2 \|x\|_2$. We also differentiated between number fields K such that $r_1 > 0$ or such that $r_1 = 0$, since the matrices used to decode do not have exactly the same sizes. However, since the results are very similar we only present here the data for $r_1 > 0$. Data when $\sigma(K) \subset \mathbb{C} \setminus \mathbb{R}$ can be found Appendix B.3.

6.2.1. *Choice of $P_K(X)$.* We study the impact of two parameters linked to $P_K(X)$, namely $\deg P_K(X)$ which is the dimension of K, and the size of its coefficients. In our experiments, we fixed the parameters of the problem linked to $f(T) \in K[T]$. More precisely we considered $f(T)$ such that:

- $\deg f(T) = 50$;
- $f(T)$ splits in K, i.e. $|Z_K(f)| = \deg f(T)$;
- $\forall x \in Z_K(f), \log_2 \|x\|_\infty \leqslant 10$.

Then we considered $P_K(X) = p_0 + p_1 X + \cdots + p_{n-1}X^{n-1} + X^n \in \mathbb{Z}[X]$ for increasing degrees n, and several coefficient sizes $s(P_K) = \log_2 \|P_K\|_\infty$. More precisely for sizes $s(P_K) \in \{1, 10\}$, we picked polynomials $P_K(X)$ such that $\forall i \in [\![0, n-1]\!], p_i \in [\![-2^{s(P_K)}, 2^{s(P_K)}]\!]$, and this for n increasing. The data obtained are shown in Figures 5 and 18.

One can remark that the time efficiency of all three methods are widely influenced by the dimension $[K:\mathbb{Q}]$. It is easily explained by the fact that all three methods require the computation of a LLL-reduced basis of a lattice with rank equal to $[K:\mathbb{Q}]$. Moreover the volume of said lattice depends also on the



(A) $s(P_K) = 1$ (B) $s(P_K) = 10$

FIGURE 5. Average timings (s) of nfroots, AbsoluteRoots and AbsoluteRootsHeur plotted against $\deg P_K(X)$ for randomly generated $P_K(X)$ such that $r_1 > 0$, with $s(P_K) \in \{1, 10\}$

dimension. Remark that as observed in Table 2, the difference between m_1 and m_2 widens when $[K : \mathbb{Q}]$ increases.

The parameter $s(P_K)$, i.e. the coefficient size of the defining polynomial of K, also influences the performances of all three algorithms. Our heuristic method seems to be slightly less impacted by this parameter.

One can see from Figure 5 and Figure 18 that AbsoluteRoots is less efficient than nfroots, at least for the fixed shape of $f(T)$. The heuristic method AbsoluteRootsHeur – using heuristic norm evaluation and formula to compute the precision – is way more efficient than the certified version. It is also more efficient than nfroots *on average*, and compete with its best running times in a number of cases. Our method seems to be more influenced by the dimension of K than the algorithm of PARI/GP, at least when the latter runs at its best.

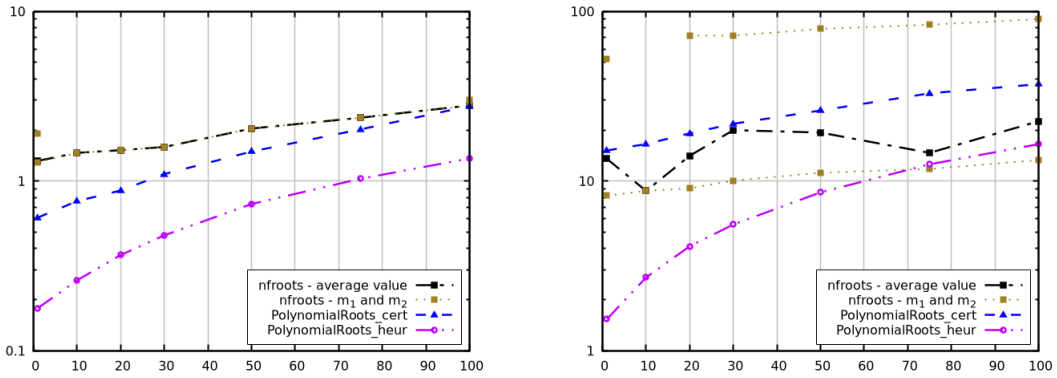
6.2.2. *Size of the roots.* We now study how the size of the elements of $Z_K(f)$ impacts the performance of the different algorithms. To this end we fixed the parameters linked to $P_K(X)$ and $f(T)$:

- $\deg f(T) = 50$;
- $f(T)$ splits in K, i.e. $|Z_K(f)| = \deg f(T)$;
- $s(P_K) = 1$;
- $\deg P_K(X) \in \{25, 50\}$.

We did experiments for increasing size of roots. Let us denote by s_Z this size, i.e. $\forall x \in Z_K(f), \log_2|x_i| \in [\![0, s_Z]\!]$. The results can be found in Figure 19.

One can see that our algorithm is more influenced by s_Z than nfroot. This is certainly due to the fact that the lattices reduced for decoding during the algorithm described in [3] are ideals, and more precisely *prime powers ideals*. These are usually easier to reduce, especially if one can use a pre-reduction such as mentioned by Belabas, or the documentation of PARI/GP [31]. Remark that the difference of running time for nfroots between bad and good fields seems constant here.

6.2.3. *Partial conclusion.* From the different situations explored and the data gathered, we can conclude that the certified version of our method AbsoluteRoots is in general less efficient than the algebraic method implemented in PARI/GP. However, their running time are similar *in average* in some situations (see



(A) $[K : \mathbb{Q}] = 25$ (B) $[K : \mathbb{Q}] = 50$

FIGURE 6. Average timings (s) of nfroots, AbsoluteRoots and AbsoluteRootsHeur plotted against s_Z for randomly generated $P_K(X)$ such that $[K : \mathbb{Q}] = 25$ and $[K : \mathbb{Q}] = 50$, with $r_1 > 0$

for instance Figure 5). The heuristic version AbsoluteRootsHeur (Alg. 11) offers a clear advantage and is more efficient than nfroot, still in average. When restricted to the "good" fields for nfroot, our heuristic version is still competitive for relatively small roots (crossing point at $s_Z \sim 75$ for $[K : \mathbb{Q}] = 50$).

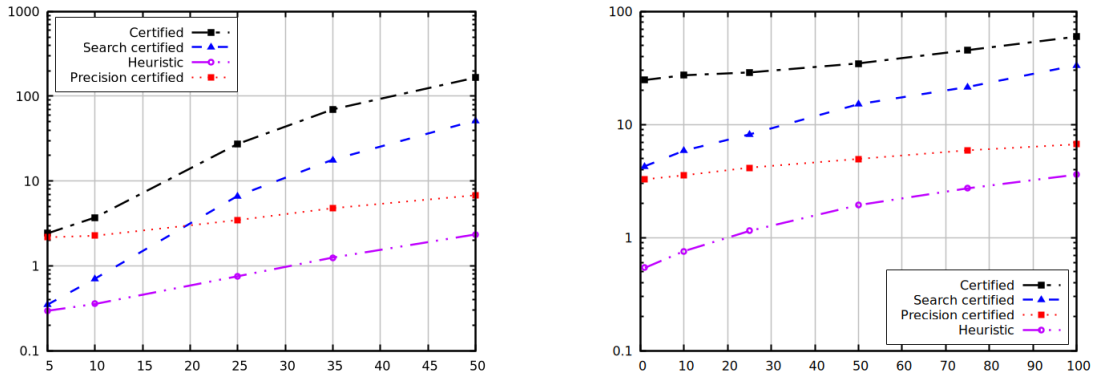
Regarding the impact of the different parameters, our method behaves less nicely with respect to the dimension and the size of the roots. This is certainly due to the nature of the lattices to be reduced in the two different methods. The algebraic method takes fully advantage of the nature of the lattice, which is a power of a prime ideal, thus mitigating the influence of the size of the vectors and the dimension. In the best cases, it uses a pre-reduction algorithm which hasten considerably the subsequent LLL reduction [3].

6.3. **Relative extensions.** Let us now consider relative extensions L/K, and study the efficiency of Algorithm 13. First we look into the impact of our heuristic strategies. Then we compare this method to Algorithm 11. Finally, we study both algorithms together with nfroots over cyclotomic fields and Kummer extensions.

Let us fix the notations. We will consider L/K together with $P_K(X) \in \mathbb{Q}[X]$ and $P_L(Y) \in K[Y]$ such that $K \cong \frac{\mathbb{Q}[X]}{(P_K(X))}$ and $L \cong \frac{K[Y]}{(P_L(Y))}$.

6.3.1. *Impact of the heuristic strategies.* We considered several versions of the relative method : the certified one as Algorithm 8, the heuristic one as Algorithm 13, and two intermediate versions, where either the precision or the search is heuristic. We studied the impact of (a) the degree of the equation $f(T)$, and (b) the size s_Z of the roots. We considered extensions L/K such that $[K : \mathbb{Q}] = 30$, $[L : K] = 3$, $s(P_K), s(P_L) \leqslant 1$, and $f(T)$ splits in K. Additionnally, we fixed $s_Z = 10$ when studying $\deg f(T)$ and $\deg f(T) = 25$ when studying the other parameter. We focused on small parameters for these experiments, as we wished to illustrate the impact of the heuristic modifications that we described. The timings obtained can be found in Figure 7.

One can remark that both heuristic observations speed-up the computations, especially when the degree of f is increasing, as shown in Figure 7a. This leads to a speed-ub up to a factor 100 for the full heuristic implementation compared to the "naive" one. Moreover from both experiments, one can



(A) $\deg f$ varying (B) s_Z varying

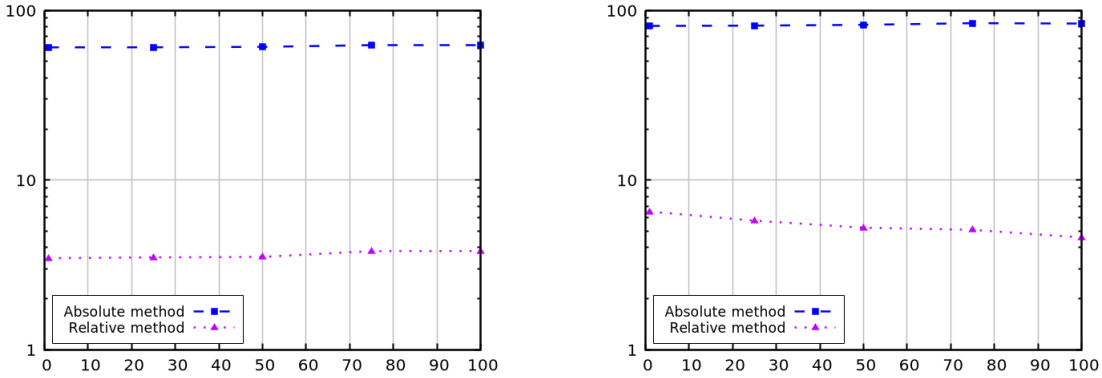
FIGURE 7. Timings (s) of certified and heuristic versions of RelativeRoots plotted against $\deg f(T)$ (resp. s_Z) for randomly generated $P_K(X), P_L(Y)$

conclude that the impact of the heuristic search (using Algorithm 12) is more important than the one of the heurictic precision – at least for the small range of parameters considered.

6.3.2. *Generic number fields.* In this section, we study the performances of our most efficient procedures, i.e. the heuristic algorithms AbsoluteRootsHeur and RelativeRoots, over "generic" number fields. Here, the term "generic" refers to fields whose defining polynomials have coefficients uniformly drawn in segments of the form $[\![-2^s, 2^s]\!]$. For practicability reasons, our experiments mainly deal with $s = 1$.

Number of roots. We first study the influence of the number of roots, as it impacts the running time of RelativeRootsHeur. We considered extensions L/K such that $[L : \mathbb{Q}] = 90$ and $[L : K] \in \{2, 3\}$ with:

- $s(P_K), s(P_L) \leqslant 1$;
- $\deg f(T) = 50$;
- $\forall x \in Z_K(f), s(x) \leqslant 10$.

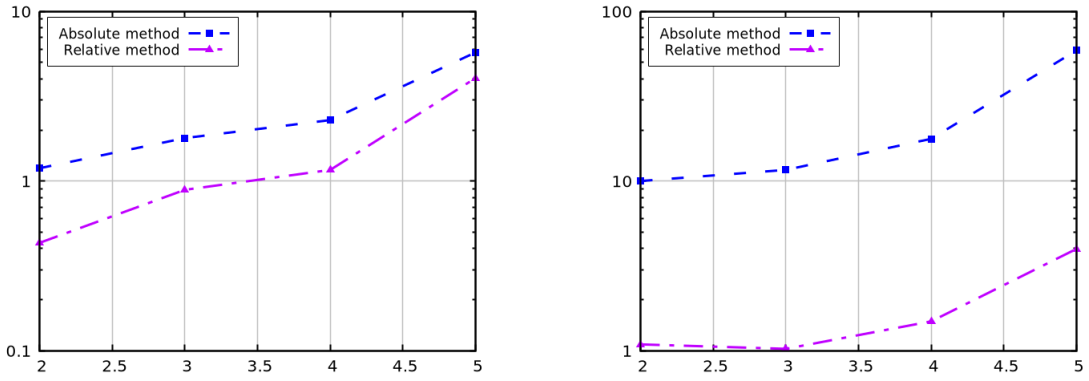


(A) $[L : K] = 2$ (B) $[L : K] = 3$

FIGURE 8. Average timings (s) of AbsoluteRoots and RelativeRoots plotted against q_f for randomly generated $P_K(X), P_L(X)$ and $f(T)$, such that $[L : \mathbb{Q}] = 90$ and $\deg f(T) = 50$.

The influence of the number of roots can be seen in Figure 8. The timings of the method AbsoluteRoots increase with the number of roots, while the ones for the relative algorithm RelativeRoots globally decrease, which is expected, see Appendix A. Indeed, with q_f increasing, roots are retrieved earlier and the search space is updated more quickly. One can remark on these first comparisons, that Algorithm 13 can be between 5 and 30 times faster than Algorithm 11. Moreover, as expected, Algorithm 13 is slower when $[L : K]$ is increasing. All these observations will be verified in other experiments.

Degree of the extension. Let us now consider the influence of the degree $[L : K]$ of the extension considered. We already know that the running time of RelativeRoots depends exponentially on this parameter. We illustrate its impact on practical computations.



(A) $[K : \mathbb{Q}] = 15$ (B) $[K : \mathbb{Q}] = 25$

FIGURE 9. Average timings (s) of AbsoluteRoots and RelativeRoots plotted against $[L : K]$ for randomly generated $P_K(X), P_L(Y)$ and $f(T)$, such that $[K : \mathbb{Q}] \in \{15, 25\}$ and $\deg f(T) = 50$.

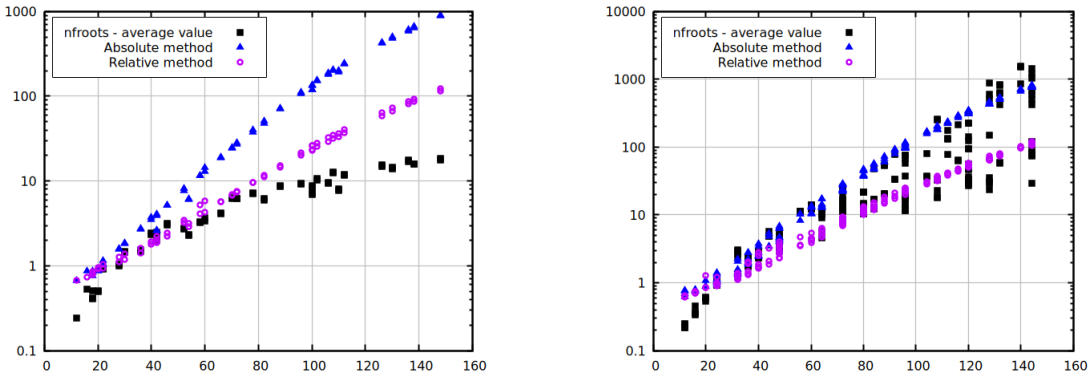
As expected, the running time of Algorithm 13 is deeply impacted by an increase in the relative degree $[L : K]$. However, when compared to Algorithm 11, one can observe that this drawback is mitigated when considering increasing degrees $[K : \mathbb{Q}]$. This shows that for a constant $[L : K]$ using our relative method should asymptotically outperform the absolute methods AbsoluteRootsHeur.

6.3.3. *Cyclotomic fields.* In this section we study the performances of AbsoluteRootsHeur, RelativeRootsHeur and nfroots over cyclotomic fields. We will write K_m the cyclotomic field $\mathbb{Q}(\zeta_m)$ with conductor m. We study these fields because they are widely used in applied mathematics such as cryptography¿ Moreover, since their generic defining polynomials have small norms, nfroots is particularly efficient.

Additionally our relative method always apply be used over these fields. Indeed, K_m always has a totally real subfield K_m^+ such that $[K_m : K_m^+] = 2$. Moreover, since $K_m^+ = K_m^\tau$ where τ is the complex conjugation, the search space has $\deg f$ elements. We use this structure only in our first experiments. Results are gathered in Figure 10.

If we compare the timings obtained to previous data such as presented in Figure 5 or Figure 18, both AbsoluteRootsHeur and nfroots seem more efficient than over randomly generated number fields. It is particularly the case of the latter.

The PARI/GP nfroots is more efficient than our absolute method AbsoluteRootsHeur over all fields. Our relative method seems to compete with the implementation of PARI/GP [31] with the set of parameters



(A) Good fields (B) Bad field

FIGURE 10. Average timings (s) of AbsoluteRootsHeur, RelativeRoots and nfroots plotted against $[K_m : \mathbb{Q}]$ for cyclotomic fields of conductor m such that $10 \leqslant m \leqslant 500$ and $[K_m : \mathbb{Q}] < 150$, for $s_Z = 50$.

fixed in the experiments over "bad" fields, see Figure 10b. It is again more influenced by the dimension than nfroots (in its best behaviour). Additionally since we know from Section 6.2 that the methods using complex embeddings have a worse behaviour with respect to the size of the roots, it seems that using only K_m^+ is not enough to beat the implementation of PARI/GP over cyclotomic fields.

Note however that most of these fields do have other subfields, which can be themselves cyclotomic fields. For example, if $m = p^r$ with p a prime integer, then writing $m' = p^{r'}$ such that $r' < r$ we have that $K_{m'}$ is a subfield of K_m such that $[K_m : K_{m'}] = p^{r-r'}$. Using the extension $K_m/K_{m'}$ would lead to a search space with $d^{p^{r-r'}}$ elements. We ran some experiments over such extensions for $p \in \{2, 3\}$, data are gathered in Table 5. Again we chose $s_Z = 50$ and $\deg f(T) = 50$ (resp. $\deg f(T) = 55$) for $p = 2$ (resp. $p = 3$). We retrieve similar timings and asymptotically, the choices $p = 2, r = 2$ and $p = 3, r = 1$ should outperform nfroots. Remark that as expected AbsoluteRoots is the least efficient algorithm.

6.3.4. *Kummer extensions.* Let us now focus on real Kummer extensions, meaning such that $L = \mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_r})$, where $p \geqslant 2$ is a prime integer and $(m_1, \ldots, m_r) \in \mathbb{Q}^r$ such that $[K : \mathbb{Q}] = p^r$. Fix $K = \mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_{r-1}})$. Then L/K is an extension over which one can take full advantage of RelativeRoots. Indeed K can be embedded in $\mathbb{R}$, so that $\mathrm{Hom}_K(L, \mathbb{C})$ has one real embedding and $p - 1$ complex ones. This gives a search space with $\deg f(T)^{(p-1)/2}$ elements. We will compare AbsoluteRoots, RelativeRoots and nfroots of PARI/GP [31]. We study these fields because real Kummer extensions are all "bad" fields for the p-adic method. Recall that is mentioned by [3] that ideal lattices are usually easy to reduce – especially when one can use a pre-reduction algorithm – which occurs when the inertial degree of said ideal is large. Moreover we saw in Section 6.1 that nfroots behaved badly when the search for an inert prime was unfruitful. It turns out that over real Kummer fields of the form $\mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_r})$, the inertial degree cannot be larger than the exponent p (see for instance [24]).

The first situation that we will explore will be the same as before, i.e. $f(T)$ is split. It will show how computations can be accelerated in good situations, and the difference between good and bad fields for nfroots. Then we will study the special case where $f(T)$ has degree p. It is the direct generalisation of $f(T) = T^p - \alpha^p$ with $\alpha \in L$, which is the type of equation that are to be solved in several tasks involving

TABLE 5. Running time of nfroots, AbsoluteRoots and AbsoluteRoots over cyclotomic fields of conductor $m = p^r$ with p a prime integer.

(A) $p = 2$

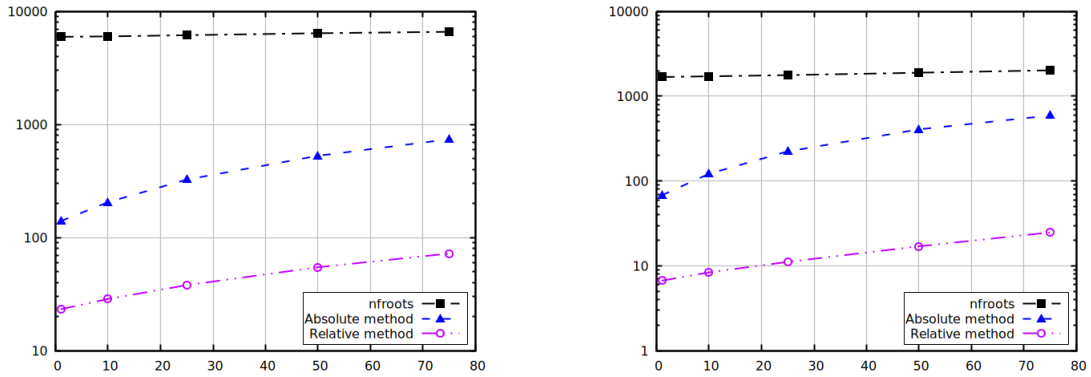
r	5	6	7	8
nfroots	0.4	2.4	11.6	30.3
AbsoluteRoots	0.7	1.4	13.0	436.8
RelativeRoots, $[L : K] = 2$	1.4	2.0	3.8	27.9
RelativeRoots, $[L : K] = 4$	19.6	22.4	29.2	38.0

(B) $p = 3$

r	3	4	5
nfroots	0.5	2.8	29.7
AbsoluteRoots	0.9	6.8	1334.0
RelativeRoots, $[L : K] = 3$	3.9	7.1	33.1

(S-)units or class group [8, 17, 35]. In particular, this task arose in several practical works these past few years [2, 25, 9, 7, 10].

Split equation. We considered split polynomials $f(T)$ of degree 50 for increasing size of roots s_Z, over Kummer fields of the form $L = \mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_r})$ with $p \in \{2, 5\}$. Moreover $[L : \mathbb{Q}] = 128$ if $p = 2$, $[L : \mathbb{Q}] = 125$ if $p = 5$, and each m_i is a prime number smaller than 40. The results are shown in Figure 11.



(A) $p = 2$ and $[L : \mathbb{Q}] = 128$ (B) $p = 5$ and $[L : \mathbb{Q}] = 125$

FIGURE 11. Average timings (s) of AbsoluteRootsHeur, RelativeRoots and nfroots plotted against s_Z for Kummer fields of exponent $p \in \{2, 5\}$.

One can observe that both Algorithm 11 and Algorithm 13 outperform nfroots. Our relative method is particularly efficient, and is up to 100 times faster. This contrasts with cyclotomic fields. Indeed, compare timings reported in Figure 10 and Figure 11a for which the degree of the relative extension is 2.

Small degree equations. As mentioned we consider here polynomials $f(T)$ with small degrees, namely $\deg f(T) = p$ over real Kummer fields $\mathbb{Q}(\sqrt[p]{m_1}, \ldots, \sqrt[p]{m_r})$ of exponent p. For such degrees, the search

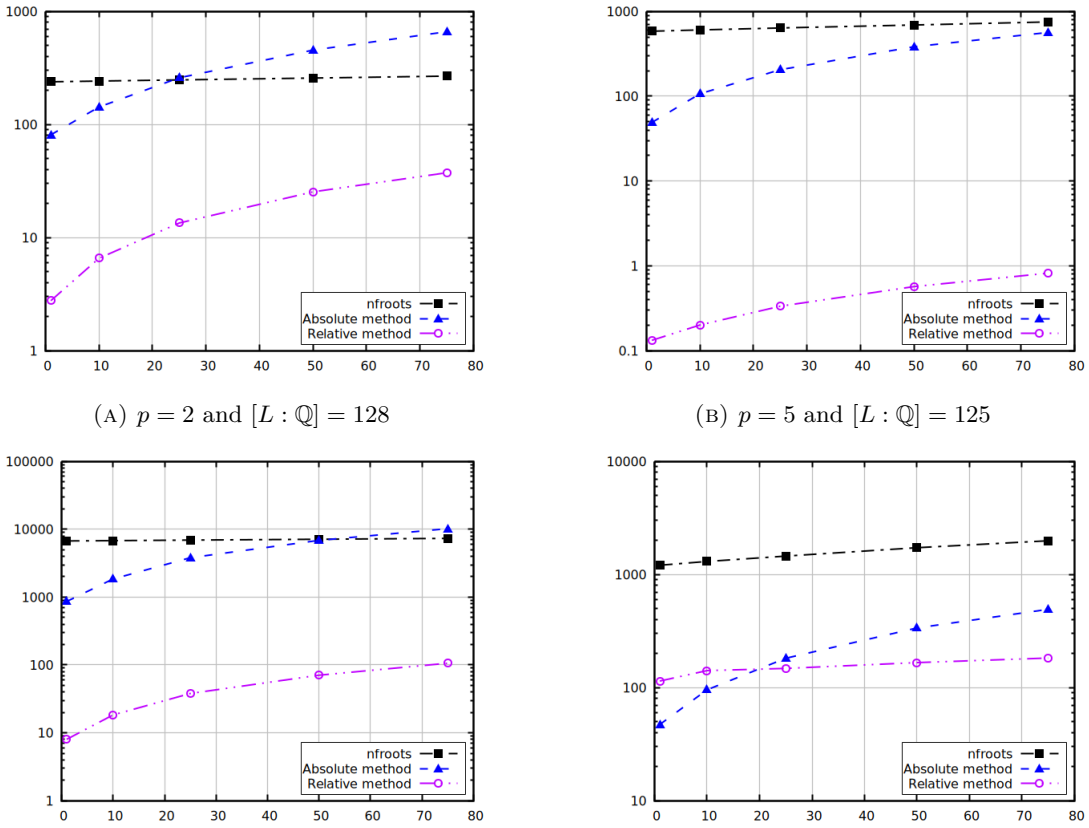
space of Algorithm 13 is small enough to take full advantage of this method. Indeed, since the complex embeddings in $\mathrm{Hom}_K(L, \mathbb{C})$ are all conjugates (except one real embedding), the cardinality of the search space is $p^{\frac{p+1}{2}}$.

Remark 6.3 (nfroots)**.** In this configuration, nfroots does not follow the method described before. Indeed, when $3 \deg f(T) < [L : \mathbb{Q}]$, the implementation of Pari/Gp uses Trager's method [42, 3] for factorising polynomials. We will therefore refer to it as Trager, to differentiate it from the p-adic method.

We considered Kummer fields of exponent p in $\{2, 3, 5, 11\}$ and the different objects are drawn such that:

- each m_i is a prime number smaller than 40;
- $\deg f(T) = p$ and $|Z_K(f)| = 1$.

The data gathered can be found in Figure 12.



(A) $p = 2$ and $[L : \mathbb{Q}] = 128$

(B) $p = 5$ and $[L : \mathbb{Q}] = 125$

(C) $p = 3$ and $[L : \mathbb{Q}] = 243$

(D) $p = 11$ and $[L : \mathbb{Q}] = 121$

FIGURE 12. Average timings (s) of AbsoluteRoots and RelativeRoots and Trager plotted against s_Z for randomly generated Kummer fields L/K with exponent $p \in \{2, 3, 5, 11\}$ and $\deg f(T) = p$.

In most cases our relative method is way more efficient than the two others. It can go up to 500 times faster for $p = 3$ for example. The method Trager implemented in Pari/Gp is always worse than both our algorithms when the roots are small, but it is stable when the size is increasing and it outperforms

Algorithm 11 for large roots and $p \in \{2, 3, 5\}$. Note however that a basis lattice $\mathcal{L}_l$ can be reduced once and used to solve different equations. Thus, a batch strategy is possible with AbsoluteRootsHeur, which is not the case with Trager.

If one compares the data gathered in Figures 12a and 12b for $p = 2$ and $p = 5$ respectively – for which the degrees $[L : \mathbb{Q}]$ and the size of the search space are similar – one can see that the timings for RelativeRootsHeur are around 10 times lower for $p = 5$. This is due to the fact that the dimension of the subfield K over which decodings are made is smaller in this case. These observations are confirmed with the timings gathered in Figure 12c.

Finally, one can remark in Figure 12d that the size of the search space is important, as the difference between RelativeRootsHeur and AbsoluteRootsHeur is less important for $p = 11$. However in this case as for the others, it seems that the size of the roots is less of a problem for the relative method than for AbsoluteRootsHeur.

6.3.5. *Small dimensions, small degree and large roots.* We consider a final situation to showcase the performances of the algorithms studied here, namely when $f(T)$ has large roots and $\deg f(T)$ is small as well as $[K : \mathbb{Q}]$. This situation is encountered in its extreme in the Number Field Sieve with $\deg f(T) = 2$ and $[K : \mathbb{Q}]$ usually smaller than 10. In our experiments, we considered $\deg f(T) = 10$ and $[K : \mathbb{Q}] \in \{12, 30\}$. Data are gathered in Figure 13.

As already observed, Pari/Gp nfroots is asymtotically more efficient than the absolute method AbsoluteRoots. However RelativeRoots offers again a certain advantage compared to nfroots, especially for larger dimensions, here $[L : \mathbb{Q}] = 30$. Note that Magma function Roots becomes quickly the least efficent method by a large factor.

References

[1] L. Babai. "On Lovasz lattice reduction and the nearest lattice point problem". In: *Combinatorica* 6 (Mar. 1986), pp. 1–13. DOI: 10.1007/BF02579403.

[2] J. Bauch et al. "Short Generators Without Quantum Computers: The Case of Multiquadratics". In: *Advances in Cryptology – EUROCRYPT 2017*. Ed. by J.-S. Coron and J. B. Nielsen. Cham: Springer International Publishing, 2017, pp. 27–59. ISBN: 978-3-319-56620-7.

[3] K. Belabas. "A relative van Hoeij algorithm over number fields". In: *J. Symb. Comput.* 37 (May 2004), pp. 641–668. DOI: 10.1016/j.jsc.2003.09.003.

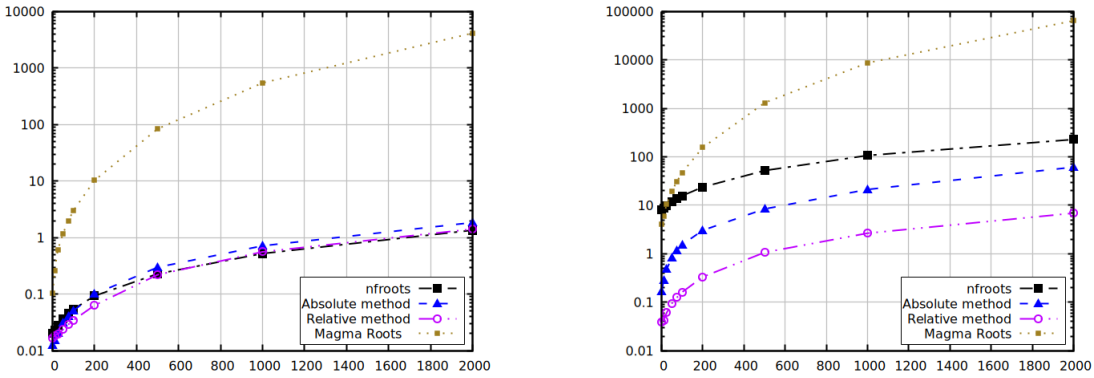
[4] K. Belabas. "Topics in computational algebraic number theory". en. In: *Journal de théorie des nombres de Bordeaux* 16.1 (2004), pp. 19–63. DOI: 10.5802/jtnb.433.

[5] K. Belabas et al. "Factoring polynomials over global fields". en. In: *Journal de Théorie des Nombres de Bordeaux* 21.1 (2009), pp. 15–39. DOI: 10.5802/jtnb.655.

[6] O. Bernard and A. Roux-Langlois. "Twisted-PHS: Using the Product Formula to Solve Approx-SVP in Ideal Lattices". In: *Advances in Cryptology – ASIACRYPT 2020*. Ed. by S. Moriai and H. Wang. Cham: Springer International Publishing, 2020, pp. 349–380. ISBN: 978-3-030-64834-3.

[7] O. Bernard et al. *Log-S-unit lattices using Explicit Stickelberger Generators to solve Approx Ideal-SVP*. Cryptology ePrint Archive, Report 2021/1384. https://ia.cr/2021/1384. 2021.

[8] J.-F. Biasse and C. Fieker. "Improved techniques for computing the ideal class group and a system of fundamental units in number fields." In: *Algorithmic Number Theory, 10th International Symposium, ANTS-IX, San Diego CA, USA, July 9-13, 2012. Proceedings*. Vol. 1. Open Book Series. Mathematical Science Publishers, 2012, pp. 113–133.



(A) $[L : K] = 2$ and $[K : \mathbb{Q}] = 6$

(B) $[L : K] = 2$ and $[K : \mathbb{Q}] = 15$

(C) $[L : K] = 3$ and $[K : \mathbb{Q}] = 4$

(D) $[L : K] = 3$ and $[K : \mathbb{Q}] = 10$

FIGURE 13. Average timings (s) of AbsoluteRoots and RelativeRoots, nfroots and Magma Roots plotted against s_Z for randomly extensions L/K with $[L : \mathbb{Q}] \in \{12, 30\}$ and $\deg f(T) = 10$.

[9] J.-F. Biasse and C. Vredendaal. "Fast multiquadratic S-unit computation and application to the calculation of class groups". In: *The Open Book Series* 2 (Jan. 2019), pp. 103–118. DOI: 10.2140/obs.2019.2.103.

[10] J.-F. Biasse et al. "Norm relations and computational problems in number fields". In: *Journal of the London Mathematical Society* 105.4 (2022), pp. 2373–2414. DOI: https://doi.org/10.1112/jlms.12563.

[11] W. Bosma, J. Cannon, and C. Playoust. "The Magma algebra system. I. The user language". In: *J. Symbolic Comput.* 24.3-4 (1997). Computational algebra and number theory (London, 1993), pp. 235–265. ISSN: 0747-7171. DOI: 10.1006/jsco.1996.0125.

[12] H. Cohen. *Advanced Topics in Computational Number Theory*. Graduate Texts in Mathematics. Springer New York, 2012. ISBN: 9781441984890.

[13] H. Cohen. *A Course in Computational Algebraic Number Theory*. Berlin, Heidelberg: Springer-Verlag, 1993. ISBN: 0-387-55640-0.

[14] J. Conway and N. Sloane. *Sphere Packings, Lattices and Groups*. Vol. 290. Jan. 1988. ISBN: 978-1-4757-2018-1. DOI: 10.1007/978-1-4757-2016-7.

[15] R. Cramer, L. Ducas, and B. Wesolowski. "Short Stickelberger Class Relations and Application to Ideal-SVP". In: *EUROCRYPT*. 2017.

[16] R. Cramer et al. "Recovering Short Generators of Principal Ideals in Cyclotomic Rings". In: *Advances in Cryptology – EUROCRYPT 2016*. Ed. by M. Fischlin and J.-S. Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 559–585. ISBN: 978-3-662-49896-5.

[17] C. Fieker and C. Friedrichs. "On Reconstruction of Algebraic Numbers". In: *Algorithmic Number Theory*. Ed. by W. Bosma. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 285–296. ISBN: 978-3-540-44994-2.

[18] J. N. Franklin. *Matrix theory*. Courier Corporation, 2012.

[19] C. Hermite. "Extraits de lettres de M. Ch. Hermite à M. Jacobi sur différents objects de la théorie des nombres." In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1850 (), pp. 261–278.

[20] N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*. Vol. 444. John Wiley & Sons, 2005.

[21] P. Kirchner, T. Espitau, and P.-A. Fouque. "Towards Faster Polynomial-Time Lattice Reduction". In: *Advances in Cryptology – CRYPTO 2021*. Ed. by T. Malkin and C. Peikert. Cham: Springer International Publishing, 2021, pp. 760–790. ISBN: 978-3-030-84245-1.

[22] A. K. Lenstra et al. "The number field sieve". In: *The development of the number field sieve*. Ed. by A. K. Lenstra and H. W. Lenstra. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 11–42. ISBN: 978-3-540-47892-8.

[23] A. Lenstra, H. Lenstra, and L. Lovász. "Factoring Polynomials with Rational Coefficients". In: *Mathematische Annalen* 261 (Dec. 1982). DOI: 10.1007/BF01457454.

[24] A. Lesavourey. "A note on the discriminant and prime ramification of some real Kummer extensions". working paper or preprint. Nov. 2021.

[25] A. Lesavourey, T. Plantard, and W. Susilo. "Short Principal Ideal Problem in multicubic fields". In: *Journal of Mathematical Cryptology* 14.1 (2020), pp. 359–392. DOI: https://doi.org/10.1515/jmc-2019-0028.

[26] K. Mahler. "An inequality for the discriminant of a polynomial." In: *Michigan Mathematical Journal* 11.3 (1964), pp. 257–262.

[27] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems*. USA: Kluwer Academic Publishers, 2002. ISBN: 0792376889.

[28] J. Neukirch. "Algebraic Number Theory". In: 1999.

[29] P. Q. Nguên and D. Stehlé. "Floating-Point LLL Revisited". In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by R. Cramer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 215–233. ISBN: 978-3-540-32055-5.

[30] P. Q. Nguyen and D. Stehlé. "An LLL Algorithm with Quadratic Complexity". In: *SIAM J. Comput.* 39 (2009), pp. 874–903.

[31] *PARI/GP version 2.11.2*. available from http://pari.math.u-bordeaux.fr/. Univ. Bordeaux: The PARI Group, 2019.

[32] A. Pellet-Mary, G. Hanrot, and D. Stehlé. "Approx-SVP in Ideal Lattices with Pre-processing". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Y. Ishai and V. Rijmen. Cham: Springer International Publishing, 2019, pp. 685–716. ISBN: 978-3-030-17656-3.

[33] A. Pellet-Mary and D. Stehlé. "On the Hardness of the NTRU Problem". In: *Advances in Cryptology – ASIACRYPT 2021*. Ed. by M. Tibouchi and H. Wang. Cham: Springer International Publishing, 2021, pp. 3–35. ISBN: 978-3-030-92062-3.

[34] T. Plantard, W. Susilo, and Z. Zhang. "LLL for ideal lattices: re-evaluation of the security of Gentry–Halevi's FHE scheme". In: *Designs, Codes and Cryptography* 76 (Mar. 2014). DOI: 10.1007/s10623-014-9957-1.

[35] M. Pohst and H. Zassenhaus. *Algorithmic Algebraic Number Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1989. DOI: 10.1017/CBO9780511661952.

[36] M. E. Pohst. "Factoring polynomials over global fields I". In: *Journal of Symbolic Computation* 39.6 (2005), pp. 617–630. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2004.09.006.

[37] X.-F. Roblot. "Polynomial factorization algorithms over number fields". In: *Journal of Symbolic Computation* 38.5 (2004), pp. 1429–1443. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2004.05.002.

[38] P. Samuel. *Algebraic theory of numbers*. Hermann, 1970.

[39] D. Stehlé. "Floating-Point LLL: Theoretical and Practical Aspects". In: *The LLL Algorithm: Survey and Applications*. Ed. by P. Q. Nguyen and B. Vallée. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 179–213. ISBN: 978-3-642-02295-1. DOI: 10.1007/978-3-642-02295-1_5.

[40] The FPLLL development team. "Fplll, a lattice reduction library, Version: 5.4.1". Available at https://github.com/fplll/fplll. 2021.

[41] E. Thomé. "Square Root Algorithms for the Number Field Sieve". In: *WAIFI*. 2012.

[42] B. M. Trager. "Algebraic Factoring and Rational Function Integration". In: *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*. SYMSAC '76. Yorktown Heights, New York, USA: Association for Computing Machinery, 1976, pp. 219–226. ISBN: 9781450377904. DOI: 10.1145/800205.806338.

Appendix A. Average cost of the search

Let us study the average cost of the search phase of Algorithm 8 or Algorithm 13. In particular we will determine the average number of decodings that will occur before finding all roots. If one denotes by $\sigma_1, \dots, \sigma_n$ the elements of $\mathrm{Hom}_K(L, \mathbb{C})$, then the two algorithms can be described as a search without replacement in the set $Z = Z_1 \times \cdots \times Z_n$ where $Z_i = Z_{\sigma_i}$.

Remark A.1. A cartesian product $S = S_1 \times \cdots \times S_n$ will be ordered using the lexicographic order. This means that for all $(x, y) \in S^2$, if $i(x, y) = \min\{i \in [\![1, n]\!] \mid x_i \neq y_i\}$, then we have $x < y \iff x_{i(x,y)} < y_{i(x,y)}$.

From now on, let us consider the sets Z_i as ordered sets of elements $z_{i,j}$ with $z_{i,j} < z_{i,j'} \iff j < j'$. Moreover we consider that we run through Z following the lexicographic order. Assume that the state of the computation is at the state with index $j = (j_1, \dots, j_n) \in [\![1, \deg f(X)]\!]^n$, and that we found a root x. We can write $x = (z_{1,j_1}, \dots, z_{n,j_n})$. Then the action of `UpdateTree` on Z is as follows. We mentioned that it removes z_{i,j_i} from Z for all $i \in [\![1, n]\!]$. This amounts to removing z_{i,j_i} from Z for all $i \in [\![2, n]\!]$ and updating the index by doing $j_1 \leftarrow j_1 + 1$ and for all $i > 1, j_i \leftarrow 1$.

Definition A.2. Let N, M, m be integers satisfying $m \leqslant M \leqslant N$. A random variable X taking non-negative values follows a *negative hypergeometric distribution with parameters (N, M, m)* if it satisfies

the following formula:

$$\mathbb{P}(X=k)=\frac{\binom{k+m-1}{k}\binom{N-m-k}{M-m}}{\binom{N}{m}}.$$

We will write $X \sim NHG(N,M,m)$.

The negative hypergeometric distribution describes exactly what we want to study. Indeed, it arises as follows. Consider a set of N elements, containing M "success elements" and $N-m$ "fail elements". Then if one draws uniformly in the set *without replacement* until m successes are found, then the number of failures drawn follows a negative hypergeometric distribution.

Proposition A.3 ([20])**.** *Let N, M, m be integers, and X be a random variable such that $X \sim NHG(N,M,m)$. Then one has:*

$$\mathbb{E}[X]=m\frac{N-M}{M+1} \quad \text{and} \quad \mathrm{Var}[X]=m\frac{(N-M)(N+1)}{(M+1)(M+2)}\left(1-\frac{m}{M+1}\right).$$

We are essentially interested in the average cost, so we focus on the expectation.

Proposition A.4. *Let L/K be an extension of number fields such that $[L:K]=n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)|=s$ and write $d=\deg f(X)$. Then, assuming the search is naïve, the average number of "failed" decodings done in Algorithm 8 before finding one root is $\frac{d^n+1}{s+1}$. The average number of "failed" decodings before finding all the roots is $s\frac{d^n+1}{s+1}$.*

Proof. Let X_1 be the random variable representing the number of failures before finding the first root, and X_s be the random variable of the number of failures before finding all the roots. Clearly one has $X_1 \sim NGH(d^n,s,1)$ and $X_s \sim NGH(d^n,s,1)$, and can apply directly the formula of the expectation from Proposition A.3 to find $\mathbb{E}[X_1]$ and $E[X_s]$. □

The study is slightly more complex when using UpdateTree. One can remark that the number of elements removed from the search space Z by `UpdateTree` depends on the index of the state. We will therefore consider the random variables corresponding to the number of failures between two found solutions.

Notation A.5. Let L/K be an extension of number fields such that $[L:K]=n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)|=s$. We will denote by $x^{(1)}<\cdots<x^{(s)}$ the elements of $Z_K(f)$ ordered in Z. For each $k \in [\![1,s]\!]$ we will consider several random variables.

(1) $X^{(k)}$ is the random variable corresponding to the number of failures between the $(k-1)$-th solution and the k-th solution.
(2) $X_1^{(k)}$ is the random variable corresponding to the number of failures in the first coordinate between the $(k-1)$-th solution and the k-th solution.
(3) $X_2^{(k)}$ is the random variable corresponding to the number of failures occurred between the $(k-1)$-th solution and the k-th solution such that $j_1=x_1^{(k)}$.
(4) $Y^{(k)}$ is the random variable corresponding to the number of failures which occured before the k-th solution is found.

Lemma A.6. *Let L/K be an extension of number fields such that $[L:K]=n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)|=s$. Then one has the following:*

$$\forall k \in [\![1,s]\!], Y^{(k)}=\sum_{j=1}^{k}\left(X_1^{(j)}(d-j+1)^{n-1}+X_2^{(j)}\right).$$

Proof. Let us fix $k \in [\![1, s]\!]$. Clearly one has $Y^{(k)} = \sum_{j=1}^{(k)} X^{(j)}$. Now let us denote by C_k the integer $|Z_2 \times \cdots \times Z_n|$ after the $(k-1)$-th root and before the k-th root are found. Recall that after each new solution is found, `UpdateTree` removes one element of each $Z_i, i > 1$. Therefore, one obtains $C_k = (d-k+1)^{n-1}$. Because the search is done following the lexicographic order, it is easy to see that for each fixed $j_1 \in [\![x_1^{(k-1)} + 1, x_1^{(k)}]\!]$ there are two possibilities. If $j_1 < x_1^{(k)}$ then the search will run through all $\{z_{1,j_1}\} \times Z_2 \times \cdots \times Z_n$, which contains no solution. This leads to C_k failures. If $j_1 = x_1^{(k)}$ then the search will run through $\{z_{1,j_1}\} \times Z_2 \times \cdots \times Z_n$ until finding the solution. It amounts to $X_2^{(k)}$ failures. Finally the number of j_1 that are passed such that $j_1 < x_1^{(k)}$ is $X_1^{(k)}$. □

Proposition A.7. *Let L/K be an extension of number fields such that $[L : K] = n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)| = s$. Write $d = \deg f(X)$. Then, for $k \in [\![1, s]\!]$, the average number of "failed" decodings done in Algorithm 8 before finding k roots is*

$$\frac{2d - s + 1}{2(s+1)} \sum_{j=0}^{k-1} (d-j)^{n-1} - \frac{k}{2}. \tag{A.1}$$

Proof. Using Lemma A.6 and by linearity of the expectation, one has:

$$\forall k \in [\![1, s]\!], \mathbb{E}[Y^{(k)}] = \sum_{j=1}^{k} \left(\mathbb{E}[X_1^{(j)}](d-j+1)^{n-1} + \mathbb{E}[X_2^{(j)}] \right).$$

Let fix $j \in [\![1, k]\!]$. Recall that $X_2^{(j)}$ is the number of failures found during the search through the set $\{x_1^{(j)}\} \times Z_2 \times \cdots \times Z_n$. We know there is exactly one solution in this set. Therefore we have $X_2^{(j)} \sim NHG((d-j+1)^{n-1}, 1, 1)$ and

$$\mathbb{E}[X_2^{(j)}] = \frac{(d-j+1)^{n-1} - 1}{2}.$$

Now let us determine $X_1^{(j)}$. It is the number of wrong first coordinates visited until finding $x_1^{(j)}$, and after finding $x_1^{(j-1)}$. The search is done over the set Z_1 minus the elements visited before $x_1^{(j-1)}$ included. It is a set with cardinal number

$$|Z_1| - \sum_{i=1}^{(j-1)} X_1^{(i)} - (j-1) = d - j + 1 - \sum_{i=1}^{(j-1)} X_1^{(i)}$$

which contains $s - (j-1)$ success elements. Therefore we have $\mathbb{E}[X_1^{(j)} \mid \sum_{i=1}^{(j-1)} X_1^{(i)} = a] \sim NHG(d - j + 1 - a, s - (j-1), 1)$, for all possible a. Using the law of total expectation $\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$ between two variables X and Y, it is easy to see that we have

$$\mathbb{E}[X_1^{(j)}] = \frac{(d-j+1) - \sum_{i=1}^{(j-1)} \mathbb{E}[X_1^{(i)}] - (s-j+1)}{s-j+2} = \frac{(d-s) - \sum_{i=1}^{(j-1)} \mathbb{E}[X_1^{(i)}]}{s-j+2}.$$

It is possible to use this recurrence relation to obtain an expression of $\mathbb{E}[X_1^{(j)}]$ in closed-form. As a matter of fact, we will prove that $\mathbb{E}[X_1^{(j+1)}] = \mathbb{E}[X_1^{(j)}]$. Indeed one has

$$\mathbb{E}[X_1^{(j+1)}] = \frac{(d-s) - \sum_{i=1}^{(j)} \mathbb{E}[X_1^{(i)}]}{s-j+1} = \frac{d - s - \mathbb{E}[X_1^{(i)}] - \sum_{i=1}^{(j-1)} \mathbb{E}[X_1^{(i)}]}{s-j+1},$$

and since

$$(s-j+2)\mathbb{E}[X_1^{(j)}] = (d-s) - \sum_{i=1}^{(j-1)} \mathbb{E}[X_1^{(i)}]$$

we obtain

$$\mathbb{E}[X_1^{(j+1)}] = \frac{d-s-\mathbb{E}[X_1^{(i)}]+(s-j+2)\mathbb{E}[X_1^{(j)}]-(d-s)}{s-j+1} = \mathbb{E}[X_1^{(j)}].$$

Then remark that $X_1^{(1)} \sim NHG(d,s,1)$, which gives the desired result. □

Remark A.8. One can see that the gain of using `UpdateTree` increases with s. Indeed if $f(X)$ has only one root, then the expected number of decodings is the same for both methods. However, if $s = \deg f(X)$ then the expected number of decodings with the naive search is $d\frac{d(d^{n-1}-1)}{d+1}$, whereas it is $\sum_{i=0}^{d-1} \frac{(d-i)^{n-1}-1}{2}$ when using `UpdateTree`.

Let us analyse more precisely the difference between the two procedures.

Notation A.9. Let L/K be an extension of number fields such that $[L:K] = n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)| = s$. For each $k \in [\![1,s]\!]$ we will consider several random variables, all related to a naïve search.

(1) $X_{\text{naive}}^{(k)}$ is the random variable corresponding to the number of failures between the $(k-1)$-th solution and the k-th solution.
(2) $Y_{\text{naive}}^{(k)}$ is the random variable corresponding to the number of failures which occured before the k-th solution is found.

Finally we will denote by $S_X^{(k)}$ the difference $X_{\text{naive}}^{(k)} - X^{(k)}$ and by $S_Y^{(k)}$ the difference $Y_{\text{naive}}^{(k)} - Y^{(k)}$.

Proposition A.10. *Let L/K be an extension of number fields such that $[L:K] = n$. Consider $f(X) \in L[X]$ such that $|Z_K(f)| = s$. Write $d = \deg f(X)$. Then, for $k \in [\![1,s]\!]$ we have $\mathbb{E}[X_{\text{naive}}^{(k)}] \geqslant \mathbb{E}[X^{(k)}]$, $\mathbb{E}[Y_{\text{naive}}^{(k)}] \geqslant \mathbb{E}[Y^{(k)}]$ and the following is true:*

$$S_Y^{(k)} = k\frac{(s-1)(d^{n-1}-1)}{2(s+1)} + \frac{2d-s+1}{s+1}\sum_{i=0}^{k-1}(d^{n-1}-(d-i)^{n-1}). \tag{A.2}$$

Proof. First, for all $k \in [\![1,s]\!]$ we will denote by c_k the integer $d-k+1$. Then, from the proofs of Lemma A.6 and Proposition A.7 we get

$$\mathbb{E}[X^{(k)}] = \frac{d-s}{s+1}c_k^{n-1} + \frac{c_k^{n-1}-1}{2} = \frac{2dc_k^{n-1}-(s-1)c_k^{n-1}-s-1}{2(s+1)}.$$

Moreover $\mathbb{E}[X_{\text{naive}}^{(k)}] = \frac{d^n-s}{s+1}$. Therefore we obtain

$$\begin{aligned} S_X^{(k)} &= \frac{2d^n-2s-2dc_k^{n-1}+(s-1)c_k^{n-1}+s+1}{2(s+1)} \\ &= \frac{2d(d^{n-1}-c_k^{n-1})+(s-1)(c_k^{n-1}-1)}{2(s+1)}, \end{aligned}$$

which is positive. Now let us determine how this difference evolves with k. Clearly c_k decreases when k increases, therefore so does $\mathbb{E}[X^{(k)}]$. Since $\mathbb{E}[X_{\text{naive}}^{(k)}]$ is constant, we can conclude that $S_X^{(k)}$ increases with k. Let us express $S_Y^{(k)}$. Given any sequence $(u_n)_n$ let us denote $u_{n+1}-u_n$ by $\Delta(u,n)$. Remark that $S_Y^{(k)} = \sum_{i=1}^k S_X^{(i)}$ and $S_X^{(k)} = S_X^{(1)} + \sum_{i=1}^{k-1} S_X^{(i+1)} - S_X^{(i)} = S_X^{(1)} + \sum_{i=1}^{k-1}\Delta(S_X,i)$. Moreover one has

$$\Delta(S_X,i) = \mathbb{E}[X^{(i+1)}] - \mathbb{E}[X^{(i)}] = \frac{(2d-s+1)(-c_{i+1}^{n-1}+c_i^{n-1})}{2(s+1)}$$

which leads to

$$S_X^{(k)} = S_X^{(1)} + \frac{2d-s+1}{s+1}\sum_{i=1}^{k-1}\Delta((-c_k^{n-1})_k, i)$$
$$= \frac{(s-1)(d^{n-1}-1)}{2(s+1)} + \frac{2d-s+1}{s+1}(d^{n-1}-c_k^{n-1}).$$

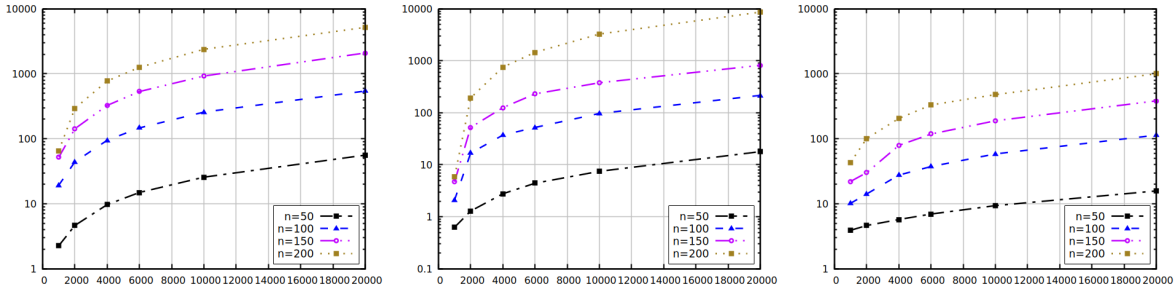
Finally we can write

$$S_Y^{(k)} = \sum_{i=1}^{k}\frac{(s-1)(d^{n-1}-1)}{2(s+1)} + \frac{2d-s+1}{s+1}(d^{n-1}-c_i^{n-1})$$
$$= k\frac{(s-1)(d^{n-1}-1)}{2(s+1)} + \frac{2d-s+1}{s+1}\sum_{i=0}^{k-1}(d^{n-1}-(d-i)^{n-1}).$$

□

Appendix B. Extra data

B.1. **Different versions of LLL.** In this section we present the timings obtained when comparing the efficiency of LLL with our method SpecLLL (Alg. 9), which correspond to the ratio presented in Figures 2 and 3. One can find said timings in Figures 14 and 15.



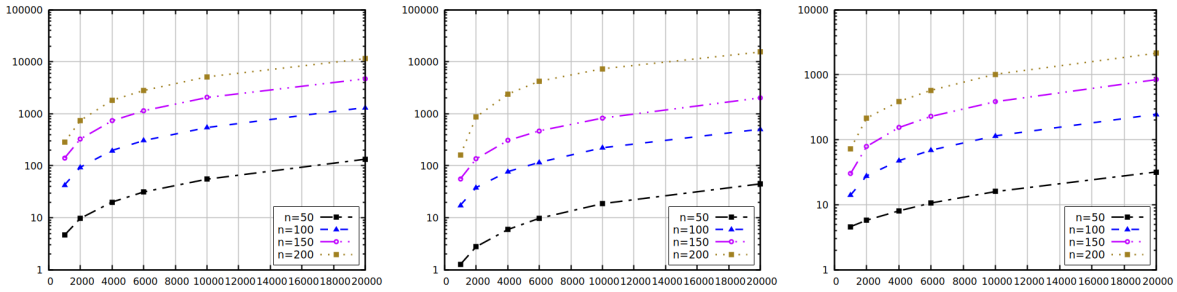
(a) Pari/Gp (b) Magma (c) Fplll

Figure 14. Timings (s) for $\sigma(K) \subset \mathbb{R}$ plotted against the precision

As mentioned previously, Fplll is faster than Magma or Pari/Gp. One can remark that the latter is around 5 times slower than Fplll asymptotically. This gives an idea of the timings that could be obtained for all our experiments by switching from the implementation of Pari/Gp to Fplll.

Precision		1000	2000	4000	6000	10000	20000
$n = 50$	LLL	2.5	5.1	10.7	16.8	29.8	69.4
	SpecLLL	2.6	5.2	10.8	16.8	29.8	69.3
$n = 150$	LLL	74.5	166.0	371.2	639.3	1132.0	2843.6
	SpecLLL	76.1	167.6	367.7	634.9	1118.1	2841.0

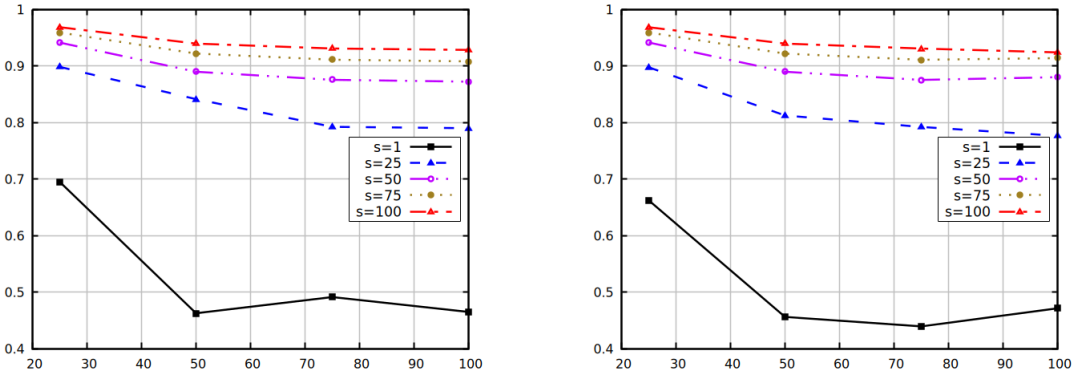
Table 6. Timings (s) over random knapsack-like matrices



(A) Pari/Gp (B) Magma (C) Fplll

FIGURE 15. Timings (s) for $\sigma(K) \not\subset \mathbb{R}$ plotted against the precision

B.2. **Precision evaluation.** In Figure 16 , we plotted the ratio of $\max q_t$ by $1+\ln(n)\ln\ln(n)/(s\ln(2)+\ln(n)/2)$.



(A) $s(P)=1$ (B) $s(P)=5$

FIGURE 16. Maximum ratios of q_t by $1+\ln n\ln\ln(n)/(s\ln 2+\ln(n)/2)$ plotted against $n=[K:\mathbb{Q}]$, for several s.

Note that we did similar experiments using an integral basis of $\mathcal{O}_K$ instead of the equation basis $(1, X, \ldots, X^{n-1})$ and obtained similar results, which can be found in Figure 17. We were to smaller degrees because of the cost of computing such basis.

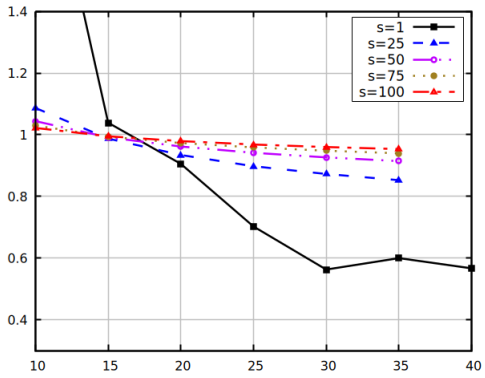
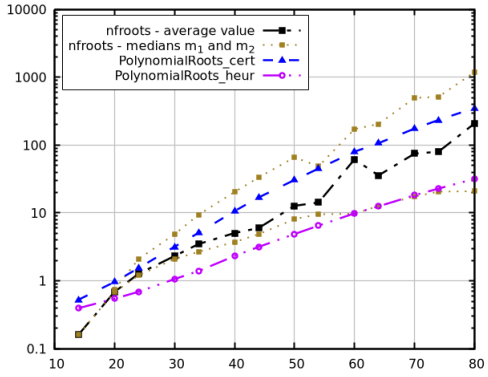


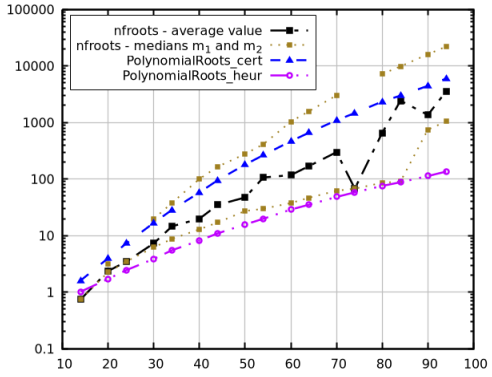
FIGURE 17. Maximum ratios of q_t by $1 + \ln n \ln\ln(n)/(s \ln 2 + \ln(n)/2)$ plotted against $n = [K : \mathbb{Q}]$ for several s, where $\mathcal{B}$ is an integral basis of $\mathcal{O}_K$

One can see that ratio of q_t by $1 + \ln n \ln\ln(n)/(s \ln 2 + \ln(n)/2)$ is larger than 1 only for very small dimensions, which also happens for the equation basis. In practice, the additional precision that is required does not impact significatively the running times the algorithms.

B.3. **Absolute method.** In this section we give data corresponding to experiments presented in Section 6.2, in the case where $r_1 = 0$, and consequently $\sigma(K) \subset \mathbb{C} \setminus \mathbb{R}$.

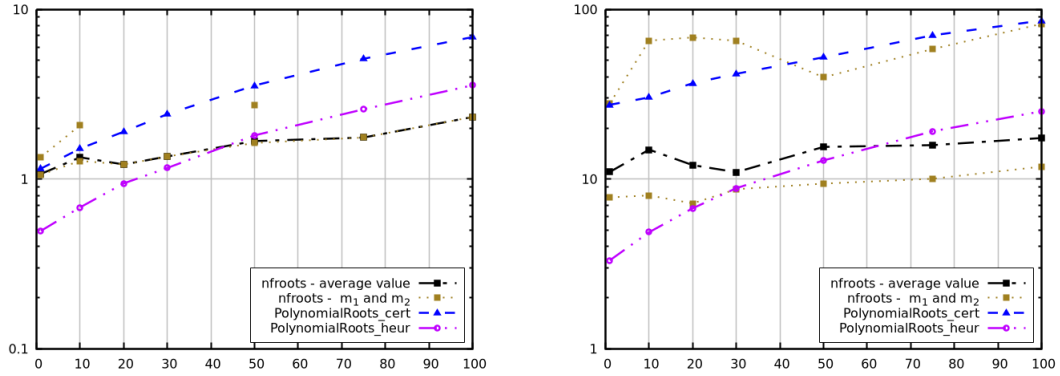


(A) $s(P_K) = 1$



(B) $s(P_K) = 10$

FIGURE 18. Average timings (s) of nfroots, AbsoluteRoots and AbsoluteRootsHeur plotted against $\deg P_K(X)$ for randomly generated $P_K(X)$ such that $r_1 = 0$, with $s(P_K) \in \{1, 10\}$



(A) $[K : \mathbb{Q}] = 24$ (B) $[K : \mathbb{Q}] = 50$

FIGURE 19. Average timings (s) of nfroots, AbsoluteRoots and AbsoluteRootsHeur plotted against s_Z for randomly generated $P_K(X)$ such that $[K : \mathbb{Q}] = 24$ and $[K : \mathbb{Q}] = 50$, with $r_1 = 0$