Algebraic number fields and the LLL algorithm

M. János Uray
uray.janos@inf.elte.hu
ELTE – Eötvös Loránd University (Budapest)
Faculty of Informatics
Department of Computer Algebra

Abstract

In this paper we analyze the computational costs of various operations and algorithms in algebraic number fields using exact arithmetic. Let K be an algebraic number field. In the first half of the paper, we calculate the running time and the size of the output of many operations in K in terms of the size of the input and the parameters of K. We include some earlier results about these, but we go further than them, e.g. we also analyze some $\mathbb{R}$-specific operations in K like less-than comparison.

In the second half of the paper, we analyze two algorithms: the Bareiss algorithm, which is an integer-preserving version of the Gaussian elimination, and the LLL algorithm, which is for lattice basis reduction. In both cases, we extend the algorithm from $\mathbb{Z}^n$ to K^n, and give a polynomial upper bound on the running time when the computations in K are performed exactly (as opposed to floating-point approximations).

1 Introduction

Exact computation with algebraic numbers is an important feature that most computer algebra systems provide. They use efficient algorithms for the calculations, described in several papers and books, e.g. in [1, 2, 3]. However, the computational costs of these algorithms are often not obvious to calculate, because the bit complexity depends on how much the representing multi-precision integers grow during the computation.

In this paper, given an algebraic number field K, we give explicit bounds on the costs of many operations in K, using the size of the input and the parameters of K. We consider polynomial representation, i.e. when the elements are represented by polynomials of a fixed primitive element $\theta \in K$. There are already some work in this area, e.g. [4] and [5], and we use the existing results whenever we can, but in some cases, we replace them with more general or in other ways more suitable results. We also consider operations that are special to $\mathbb{R}$, like less-than comparison and integer rounding functions, and the author is not aware of any previous results on these.

The obtained explicit formulas enable us to calculate the running time of several well-known algorithms if we apply them to K with exact arithmetic. For example consider the Gaussian elimination, which performs $O(n^3)$ arithmetic operations, but if we use exact arithmetic in $\mathbb{Z}$ or K, then the size of the entries may grow exponentially. The Bareiss algorithm [6] is a modification for $\mathbb{Z}$ which addresses this problem by certain simplifications to ensure polynomial coefficient growth and running time (although with larger exponent). The idea can be generalized to K in a straightforward manner, which is calculated in this paper.

Another algorithm of our interest is the LLL lattice reduction algorithm [7], which converts a basis of a lattice to a reduced basis of the same lattice. It is a well-studied algorithm for integers and floating-point numbers. The creators of the algorithm showed in [7] that the algorithm runs in polynomial time if the input vectors are in $\mathbb{Z}^n$. There are also many efficient generalizations for $\mathbb{R}^n$ using floating-point arithmetic ([8, 9, 10, 11]). But we are interested in another generalization: we consider the LLL algorithm on vectors in K^n for real K with exact arithmetic. The LLL algorithm over number fields has already been studied (e.g. in [12, 13]), but we are not aware of a complexity analysis using exact arithmetic. In the present paper, using the calculated operation costs in K, we analyse the algorithm: we find an upper bound on the number of main iterations, examine how the sizes of the variables grow during these iterations, and conclude with an explicit upper bound on the running time of the algorithm. All these results are given in terms of very basic parameters like the dimension, the properties of K and the bit size of the input. The running time turns out to be polynomial in these parameters, though with larger exponents than in $\mathbb{Z}$.

Exact calculation in the LLL algorithm might not seem very useful, since for many practical applications, the goal is to find a well-reduced basis or a sufficiently short vector, therefore the floating-point LLL versions can be applied, and they are much faster than exact computation. However, we still think that the analysis of the exact algorithm in K^n deserves interest. First, it is interesting from a theoretical point of view. Second, there are applications when the exact values in the reduction are needed. For example in [14], algebraic integers are represented by ultimately periodic series of integer vectors, obtained by a repeated application of the LLL algorithm. This representation is a generalization of continued fractions, and as with continued fractions, the exact representation is not guaranteed to be obtained if we use approximations. This is analoguous to the Gaussian elimination where the floating-point version is preferred whenever applicable, but if we need the exact result, we have no choice but to use a slower but exact version like the Bareiss algorithm.

The paper is built up as follows: Section 2 analyzes the computational costs of several operations in K, Section 3 calculates the running time of the Bareiss algorithm over K, and Section 4 does the same with the LLL algorithm.

2 Operations in number fields

Let K be an algebraic number field of degree m, and let $\theta \in K$ be a primitive element, i.e. $K = \mathbb{Q}(\theta)$. For some results we will require that $\theta \in \mathbb{R}$ and so $K \subset \mathbb{R}$, but for the others, we state them more generally. Without loss of generality we can assume that θ is

an algebraic integer. Denote its minimal polynomial by $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ ($f_i \in \mathbb{Z}$). We will consider f, θ and m as fixed throughout this article.

Elements of K can be represented by rational linear combinations of $1, \theta, \theta^2, \ldots, \theta^{m-1}$. However, in order to minimalize the problems with rational numbers like simplification, we use an integer linear combination and a common denominator. Furthermore, we consider only the numerator, i.e. the ring $\mathbb{Z}[\theta]$, because dealing with the single denominator is trivial, and in many algorithms, they can be cleared in the beginning.

We define $H(f)$, the height of f, and a related quantity:

$$H(f) := \max_{i=0}^{m-1} |f_i|, \tag{2.1}$$

$$F := \log(H(f) + 1). \tag{2.2}$$

(Note that in this paper, the bases of the logarithms are mostly omitted, but we mean the same fixed base throughout the paper. We can think of them as binary logarithms, because they mainly mean the number of bits, but the calculations remain consistent for any fixed ≥ 2 base, as changing the base makes only a constant factor difference.)

In many results, the dependence on the number field will be expressed by the two constants m and F. We start with two simple bounds on θ:

Lemma 2.1.

$$\log |\theta| < F, \tag{2.3}$$

$$\log \left(1 + |\theta| + |\theta|^2 + \ldots + |\theta|^{m-1}\right) < mF. \tag{2.4}$$

Proof. Cauchy's inequality states that $|\theta| < H(f) + 1$ (see e.g. [2, p. 323]), which is equivalent to the first statement. The second one is a consequence:

$$1 + |\theta| + |\theta|^2 + \ldots + |\theta|^{m-1} < 1 + (H(f)+1) + (H(f)+1)^2 + \ldots + (H(f)+1)^{m-1} =$$
$$= \frac{(H(f)+1)^m - 1}{(H(f)+1) - 1} < (H(f)+1)^m.$$

□

2.1 Size of the elements

We use some results from [5, Sect. 3]. The authors use a more general representation: they fix an integral basis in the number field: $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$ with $\omega_1 = 1$, and write algebraic integers as $a_1\omega_1 + a_2\omega_2 + \ldots + a_m\omega_m$ for $a_1, \ldots, a_m \in \mathbb{Z}$.

They introduce three constants, C_1, C_2 and C_3, dependent on Ω, and state their results in terms of these constants. First, they define C_1 and C_2 by stating in [5, (1)] that (using our notation):

Lemma 2.2. *Let $x := (a_0, a_1, \ldots, a_{m-1})^T$ be the coefficient vector of a, denote the roots of f by $\theta_1, \theta_2, \ldots, \theta_m$, let $a^{(i)} := a_0 + a_1\theta_i + \ldots + a_{m-1}\theta_i^{m-1}$, and let $y := (a^{(1)}, a^{(2)}, \ldots, a^{(m)})^T$. Then there exists such $C_1, C_2 > 0$ that*

$$\frac{1}{C_2}\|x\|_\infty \leq \|y\|_2 \leq C_1\|x\|_\infty.$$

Then they define C_3 to be a bound on the cofficients $m_{i,j,k}$ appearing in the

multiplication table of $(\omega_i\omega_j)_{i,j}$:

$$\omega_i\omega_j = m_{i,j,1}\omega_1 + m_{i,j,2}\omega_2 + \ldots + m_{i,j,m}\omega_m,$$
$$C_3 := \max_{i,j,k} |m_{i,j,k}|.$$

In order to use their results, we need to express C_1, C_2 and C_3 for our special case, $\Omega = \{1, \theta, \theta^2, \ldots, \theta^{m-1}\}$. We have the following result:

Lemma 2.3.

$$\log C_1 < mF + \frac{1}{2}\log m, \tag{2.5}$$

$$\log C_2 < \binom{m}{2} F + m \log m, \tag{2.6}$$

$$\log C_3 < (m-1)F. \tag{2.7}$$

Proof. We can describe the connection between x and y from Lemma 2.2 as $y = Vx$ where V is the following Vandermonde matrix:

$$V = \begin{pmatrix} 1 & \theta_1 & \theta_1^2 & \ldots & \theta_1^{m-1} \\ 1 & \theta_2 & \theta_2^2 & \ldots & \theta_2^{m-1} \\ \vdots & \vdots & & & \vdots \\ 1 & \theta_m & \theta_m^2 & \ldots & \theta_m^{m-1} \end{pmatrix}.$$

Since $\|y\|_\infty \le \|y\|_2 \le \sqrt{m}\,\|y\|_\infty$, we can choose C_1 and C_2 as

$$C_1 := \sqrt{m}\,\|V\|_\infty, \qquad C_2 := \|V^{-1}\|_\infty,$$

so we need to find $\|V\|_\infty$ and $\|V^{-1}\|_\infty$.

The former can be calculated using (2.4), thus proving (2.5):

$$\log \|V\|_\infty = \log \max_{i=1}^{m} (1 + |\theta_i| + |\theta_i|^2 + \ldots + |\theta_i|^{m-1}) < mF.$$

For (2.6), we need $\|V^{-1}\|_\infty$. The inverse of V can be written as $(V^{-1})_{ij} = C_{ji}/\det V$, where C_{ij} is the minor of V at row i and column j multiplied by $(-1)^{i+j}$. It is well-known that the determinant of the Vandermonde matrix is (see e.g. [2, p. 185]):

$$\det V = \prod_{i=1}^{m} \prod_{j=i+1}^{m} (\theta_j - \theta_i),$$

and also that its square is the discriminant of f (see e.g. [1, Prop. 3.3.5.]). Since the latter is integer and nonzero, $|\det V| \ge 1$, so $|(V^{-1})_{ij}| \le |C_{ji}|$. The cofactor C_{ji} is an $(m-1)\times(m-1)$ determinant, whose full expansion has $(m-1)!$ terms, where each term is a product of $m-1$ elements of V in different columns. Since $V_{kl} = \theta_k^{l-1}$, $\log|V_{kl}| < (l-1)F$ by (2.3), and the sum of exponents in each product is at most $0+1+2+\ldots+(m-1) = \binom{m}{2}$, we get $\log|C_{ji}| < \binom{m}{2}F + \log(m-1)!$. Therefore, we can finish (2.6) by:

$$\begin{aligned} \log C_2 = \log \|V^{-1}\|_\infty &= \log \max_{i=1}^{m} \left(\left|(V^{-1})_{i1}\right| + \left|(V^{-1})_{i2}\right| + \ldots + \left|(V^{-1})_{im}\right| \right) \le \\ &\le \log \max_{i=1}^{m} (|C_{1i}| + |C_{2i}| + \ldots + |C_{mi}|) < \\ &< \binom{m}{2} F + \log(m-1)! + \log m \le \binom{m}{2} F + m \log m. \end{aligned}$$

For (2.7), we write θ^{m+k} for $k = 0, 1, \ldots, m-2$ in terms of lower powers of θ:

$$\theta^{m+k} = r_{k,0} + r_{k,1}\theta + r_{k,2}\theta^2 + \ldots + r_{k,m-1}\theta^{m-1}, \tag{2.8}$$

and we have $C_3 = \max_{k=0}^{m-2} \max_{j=0}^{m-1} |r_{k,j}|$.

By using that $f(\theta) = 0$, one can get a recursive formula for $r_{k,l}$ (see e.g. [1, p. 159]):

$$r_{0,l} = -f_l, \qquad r_{k+1,l} = r_{k,l-1} - f_l r_{k,m-1}, \tag{2.9}$$

where $r_{k,-1} = 0$. Then one can easily show by induction that:

$$|r_{k,l}| < (H(f) + 1)^{k+1}, \tag{2.10}$$

and (2.7) follows. □

For an algebraic integer $a \in \mathbb{Z}[\theta]$, $a = a_0 + a_1\theta + a_2\theta^2 + \ldots + a_{m-1}\theta^{m-1}$ ($a_i \in \mathbb{Z}$), we will use the following function to measure its coefficient size:

$$\mathrm{s}(a) := \begin{cases} \log \max_{i=0}^{m-1} |a_i| & \text{if } a \neq 0, \\ 0 & \text{if } a = 0. \end{cases} \tag{2.11}$$

This quantity together with the field degree m (which is constant for a fixed field) indicates the storage size needed by the algebraic integer a. (Note that when comparing results with [5], their notation $\mathrm{S}(a)$ is slightly different: $\mathrm{S}(a) = m\,\mathrm{s}(a)$.) For an integer n, we simply have $\mathrm{s}(n) = \log|n|$ if $n \neq 0$, otherwise 0. For convenience, we also define $\mathrm{s}(a_1, a_2, \ldots, a_n) := \max_{i=1}^{n} \mathrm{s}(a_i)$.

The following several results show how much $\mathrm{s}(\cdot)$ can grow during several operations. First we start with the most trivial ones.

Lemma 2.4.

$$a, b \in \mathbb{Z}[\theta]: \qquad \mathrm{s}(a \pm b) \leq \mathrm{s}(a, b) + \log 2, \tag{2.12}$$

$$n \in \mathbb{Z}^+,\ a_1, \ldots, a_n \in \mathbb{Z}[\theta]: \qquad \mathrm{s}\,(a_1 + \ldots + a_n) \leq \mathrm{s}(a_1, \ldots, a_n) + \log n, \tag{2.13}$$

$$n \in \mathbb{Z},\ a \in \mathbb{Z}[\theta]: \qquad \mathrm{s}(na) \leq \mathrm{s}(a) + \mathrm{s}(n). \tag{2.14}$$

For the multiplicative operations, [5, before Prop. 4] shows the following:

Lemma 2.5. *If $a, b \in \mathbb{Z}[\theta]$, then:*

$$\mathrm{s}(ab) \leq \mathrm{s}(a) + \mathrm{s}(b) + \log C_3 + 2\log m,$$

and if $a \in \mathbb{Z}[\theta]$, $a \neq 0$ and $a^{-1} = b/n$ with $b \in \mathbb{Z}[\theta]$ and $n \in \mathbb{Z}$, then:

$$\mathrm{s}(b) \leq (m-1)\,\mathrm{s}(a) + (m-1)\log C_1 + \log C_2 + \frac{1}{2}\log m,$$

$$\log|n| \leq m\,\mathrm{s}(a) + m\log C_1 - \frac{m}{2}\log m.$$

Substituting our bounds on C_1, C_2 and C_3 from (2.5), (2.6) and (2.7) gives the following:

Lemma 2.6. *If $a, b \in \mathbb{Z}[\theta]$, then:*

$$\mathrm{s}(ab) \leq \mathrm{s}(a) + \mathrm{s}(b) + (m-1)F + 2\log m, \tag{2.15}$$

and if $a \in \mathbb{Z}[\theta]$, $a \neq 0$ and $a^{-1} = b/n$ with $b \in \mathbb{Z}[\theta]$ and $n \in \mathbb{Z}$, then:

$$\mathrm{s}(b) \leq (m-1)\,\mathrm{s}(a) + \frac{3}{2}m(m-1)F + \frac{3}{2}m\log m, \tag{2.16}$$

$$\log|n| \leq m\,\mathrm{s}(a) + m^2 F. \tag{2.17}$$

The next result is an upper and a lower bound for the absolute value of an element in terms of its coefficient size.

Lemma 2.7. *If $a \in \mathbb{Z}[\theta]$ and $a \neq 0$:*

$$\log|a| < \mathrm{s}(a) + mF, \tag{2.18}$$

$$\log|a^{-1}| < (m-1)\,\mathrm{s}(a) + \frac{3}{2}m^2(F+1). \tag{2.19}$$

Proof. (2.18) follows from (2.4):

$$\log|a| = \log\left|\sum_{j=0}^{m-1} a_j\theta^j\right| \leq \mathrm{s}(a) + \log\sum_{j=0}^{m-1}|\theta|^j < \mathrm{s}(a) + mF.$$

(2.19) comes from (2.18) and (2.16):

$$\log|a^{-1}| = \log\frac{|b|}{|n|} \leq \log|b| < \mathrm{s}(b) + mF \leq$$

$$\leq (m-1)\,\mathrm{s}(a) + m\frac{3m-1}{2}F + \frac{3}{2}m\log m,$$

and after some simplifications, we get (2.19). □

Sometimes we need to work with determinants involving algebraic integers, so we derive the following lemma from the previous results.

Lemma 2.8. *Let $A \in \mathbb{Z}[\theta]^{n\times n}$ be a matrix with entries a_{ij}. Then:*

$$\mathrm{s}(\det A) \leq n\max_{i,j}\mathrm{s}(a_{ij}) + (n-1)\left((m-1)F + 2\log m\right) + n\log n.$$

Proof. The full expansion of the determinant is:

$$\det A = \sum_{\sigma\in S_n}(-1)^{N(\sigma)}\prod_{i=1}^{n} a_{i,\sigma(i)}, \tag{2.20}$$

where S_n is the set of all $n!$ permutations of $\{1,2,\ldots,n\}$, and $N(\sigma)$ is the number of inversions in σ. Applying (2.13):

$$\mathrm{s}(\det A) \leq \max_{\sigma\in S_n}\mathrm{s}\left(\prod_{i=1}^{n} a_{i,\sigma(i)}\right) + \log n!,$$

and the proof finishes by applying (2.15) repeatedly and using that $n! \leq n^n$. □

2.2 Running time of field operations

In this section we give bounds on the running time of several operations in algebraic number fields. There are already some results in this topic: [4, Sect. 5] and [5, Sect. 3]. However, [4] presents only multiplication and not in the form we can use (e.g. it does not differentiate between F and the input size), and the results of [5] are not general enough (e.g. considering only fast multiplication). Nevertheless, we will use some ideas from these works. And we also present some operations that neither of them discusses.

As the field elements are represented by integers, these calculations rely on the running time of integer operations, especially multiplication and division. But there are several different algorithms for multiplying and dividing arbitrarily large integers, and each have different time complexity. For the sake of generality, we give our results in terms of the complexity of integer multiplication, using the following notation, similar to that in [4].

Let $\mathrm{Mul}(A,B)$ be the running time of multiplying two integers $a, b \in \mathbb{Z}$ whose bit length are bounded by A and B (i.e. $\log|a| \leq A$ and $\log|b| \leq B$), and let $\mathrm{Mul}(A) :=$

$\mathrm{Mul}(A, A)$. The value depends on the actual integer multiplication algorithm used, for example:

- basic multiplication: $\mathrm{Mul}(A, B) = O(AB)$,
- Karatsuba multiplication: $\mathrm{Mul}(A) = O(A^{\log_2 3})$,
- Schönhage–Strassen algorithm: $\mathrm{Mul}(A) = O(A \log A \log\log A)$.

For more details about integer multiplication, see e.g. [16, 4.3.3.]. In this paper we use only the following assumptions about the Mul function:

$$\begin{aligned} \mathrm{Mul}(A, B) &= \mathrm{Mul}(B, A), \\ B \le C &\Rightarrow \mathrm{Mul}(A, B) \le \mathrm{Mul}(A, C), \\ \mathrm{Mul}(A, B + C) &\le \mathrm{Mul}(A, B) + \mathrm{Mul}(A, C), \\ \mathrm{Mul}(A, nB) &\le n\,\mathrm{Mul}(A, B) \quad (n \in \mathbb{Z}^+), \\ n\,\mathrm{Mul}(A) &\le \mathrm{Mul}(nA), \\ A \le \mathrm{Mul}(A) &\le A^2. \end{aligned}$$

We assume furthermore that the integer division a/b with quotient q and remainder r can be performed in $O(\mathrm{Mul}(\log|b|, \log|q|))$ time (see e.g. [16, 4.3.3. D]).

We denote by $\mathrm{T}(expr)$ the number of arithmetic operations on machine words to calculate the expression $expr$. First we consider the most trivial operations:

Lemma 2.9. *If $a, b \in \mathbb{Z}[\theta]$ and $n \in \mathbb{Z}$, then*

$$\mathrm{T}(a \pm b) = O\left(m\,\mathrm{s}(a, b))\right), \tag{2.21}$$

$$\mathrm{T}(na) = O\left(m\,\mathrm{Mul}(\mathrm{s}(n), \mathrm{s}(a))\right). \tag{2.22}$$

For multiplication, we have the following result:

Lemma 2.10. *If $a, b \in \mathbb{Z}[\theta]$, then*

$$\mathrm{T}(ab) = O\left(m^2\,\mathrm{Mul}(\mathrm{s}(a), \mathrm{s}(b)) + m^2\,\mathrm{Mul}(mF,\, \mathrm{s}(a) + \mathrm{s}(b) + \log m)\right). \tag{2.23}$$

Proof. The product $c = ab$ can be computed by the following steps:

1. Calculate the product as if θ were a symbol, i.e. perform a polynomial multiplication:
$$d_l := \sum_j a_j b_{l-j} \quad (0 \le l \le 2m - 2).$$
2. Calculate its remainder modulo f:
$$c_l := d_l + \sum_{k=0}^{m-2} d_{m+k} r_{k,l} \quad (0 \le l \le m - 1),$$
where $r_{k,l}$ are defined by (2.8), and they can be precalculated from f by (2.9).

Both steps are dominated by the $O(m^2)$ multiplications, so the total running time is:

$$\mathrm{T}(ab) = O\left(m^2\,\mathrm{Mul}(\mathrm{s}(a), \mathrm{s}(b)) + m^2\,\mathrm{Mul}(\log \max_l |d_l|, \log \max_{k,l} |r_{k,l}|)\right).$$

To bound the variables in the second $\mathrm{Mul}(\cdot)$, we use the definition of d_l above and (2.10) for $r_{k,l}$:

$$\begin{aligned} \log|d_l| &\le \log\left(\sum_j |a_j||b_{l-j}|\right) \le \mathrm{s}(a) + \mathrm{s}(b) + \log m, \\ \log|r_{k,l}| &\le (k+1)F \le mF, \end{aligned}$$

and substituting these gives the statement. □

There is another method, used by [5], to calculate $c = ab$: we calculate a matrix M_a from a which turns b into c, i.e. $M_a(b_0, b_1, \ldots, b_{m-1})^T = (c_0, c_1, \ldots, c_{m-1})^T$. In our case, it can be derived from the formulas above that:

$$(M_a)_{i,j} = a_{i-j} + \sum_{k=0}^{j-2} a_{m+k-j+1} r_{k,i-1}, \tag{2.24}$$

(where a_{i-j} is meant to be 0 when $i < j$). Calculating M_a therefore requires

$$\mathrm{T}(M_a) = O\left(m^3 \operatorname{Mul}(\mathrm{s}(a), mF)\right) \tag{2.25}$$

time. This is already slower than our method with (2.23), but if we often multiply by the same a, we can calculate M_a only once, and then the actual multiplication is just a matrix-vector multiplication, which has a slightly better complexity:

$$\mathrm{T}(ab)_{M_a} = O\left(m^2 \operatorname{Mul}(\mathrm{s}(a) + mF, \mathrm{s}(b))\right).$$

There is also another use of M_a, when calculating the inverse of a.

Lemma 2.11. *If $a \in \mathbb{Z}[\theta]$ and $a \neq 0$, then:*

$$\mathrm{T}(a^{-1}) = O\left(m^3 \operatorname{Mul}(m\,\mathrm{s}(a) + m^2 F)\right). \tag{2.26}$$

Proof. We follow the ideas of [5, Prop. 10], though using the more general Mul function and the more precise $O(\cdot)$ notation. The calculation of $a^{-1} = b/n$ goes as follows:

1. Calculate a matrix M_a from a as described above.
2. Solve the linear system of equations $M_a(b_0/n, b_1/n, \ldots, b_{m-1}/n)^T = (1, 0, \ldots, 0)$ for b/n.

The latter can be done e.g. by the Bareiss algorithm [6], which runs in the following time, if A is a bound on the matrix entries:

$$\mathrm{T}(\text{Bareiss}) = O\left(m^3 \operatorname{Mul}(m \log(mA))\right).$$

From the formula of M_a (2.24), we can get that:

$$\log A = \log \max_{i,j} |(M_a)_{i,j}| = O(\mathrm{s}(a) + mF).$$

Substituting this already gives (2.26), and $\mathrm{T}(M_a)$ (2.25) has lower order. □

As a consequence, we can also calculate the running time of division:

Lemma 2.12. *If $a, b \in \mathbb{Z}[\theta]$ and $b \neq 0$, then:*

$$\mathrm{T}\left(\frac{a}{b}\right) = O\left(m^3 \operatorname{Mul}(\mathrm{s}(a) + m\,\mathrm{s}(b) + m^2 F)\right). \tag{2.27}$$

Proof. If $b^{-1} = \frac{c}{n}$ with $n \in \mathbb{Z}$, then $\frac{a}{b} = \frac{ac}{n}$. By (2.16), $\mathrm{s}(c) = O\left(m\,\mathrm{s}(b) + m^2 F\right)$, and then $\mathrm{T}(b^{-1})$ by (2.26) and $\mathrm{T}(ac)$ by (2.23) gives the statement. □

In the rest of this section, we discuss some operations special to $\mathbb{R}$. Therefore, from now on we assume that $\theta \in \mathbb{R}$ and so $K \subset \mathbb{R}$.

First, we need the following auxiliary statement.

Lemma 2.13. *Let $a, b \in \mathbb{Z}[\theta] \setminus \{0\}$. Let $\tilde{\theta} \in \mathbb{R}$ be so close to θ that*

$$\log |\tilde{\theta} - \theta|^{-1} \geq \mathrm{s}(a) + (m-1)\,\mathrm{s}(b) + \frac{3}{2} m(m+1)(F+1), \tag{2.28}$$

and let $\tilde{a} = a_0 + a_1\tilde{\theta} + a_2\tilde{\theta}^2 + \ldots + a_{m-1}\tilde{\theta}^{m-1}$. Then:

$$|\tilde{a} - a| < |b|.$$

Proof. First we use only that $m|\tilde{\theta}-\theta| < \frac{1}{2}$ from (2.28).

$$|\tilde{a}-a| = \left|\sum_{k=0}^{m-1} a_k \left(\left(\theta+(\tilde{\theta}-\theta)\right)^k - \theta^k\right)\right| = \left|\sum_{k=0}^{m-1} a_k \sum_{j=1}^{k} \binom{k}{j} \theta^{k-j}(\tilde{\theta}-\theta)^j\right| \leq$$

$$\leq \sum_{k=0}^{m-1} |a_k| \sum_{j=1}^{k} \binom{k}{j} |\theta|^{k-j}|\tilde{\theta}-\theta|^j = \sum_{j=1}^{m-1}\sum_{l=0}^{m-j-1} |a_{l+j}| \binom{l+j}{j} |\theta|^l |\tilde{\theta}-\theta|^j \leq$$

$$\leq \left(\max_{i=0}^{m-1} |a_i|\right) \sum_{j=1}^{m-1} m^j |\tilde{\theta}-\theta|^j \sum_{l=0}^{m-j-1} |\theta|^l < \left(\max_{i=0}^{m-1} |a_i|\right) 2m|\tilde{\theta}-\theta| \sum_{l=0}^{m-1} |\theta|^l.$$

Taking logarithm and using (2.4), we get:

$$\log|\tilde{a}-a| < \mathrm{s}(a) + \log 2m|\tilde{\theta}-\theta| + mF \leq \mathrm{s}(a) + \frac{3}{2}m(F+1) + \log|\tilde{\theta}-\theta|.$$

We add this inequality to (2.19) for b:

$$\log|\tilde{a}-a| + \log|b^{-1}| < \mathrm{s}(a) + (m-1)\,\mathrm{s}(b) + \frac{3}{2}m(m+1)(F+1) + \log|\tilde{\theta}-\theta|.$$

By (2.28), the right-hand-side is ≤ 0, so the left-hand-side is < 0, which is equivalent to the statement. □

Now we are able to calculate the worst-case complexity of the following operations.

Lemma 2.14. *If $a, b \in \mathbb{Z}[\theta] \subset \mathbb{R}$, then:*

$$\mathrm{T}(a<b) = \mathrm{T}(a\leq b) = O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a,b)+m^2F)\right), \tag{2.29}$$

Proof. Since $a < b$ is equivalent to $a - b < 0$, we need to consider only the $a < 0$ comparison, and the same is true for $\leq$. Assume the nontrivial case $a \neq 0$.

We approximate θ by a rational number $\tilde{\theta} = \frac{u}{d}$ where $d \in \mathbb{Z}^+$ and u is either $\lfloor \theta d \rfloor$ or $\lceil \theta d \rceil$, the one with the smaller absolute value. We need so large d, hence so close $\tilde{\theta}$ to θ, that the approximation $\tilde{a} = a_0 + a_1\tilde{\theta} + a_2\tilde{\theta}^2 + \ldots + a_{m-1}\tilde{\theta}^{m-1}$ and the exact a have the same sign. This can be guaranteed if $|\tilde{a}-a| < |a|$, because then

$$a > 0 \implies \tilde{a} \geq a - |\tilde{a}-a| > 0,$$
$$a < 0 \implies \tilde{a} \leq a + |\tilde{a}-a| < 0.$$

Now by Lemma 2.13 with $b = a$, we can ensure this by choosing a d with

$$\log d \geq m\,\mathrm{s}(a) + \frac{3}{2}m(m+1)(F+1),$$

because $\log|\tilde{\theta}-\theta|^{-1} > \log d$. We can choose d to be the smallest such integer, but it is more efficient to round up to the nearest power of two. In any case,

$$\log d = O\left(m\,\mathrm{s}(a) + m^2F\right). \tag{2.30}$$

Now the approximation can be written as:

$$\tilde{a} = a_0 + a_1\frac{u}{d} + \ldots + a_{m-1}\left(\frac{u}{d}\right)^{m-1} = \frac{a_0 d^{m-1} + a_1 u d^{m-2} + \ldots + a_{m-1}u^{m-1}}{d^{m-1}}.$$

Call the numerator r. It can be calculated by the following recursion:

$$r_0 := 0, \qquad r_{k+1} := r_k u + a_{m-k-1}d^k, \qquad r := r_m.$$

Since d is a power of two, the dominating operation is the multiplication $r_k u$.

We can bound u and r_k as follows, the latter by induction:

$$|u| = \min(|\lfloor \theta d \rfloor|, |\lceil \theta d \rceil|) \le |\theta d| = |\theta| d,$$
$$|r_k| \le \left(\max_{i=0}^{m-1} |a_i| \right) d^{k-1} \left(1 + |\theta| + \ldots + |\theta|^{k-1} \right).$$

Therefore, using (2.4) and (2.30):

$$\begin{aligned} \mathrm{T}(r_k u) &= \mathrm{Mul}(\log |r_k|, \log |u|) \le \mathrm{Mul}(\mathrm{s}(a) + m \log d + mF, F + \log d) = \\ &= O(\mathrm{Mul}(m^2\,\mathrm{s}(a) + m^3 F, m\,\mathrm{s}(a) + m^2 F)) = O(m\,\mathrm{Mul}(m\,\mathrm{s}(a) + m^2 F)), \end{aligned}$$

and the whole calculation performs m such multiplications. □

Lemma 2.15. *If $a, b \in \mathbb{Z}[\theta] \subset \mathbb{R}$, $n \in \mathbb{Z}$ and $b \neq 0$, $n \neq 0$, then:*

$$\mathrm{T}\left(\left\lfloor \frac{a}{n} \right\rfloor\right) = \mathrm{T}\left(\left\lceil \frac{a}{n} \right\rceil\right) = \mathrm{T}\left(\left\lfloor \frac{a}{n} \right\rceil\right) = O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a,n) + m^2 F)\right), \tag{2.31}$$

$$\mathrm{T}\left(\left\lfloor \frac{a}{b} \right\rfloor\right) = \mathrm{T}\left(\left\lceil \frac{a}{b} \right\rceil\right) = \mathrm{T}\left(\left\lfloor \frac{a}{b} \right\rceil\right) = O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a) + m^2\,\mathrm{s}(b) + m^3 F)\right). \tag{2.32}$$

Proof. Since $\lfloor a/n \rceil = \lfloor a/n + 1/2 \rfloor$, we need to consider only $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$. If $a \in \mathbb{Z}$, we can use integer division in $O(\mathrm{Mul}(\mathrm{s}(a), \mathrm{s}(n)))$ time, so assume that $a \notin \mathbb{Z}$.

The proof of (2.31) goes similarly to that of Lemma 2.14. Here the denominator d must be so large that $\tilde{a}/n$ and a/n have the same integer part. This can be ensured by

$$\left| \frac{\tilde{a}}{n} - \frac{a}{n} \right| < \min\left(\left| \frac{a}{n} - \left\lfloor \frac{a}{n} \right\rfloor \right|, \left| \frac{a}{n} - \left\lceil \frac{a}{n} \right\rceil \right| \right).$$

After multiplying this by $|n|$, we can use Lemma 2.13 with $b \in \{a - n\lfloor a/n \rfloor,\ a - n\lceil a/n \rceil\}$, but first we need to calculate $\mathrm{s}(b)$. Since $b - a$ is an integer, the coefficients of a and b are the same except the constant term, so $\mathrm{s}(b) \le \max(\mathrm{s}(a), \log |b_0|)$. We can bound $|b_0|$ as follows (note that $|b/n| < 1$, so $|b| < |n|$):

$$|b_0| = |a_0 - a + b| < |a - a_0| + |n| = \left| \sum_{k=1}^{m-1} a_k \theta^k \right| + |n| \le \max\left(|n|, \max_{i=1}^{m-1} |a_i| \right) \left(\sum_{i=0}^{m-1} |\theta|^i \right),$$

whose logarithm is bounded by $\mathrm{s}(a, n) + mF$, so:

$$\mathrm{s}(b) \le \mathrm{s}(a, n) + mF.$$

Now by Lemma 2.13, we can choose d such that

$$\log d = O\left(m\,\mathrm{s}(a, n) + m^2 F\right).$$

We can calculate in the same way as in Lemma 2.14 that for r, the numerator of $\tilde{a}$:

$$\begin{aligned} \mathrm{T}(r) &= O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a, n) + m^2 F)\right), \\ |r| &= O\left(m^2\,\mathrm{s}(a, n) + m^3 F\right). \end{aligned}$$

The last step is to divide it by n. This is an integer division, and takes $\mathrm{Mul}(m^2\,\mathrm{s}(a, n) + m^3 F, \mathrm{s}(n))$ time. Adding this to $\mathrm{T}(r)$ we get (2.31).

For (2.32), first we calculate a/b in the form ac/n with $n \in \mathbb{Z}$. This takes $\mathrm{T}(a/b) = O\left(m^3\,\mathrm{Mul}(\mathrm{s}(a) + m\,\mathrm{s}(b) + m^2 F)\right)$ time by Lemma 2.12. The length of the parameters are, by (2.16), (2.15) and (2.17):

$$\begin{aligned} \mathrm{s}(c) &= O\left(m\,\mathrm{s}(b) + m^2 F\right), \\ \mathrm{s}(ac) &= O\left(\mathrm{s}(a) + \mathrm{s}(c) + mF\right) = O\left(\mathrm{s}(a) + m\,\mathrm{s}(b) + m^2 F\right), \\ \log |n| &= O\left(m\,\mathrm{s}(b) + m^2 F\right). \end{aligned}$$

Putting these into (2.31) and adding $\mathrm{T}(a/b)$ gives the result. □

2.3 Summary of the operations

The following table summarizes the results of Section 2 on the time complexity and the size of the results of the operations in $\mathbb{Z}[\theta]$. In this table, $a, b, c \in \mathbb{Z}[\theta]$ and $n \in \mathbb{Z}$.

Operation	Output size	Time
$a \pm b$	$\mathrm{s}(a,b) + \log 2$	$O\left(m\,\mathrm{s}(a,b)\right)$
na	$\mathrm{s}(a) + \mathrm{s}(n)$	$O\left(m\,\mathrm{Mul}(\mathrm{s}(n), \mathrm{s}(a))\right)$
ab	$O\left(\mathrm{s}(a) + \mathrm{s}(b) + mF\right)$	$O\left(m^2\,\mathrm{Mul}(\mathrm{s}(a), \mathrm{s}(b)) + m^2\,\mathrm{Mul}(mF,\, \mathrm{s}(a) + \mathrm{s}(b) + \log m)\right)$
$a^{-1} \to \frac{b}{n}$	$O\left(m\,\mathrm{s}(a) + m^2F\right)$	$O\left(m^3\,\mathrm{Mul}(m\,\mathrm{s}(a) + m^2F)\right)$
$\frac{a}{b} \to \frac{c}{n}$	$O\left(\mathrm{s}(a) + m\,\mathrm{s}(b) + m^2F\right)$	$O\left(m^3\,\mathrm{Mul}(\mathrm{s}(a) + m\,\mathrm{s}(b) + m^2F)\right)$
$a < b$, $a \le b$	$O\left(1\right)$	$O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a,b) + m^2F)\right)$
$\left\lfloor \frac{a}{n} \right\rfloor$, $\left\lceil \frac{a}{n} \right\rceil$, $\left\lfloor \frac{a}{n} \right\rceil$	$O\left(\mathrm{s}(a) + mF\right)$	$O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a,n) + m^2F)\right)$
$\left\lfloor \frac{a}{b} \right\rfloor$, $\left\lceil \frac{a}{b} \right\rceil$, $\left\lfloor \frac{a}{b} \right\rceil$	$O\left(s(a) + ms(b) + m^2F\right)$	$O\left(m^2\,\mathrm{Mul}(m\,\mathrm{s}(a) + m^2\,\mathrm{s}(b) + m^3F)\right)$

3 Bareiss algorithm

The Bareiss algorithm [6] is an integer-preserving modification of Gaussian elimination for $\mathbb{Z}$ that maintains as small integers as generally possible by using provably exact divisions to reduce their sizes. In this section we apply the algorithm to $\mathbb{Z}[\theta]$ (where θ is not neccessarily real) and calculate its running time using the results of the previous section, and compare it with the running time in $\mathbb{Z}$. We consider the simpliest form of the algorithm, when a square matrix is converted into an upper triangular form (e.g. to calculate its determinant).

Let $M \in \mathbb{Z}[\theta]^{n\times n}$ with elements a_{ij}, and let $A := \max_{i,j} \mathrm{s}(a_{ij})$. The Bareiss algorithm uses the following formula [6, p. 570]:

$$a_{00}^{(0)} = 1, \quad a_{ij}^{(1)} = a_{ij}, \qquad a_{ij}^{(k+1)} = \frac{a_{kk}^{(k)} a_{ij}^{(k)} - a_{ik}^{(k)} a_{kj}^{(k)}}{a_{k-1,k-1}^{(k-1)}} \tag{3.1}$$

with $1 \le k \le n-1$ and $k+1 \le i, j \le n$.

First we bound the size of all intermediate variables, $a_{ij}^{(k)}$. They can be written as determinants of order k ($\le n$) whose entries are from M [6, p. 565]. Therefore, the division in (3.1) is exact and $a_{ij}^{(k)} \in \mathbb{Z}[\theta]$, so we can use $\mathrm{s}(\cdot)$ to measure their size. By Lemma 2.8, we have:

$$B := \max_{i,j,k} \mathrm{s}\left(a_{ij}^{(k)}\right) = O(nA + nmF + n\log n). \tag{3.2}$$

Now we estimate the running time of the recursive formula (3.1). The calculation consists of the following main operations:

1. two multiplications:
   - time: $O\left(m^2\,\mathrm{Mul}(B)\right)$ by (2.23),
   - output size: $O(B)$ by (2.15);
2. exact division:
   (a) calculating $\left(a_{k-1,k-1}^{(k-1)}\right)^{-1}$ in the form $b_{k-1,k-1}^{(k-1)} / d_{k-1,k-1}^{(k-1)}$:
      - time: $O\left(m^3\,\mathrm{Mul}(mB)\right)$ by (2.26),

- output size: $O(mB)$ by (2.16) and (2.17);

(b) multiplying the numerator by $b_{k-1,k-1}^{(k-1)}$:
- time: $O\left(m^2 \operatorname{Mul}(B, mB)\right) = O\left(m^3 \operatorname{Mul}(B)\right)$ by (2.23),
- output size: $O(mB)$ by (2.15);

(c) and dividing the resulting algebraic number exactly by the integer $d_{k-1,k-1}^{(k-1)}$:
- time: $O\left(m \operatorname{Mul}(mB)\right)$.

Most of these are done for each i, j, k, i.e. $O(n^3)$ times, but 2/(a) depends only on k, so it is done $O(n)$ times. Adding all these together, we have:

$$\mathrm{T}_{\mathbb{Z}[\theta]}(\text{Bareiss}) = O\left(n^3 m^3 \operatorname{Mul}(B) + n m^3 \operatorname{Mul}(mB)\right).$$

For comparison, the running time over $\mathbb{Z}$ is:

$$\mathrm{T}_{\mathbb{Z}}(\text{Bareiss}) = O\left(n^3 \operatorname{Mul}(nA + n \log n)\right).$$

We can see that if we ignore the field-dependent constants m and F, the running time is asymptotically the same. In the following table, we compare several different $\operatorname{Mul}(\cdot)$ functions (see Section 2.2) for both cases. In the last row, we introduced a simplified notation $\tilde{O}(\ldots)$ for omitting the logarithmic factors, i.e. $\tilde{O}(N) = O(N \log N \log\log N)$.

$\operatorname{Mul}(X)$	$\mathrm{T}_{\mathbb{Z}[\theta]}(\text{Bareiss})$	$\mathrm{T}_{\mathbb{Z}}(\text{Bareiss})$
X^2	$O\left(n^3 m^3 (n^2 + m^2)(A + mF + \log n)^2\right)$	$O\left(n^5 (A + \log n)^2\right)$
$X^{\log_2 3}$	$O\left(n^{2.6} m^3 (n^2 + m^{1.6})(A + mF + \log n)^{1.6}\right)$	$O\left(n^{4.6} (A + \log n)^{1.6}\right)$
$X \log X \log\log X$	$\tilde{O}\left(n^2 m^3 (n^2 + m)(A + mF + \log n)\right)$	$\tilde{O}\left(n^4 (A + \log n)\right)$

4 LLL algorithm

The LLL algorithm is a lattice basis reduction algorithm invented by Lenstra, Lenstra and Lovász [7]. It is known that it runs in polynomial time if the vectors are in $\mathbb{Z}^n$. In this section we show that it is also polynomial for vectors in $\mathbb{Z}[\theta]^n$ for real θ.

Let $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$ be a basis. Then

$$\Lambda(b_1, b_2, \ldots, b_n) := \left\{ c_1 b_1 + c_2 b_2 + \ldots + c_n b_n \;\middle|\; c_1, c_2, \ldots, c_n \in \mathbb{Z} \right\}$$

is called the *lattice* spanned by $b_1, b_2, \ldots, b_n$.

The LLL algorithm modifies $b_1, b_2, \ldots, b_n$ step by step, preserving the spanned lattice, and finally turning the vectors to a reduced basis (in the sense defined below). At any point in the algorithm, we define the Gram–Schmidt orthogonalization of the actual b_i vectors as follows:

$$b_i^* := b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \qquad (1 \le i \le n), \tag{4.1}$$

$$\mu_{ij} := \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \qquad (1 \le j < i \le n). \tag{4.2}$$

When the algorithm terminates, the b_i vectors are LLL-reduced, which means the following two properties:

$$|\mu_{ij}| \le \frac{1}{2} \qquad (1 \le j < i \le n), \tag{4.3}$$

$$\left\| b_i^* + \mu_{i\,i-1} b_{i-1}^* \right\|_2^2 \ge \delta \left\| b_{i-1}^* \right\|_2^2 \qquad (2 \le i \le n), \tag{4.4}$$

where δ is a parameter of the algorithm between $\frac{1}{4} < \delta < 1$, often $\delta = \frac{3}{4}$.

The skeleton of the LLL algorithm is the following. This contains only the changes of b_i's. The full algorithm keeps track of other variables after each b_i-change to maintain (4.1) and (4.2).

$k := 2$
while $k \leq n$ **do**
  $b_k := b_k - \lfloor \mu_{k\,k-1} \rceil \, b_{k-1}$
  if $k \geq 2 \wedge \left\| b_k^* + \mu_{k\,k-1} b_{k-1}^* \right\|_2^2 < \delta \left\| b_{k-1}^* \right\|_2^2$ **then**
    (swap step)
    $b_k \leftrightarrow b_{k-1}$
    $k := k - 1$
  else
    (reduction step)
    for $l := k - 2$ **to** 1 **do**
      $b_k := b_k - \lfloor \mu_{kl} \rceil \, b_l$
    $k := k + 1$

4.1 Properties in $\mathbb{R}^n$

First we discuss some properties of the LLL algorithm on any real basis $b_1, b_2, \ldots, b_n \in \mathbb{R}^n$. We start with some definitions.

Given the lattice $\Lambda := \Lambda(b_1, b_2, \ldots, b_n)$, we need the shortest vector length in Λ:

$$L_0 := \min \left\{ \|x\|_2^2 \;\middle|\; x \in \Lambda \setminus \{0\} \right\}.$$

Other important quantities are the numbers $d_1, d_2, \ldots, d_n \in \mathbb{R}$, which depend on the b_i vectors, and can be defined by any of the following equivalent expressions [7, p. 521]:

$$d_l = \|b_1^*\|_2^2 \, \|b_2^*\|_2^2 \ldots \|b_l^*\|_2^2, \tag{4.5}$$

$$d_l = \det \left(\langle b_i, b_j \rangle \right)_{1 \leq i,j \leq l}, \tag{4.6}$$

$$d_l = \det(\Lambda(b_1, b_2, \ldots, b_l))^2. \tag{4.7}$$

For convenience, we extend this to $d_0 := 1$.

We will need the following inequalities between these quantities, which are independent from the context of the LLL algorithm.

Lemma 4.1.

$$d_l \geq \left(\frac{L_0}{l} \right)^l, \qquad (1 \leq l \leq n) \tag{4.8}$$

$$\min_{i=1}^{n} \|b_i^*\|_2^2 \leq L_0 \leq \|b_1^*\|_2^2. \tag{4.9}$$

Proof. Minkowski's theorem [15, III.2.2.] states that if $S \subseteq \mathbb{R}^l$ is convex, symmetric to the origin, and has no other common point with the $\Lambda_l := \Lambda(b_1, b_2, \ldots, b_l)$ lattice than the origin, then:

$$\det(\Lambda_l) \geq 2^{-l} \operatorname{Vol}(S),$$

where $\mathrm{Vol}(S)$ volume of S. Applying this to a hypercube with side $2r/\sqrt{l}$ with $r < \sqrt{L_0}$ (note that its circumscribed sphere has radius r), then by $r \to \sqrt{L_0}$ and squaring, we get (4.8).

The upper bound on L_0 in (4.9) is a consequence of this, since $d_1 = \|b_1^*\|_2^2$. The lower bound on L_0 is elementary, and follows e.g. from the proof of [7, (1.11)]. □

The following lemma shows that after any whole number of iterations in the LLL algorithm, the variables and related quantities can be bounded by initially known expressions.

Lemma 4.2. *Let $B := \max_{i=1}^n \|b_i\|_2^2$ for the input vectors in the LLL algorithm. Then, at the beginning or end of the body of the main* **while***-loop, the following inequalities hold, depending on the loop variable k:*

$$\|b_i^*\|_2^2 \le B, \tag{4.10}$$

$$\|b_i\|_2^2 \le nB \qquad (i \ne k), \tag{4.11}$$

$$|\mu_{ij}| \le \frac{1}{2} \qquad (i < k), \tag{4.12}$$

$$|\mu_{ij}| \le 2^{n-i}\sqrt{n}\left(\frac{nB}{L_0}\right)^{\frac{n-1}{2}} \qquad (i = k), \tag{4.13}$$

$$|\mu_{ij}| \le \sqrt{n}\left(\frac{jB}{L_0}\right)^{\frac{j}{2}} \qquad (i > k), \tag{4.14}$$

$$d_j \le B^j. \tag{4.15}$$

Proof. These inequalities are analogous to [7, p. 523], with the difference that [7] works in $\mathbb{Z}^n$, so it can use the fact that $d_l \ge 1$ since d_l is both integer and positive. We replace this by the more general (4.8).

(4.10), (4.11), (4.12) and (4.15) are the same as in [7].

We prove (4.14) by using our other inequalities and the Cauchy–Schwarz inequality:

$$|\mu_{ij}|^2 \overset{(4.2)}{=} \frac{\left|\langle b_i, b_j^*\rangle\right|^2}{\|b_j^*\|_2^4} \overset{\text{C.-S.}}{\le} \frac{\|b_i\|_2^2\|b_j^*\|_2^2}{\|b_j^*\|_2^4} \overset{(4.5)}{=} \frac{d_{j-1}}{d_j}\|b_i\|_2^2 \overset{\substack{(4.15)\\(4.8)}}{\le} \frac{B^{j-1}}{\left(\frac{L_0}{j}\right)^j}\|b_i\|_2^2 \overset{(4.11)}{\le} n\left(\frac{jB}{L_0}\right)^j.$$

Then the derivation of (4.13) from (4.14) works in the same way as the analogue [7, (1.34)]. □

Now we can give an estimate on the number of iterations in the LLL algorithm.

Lemma 4.3. *Let N be the number of main iterations (both reduction steps and swap steps), and let $K_\delta := \frac{1}{\log\frac{1}{\delta}}$. Then:*

$$N = O\left(n^2 \log\frac{nB}{L_0} K_\delta\right).$$

Proof. Let N_r be the number of reduction steps, and N_s be the number of swap steps. Since a reduction adds, and a swap subtracts 1 from k, and since the algorithm starts with $k = 2$ and finishes when $k = n+1$, therefore $N_r - N_s = n-1$, so $N = N_r + N_s = 2N_s + n - 1$, i.e. it is sufficient to estimate N_s.

Let $D := d_1 d_2 \dots d_n$, and let $D^{(s)}$ be the value of D after s swap steps.

[7] proves that for integer values (i.e. for $b_1, \ldots, b_n \in \mathbb{Z}^n$) there are at most $O(n^2 \log B)$ iterations (or rather $O(n^2 \log B\, K_\delta)$ if we want to capture δ), but this uses the fact that D is an integer, hence $D \geq 1$. We replace this with another lower bound for D. We also need an upper bound for D:

$$D = \prod_{j=1}^{n} d_j \overset{(4.8)}{\geq} \prod_{j=1}^{n} \left(\frac{L_0}{j}\right)^j \geq \prod_{j=1}^{n} \left(\frac{L_0}{n}\right)^j = \left(\frac{L_0}{n}\right)^{\frac{n(n+1)}{2}},$$

$$D = \prod_{j=1}^{n} d_j \overset{(4.15)}{\leq} \prod_{j=1}^{n} B^j = B^{\frac{n(n+1)}{2}}.$$

These bounds are true after any number of iterations, i.e. for any $D^{(s)}$. Furthermore, we use the fact that a reduction step does not change D, and that a swap step reduces D by a factor $< \delta$: $D^{(s+1)} < \delta D^{(s)}$ – both are proved in [7, p. 521] without the use of the integer property. By induction, it follows that $D^{(s)} < \delta^s D^{(0)}$. Putting these inequalities together:

$$\left(\frac{L_0}{n}\right)^{\frac{n(n+1)}{2}} \leq D^{(N_s)} < \delta^{N_s} D^{(0)} \leq \delta^{N_s} B^{\frac{n(n+1)}{2}}.$$

After taking logarithms from both ends and rearranging, we get:

$$N_s < \frac{1}{\log \frac{1}{\delta}} \frac{n(n+1)}{2} \log \frac{nB}{L_0},$$

and the statement follows from this, because $N = 2N_s + n - 1$. □

4.2 Coefficient size in $\mathbb{Z}[\theta]^n$

Now we restrict the basis to be over a real number field, i.e. from now on, $b_1, b_2, \ldots, b_n \in \mathbb{Z}[\theta]^n \subset \mathbb{R}^n$. We use the notations m, F and $\mathrm{s}(\cdot)$ as in Section 2, and extend $\mathrm{s}(\cdot)$ naturally to vectors: $\mathrm{s}(x) := \max_{j=1}^{n} \mathrm{s}(x_j)$.

When implementing the LLL algorithm exactly, we do not need to maintain the non-integral quantities b_i^* and μ_{ij}. Instead, as presented e.g. in [1, Alg. 2.6.7] for $\mathbb{Z}$, we can use the integer d_j, and write $\mu_{ij} = \lambda_{ij}/d_j$ where λ_{ij} is also an integer (see e.g. [1, Prop. 2.6.5]). The same applies to $\mathbb{Z}[\theta]$, i.e. $d_j, \lambda_{ij} \in \mathbb{Z}[\theta]$.

We need to give bounds on $\mathrm{s}(d_j)$ and $\mathrm{s}(\lambda_{ij})$ during the algorithm. First we do this in terms of the current $\mathrm{s}(b_i)$, outside of the context of the LLL algorithm.

Lemma 4.4. *For the corresponding values of b_i, d_j and λ_{ij}, we have:*

$$\mathrm{s}\left(d_j, \lambda_{ij}\right) \leq 2n \left(\max_{i=1}^{n} \mathrm{s}(b_i) + (m-1)F + 2\log m + \log n\right).$$

Proof. We already know from (4.6) that d_j is a $j \times j$ determinant of elements like $\langle b_{i'}, b_{j'} \rangle$. We show that λ_{ij} has a similar structure. This follows e.g. from the proof of [1, Prop. 2.6.5] showing that λ_{ij} is integer, where we have

$$\begin{pmatrix} \langle b_1, b_1 \rangle & \cdots & \langle b_1, b_j \rangle \\ \vdots & \ddots & \vdots \\ \langle b_j, b_1 \rangle & \cdots & \langle b_j, b_j \rangle \end{pmatrix} \begin{pmatrix} \xi_1 \\ \vdots \\ \xi_j \end{pmatrix} = \begin{pmatrix} \langle b_1, b_i \rangle \\ \vdots \\ \langle b_j, b_i \rangle \end{pmatrix}$$

with $\xi_j = \mu_{ij}$. Solving the system for $\mu_{ij} = \lambda_{ij}/d_j$ by Cramer's rule gives the needed determinant structure for both λ_{ij} and d_j.

First we give an estimate on the coefficient size of the individual $\langle b_{i'}, b_{j'} \rangle$ elements using the properties of the $\mathrm{s}(\cdot)$ operator:

$$\mathrm{s}\left(\langle b_{i'}, b_{j'} \rangle\right) \leq \mathrm{s}(b_{i'}) + \mathrm{s}(b_{j'}) + (m-1)F + 2\log m + \log n.$$

Then applying Lemma 2.8 to the $j \times j$ determinants (using that $j \leq n$) finishes the proof. □

In Section 4.1 about the LLL algorithm over $\mathbb{R}$, several inequalities used the constant L_0, mostly in the form $\frac{nB}{L_0}$. In $\mathbb{Z}[\theta]$, we can get rid of L_0, and use only basic parameters, including the coefficient size of the input vectors:

$$A := \max_{i=1}^{n} \mathrm{s}(b_i). \tag{4.16}$$

Lemma 4.5. *Consider the LLL algorithm over $\mathbb{Z}[\theta]$, and let*

$$H := \frac{1}{n} \log \frac{nB}{L_0}, \tag{4.17}$$

then we have:

$$H = O\left(mA + m^2F + m\log n\right). \tag{4.18}$$

Proof. We can give the following lower bound on L_0 in terms of the initial d_l's:

$$\frac{1}{L_0} \overset{(4.9)}{\leq} \max_{i=1}^{n} \frac{1}{\|b_i^*\|_2^2} \overset{(4.5)}{=} \max_{i=1}^{n} \frac{d_{i-1}}{d_i},$$

and use Lemma 2.7 to bound d_l from two sides by $\mathrm{s}(d_l)$, and use Lemma 4.4 about $\mathrm{s}(d_l)$:

$$\log \frac{d_{i-1}}{d_i} = \log d_{i-1} + \log d_i^{-1} < m \max_{j=0}^{n} \mathrm{s}(d_j) + O(m^2F) = O\left(nmA + nm^2F + nm\log n\right).$$

It remains to give an upper bound on B in terms of A, again by Lemma 2.7:

$$\log B = \log \max_{i=1}^{n} \|b_i\|_2^2 \leq 2\log \max_{i=1}^{n} \|b_i\|_\infty + \log n < 2A + 2mF + \log n,$$

and we get the statement by combining these results. □

Lemma 4.6. *Consider one main step of the LLL algorithm, either a reduction step or a swap step. If the vectors before and after the step are called b_i and b_i' respectively, then:*

$$\mathrm{s}(b_i') \leq \max_{l=1}^{n} \mathrm{s}(b_l) + \frac{n^2}{2}H + n\log 2.$$

Proof. The following pseudocode shows the changes made to b_i and μ_{ij} in a single reduction step. (Note that compared to the full algorithm presented earlier, here the μ_{kj}'s are also present, and all changes of b_k are joined into the l-loop. Also note that this is presented for the sake of the calculation, but we use the integral version of the algorithm instead, as described in Section 4.3, which manipulates λ_{ij} and d_j instead of the rational μ_{ij}.)

for $l := k-1$ **to** 1 **do**
  $b_k := b_k - \lfloor \mu_{kl} \rceil b_l$
  for $j := 1$ **to** $l-1$ **do**
    $\mu_{kj} := \mu_{kj} - \lfloor \mu_{kl} \rceil \mu_{lj}$
  $\mu_{kl} := \mu_{kl} - \lfloor \mu_{kl} \rceil$

A swap step makes fewer of these changes (only for $l = k-1$), and additionally an exchange of two b_i's, which does not change the maximum of $\mathrm{s}(b_i)$. Therefore, we can concentrate on the reduction step only.

None of the b_i or μ_{ij} change for $i \neq k$, so the statement trivially holds for these b_i. In order to calculate the change of $\mathrm{s}(b_k)$, we need the size of $\lfloor \mu_{kl} \rceil$ in the algorithm. To distinguish between different values of the variable, we call μ_{kl} the initial value, and μ'_{kl} the value when taking $\lfloor \cdot \rceil$. Examining the code above, we can see that

$$\mu'_{kl} = \mu_{kl} - \sum_{i=l+1}^{k-1} \lfloor \mu'_{ki} \rceil \mu_{il},$$

so, using the bounds on μ_{ij}, (4.12) and (4.13):

$$|\lfloor \mu'_{kl} \rceil| \leq 2|\mu'_{kl}| \leq 2|\mu_{kl}| + 2 \sum_{i=l+1}^{k-1} |\lfloor \mu'_{ki} \rceil| \, |\mu_{il}| \leq 2^{n-k+1} \sqrt{n} \left(\frac{nB}{L_0} \right)^{\frac{n-1}{2}} + \sum_{i=l+1}^{k-1} |\lfloor \mu'_{ki} \rceil| \,.$$

From this, we can show by induction from $l = k-1$ to 1 that

$$|\lfloor \mu'_{kl} \rceil| \leq 2^{n-l} \sqrt{n} \left(\frac{nB}{L_0} \right)^{\frac{n-1}{2}}.$$

Now we can calculate the change of $\mathrm{s}(b_k)$:

$$\mathrm{s}(b'_k) = \mathrm{s}\left(b_k - \sum_{l=1}^{k-1} \lfloor \mu'_{kl} \rceil b_l \right) \leq \max_{l=1}^{k} \mathrm{s}(b_l) + \log \left(1 + \sum_{l=1}^{k-1} \left| \lfloor \mu'_{kl} \rceil \right| \right).$$

We give an upper bound on the argument of this logarithm using our bound on $\lfloor \mu'_{kl} \rceil$:

$$1 + \sum_{l=1}^{k-1} \left| \lfloor \mu'_{kl} \rceil \right| \leq 1 + \sum_{l=1}^{k-1} 2^{n-l} \sqrt{n} \left(\frac{nB}{L_0} \right)^{\frac{n-1}{2}} =$$

$$= 1 + \left(2^n - 2^{n-k+1} \right) n^{\frac{1}{2}} \left(\frac{nB}{L_0} \right)^{\frac{n-1}{2}} \leq 2^n \left(\frac{nB}{L_0} \right)^{\frac{n}{2}}.$$

In the last step, we used that $\frac{nB}{L_0} \geq n$, which follows from $L_0 \leq B$ by (4.9) and (4.10). Since the logarithm of the right-hand-side is $\frac{n^2}{2} H + n \log 2$ (see the definition of H: (4.17)), the proof is completed. □

Now we can combine all these results together to get the $\mathrm{s}(\cdot)$ of the main variables during the LLL algorithm.

Lemma 4.7. *In the LLL algorithm, at the beginning or end of the body of the main* **while***-loop, the following inequalities hold:*

$$\mathrm{s}(b_i) = O\left(n^5 H^2 K_\delta \right), \tag{4.19}$$

$$\mathrm{s}(d_j) = O\left(n^6 H^2 K_\delta \right), \tag{4.20}$$

$$\mathrm{s}(\lambda_{ij}) = O\left(n^6 H^2 K_\delta \right). \tag{4.21}$$

Proof. If we repeatedly apply Lemma 4.6 for the first t steps of the LLL algorithm, then:

$$\mathrm{s}(b_i) \leq A + O\left(n^2 H \right) t,$$

and because $t = O\left(n^3 H K_\delta \right)$ by Lemma 4.3 and H dominates A by Lemma 4.5, we get (4.19). The other two follows from this by Lemma 4.4. □

4.3 Running time of the LLL algorithm

Now we have enough information to calculate an upper bound for the running time of the LLL algorithm for algebraic numbers. The basic structure of the algorithm is

presented at the beginning of Section 4, and for the details, we use [1, Alg. 2.6.7], but adapted to $\mathbb{Z}[\theta]$ instead of $\mathbb{Z}$. The algorithm has three parts where significant operations take place:

1. reduction with a single μ_{kl}, i.e. the assignment $b_k := b_k - \lfloor \mu_{kl} \rceil b_l$ and related neccessary changes (note that this is not a complete reduction step, which does this $k-1$ times);
2. exchange of b_k and b_{k-1} and the related neccessary changes;
3. the comparison $\left\| b_k^* + \mu_{k\,k-1} b_{k-1}^* \right\|_2^2 < \delta \left\| b_{k-1}^* \right\|_2^2$.

We denote the running time of these steps as T(red), T(swap) and T(cmp) respectively.

Let D be the bound on all $\mathrm{s}(d_j)$, $\mathrm{s}(\lambda_{ij})$ and $\mathrm{s}(b_i)$ after any number of iterations, and we know by Lemma 4.7 that $D = O\left(n^6 H^2 K_\delta\right)$.

First consider T(red). Its crutial part is to calculate $q := \lfloor \mu_{kl} \rceil = \lfloor \lambda_{kl}/d_l \rceil$. This is an integer, and no matter how big $\mathrm{s}(\lambda_{kl})$ and $\mathrm{s}(d_l)$ were, it can be much smaller:

$$\log |q| = \log \left| \lfloor \mu_{kl} \rceil \right| \le \log 2 |\mu_{kl}| \overset{(4.13)}{\le} \frac{n-1}{2} \log \frac{nB}{L_0} + \frac{1}{2} \log n + n \log 2 \overset{(4.17)}{=} O\left(n^2 H\right).$$

On the left, we show all steps of the reduction, and on the right, we gave the complexities of the major operations (note that D dominates mF, and that b_l is a vector with n components):

$q := \left\lfloor \frac{\lambda_{kl}}{d_l} \right\rceil$	$\mathrm{T}\left(\left\lfloor \frac{\lambda_{kl}}{d_l} \right\rceil \right) = O\left(m^2 \operatorname{Mul}(m^2 D)\right)$ by (2.32)
$b_k := b_k - q b_l$	$\mathrm{T}(q b_l) = O\left(nm \operatorname{Mul}(n^2 H, D)\right)$ by (2.22)
for $j := 1$ **to** $l-1$ **do**	
$\quad \lambda_{kj} := \lambda_{kj} - q \lambda_{lj}$	$\mathrm{T}(q \lambda_{lj}) = O\left(m \operatorname{Mul}(n^2 H, D)\right)$ by (2.22)
$\lambda_{kl} := \lambda_{kl} - q d_l$	$\mathrm{T}(q d_l) = O\left(m \operatorname{Mul}(n^2 H, D)\right)$ by (2.22)

The complexity of all these steps is:

$$\begin{aligned} \mathrm{T}(\mathrm{red}) &= \mathrm{T}\left(\left\lfloor \tfrac{\lambda_{kl}}{d_l} \right\rceil \right) + \mathrm{T}(q b_l) + (l-1)\,\mathrm{T}(q \lambda_{lj}) + \mathrm{T}(q d_l) = \\ &= O\left(m^2 \operatorname{Mul}(m^2 D) + nm \operatorname{Mul}(n^2 H, D)\right). \end{aligned}$$

Now consider the swap operation. It performs the following calculations:

$b_k \leftrightarrow b_{k-1}$
for $j := 1$ **to** $k-2$ **do**
$\quad \lambda_{k,j} \leftrightarrow \lambda_{k-1,j}$
$d'_{k-1} := \frac{d_{k-2} d_k + \lambda_{k,k-1}^2}{d_{k-1}}$
for $i := k+1$ **to** n **do**
$\quad \lambda'_{i,k} := \frac{d_k \lambda_{i,k-1} - \lambda_{k,k-1} \lambda_{i,k}}{d_{k-1}}$
$\quad \lambda_{i,k-1} := \frac{d'_{k-1} \lambda_{i,k} + \lambda_{k,k-1} \lambda'_{i,k}}{d_k}$
$\quad \lambda_{i,k} := \lambda'_{i,k}$
$d_{k-1} := d'_{k-1}$

The major operations have a similar structure than the recursive formula of the Bareiss algorithm (3.1), so a very similar calculation can be performed. The differences are that B is replaced by D (but they both dominate mF), that all operations are performed

$O(n)$ times except the inversion of the denominator, which is done twice. This leads to the total time of the swap operation, which is:

$$\mathrm{T}(\text{swap}) = O\left(m^3 \operatorname{Mul}(mD) + nm^3 \operatorname{Mul}(D)\right).$$

The third main part is the comparison in the main **if** statement, which can be expressed equivalently as:

$$d_{k-2}d_k + \lambda_{k,k-1}^2 < \delta d_{k-1}^2.$$

The multiplications, like in the swap part, take $O(m^2 \operatorname{Mul}(D))$ time, and the comparison itself is $O(m^2 \operatorname{Mul}(mD))$ by (2.29), so

$$\mathrm{T}(\text{cmp}) = O(m^2 \operatorname{Mul}(mD)).$$

Now we can put together the running time of the whole algorithm. It has a main **while** loop, where each iteration is either a swap step or a reduction step (not to be confused with T(swap) and T(red)). The swap step makes a reduction, a comparison and a swap, and the reduction step makes a comparison and $k-1$ reductions. Their running time is:

$$\mathrm{T}(\text{swap step}) = \mathrm{T}(\text{cmp}) + \mathrm{T}(\text{red}) + \mathrm{T}(\text{swap}) = O\left(m^2 \operatorname{Mul}(m^2 D) + nm^3 \operatorname{Mul}(D)\right),$$

$$\mathrm{T}(\text{red step}) = \mathrm{T}(\text{cmp}) + (k-1)\,\mathrm{T}(\text{red}) = O\left(nm^2 \operatorname{Mul}(m^2 D) + n^2 m \operatorname{Mul}(n^2 H, D)\right).$$

If N is the number of main iterations, we proved in Lemma 4.3 (combined with Lemma 4.5) that $N = O\left(n^3 H K_\delta\right)$. Therefore, the running time of the LLL algorithm is:

$$\begin{aligned}
\mathrm{T}_{\mathbb{Z}[\theta]}(\text{LLL}) &\le N\,\mathrm{T}(\text{swap step}) + N\,\mathrm{T}(\text{red step}) = \\
&= NO\left(nm^2 \operatorname{Mul}(m^2 D) + n^2 m \operatorname{Mul}(n^2 H, D)\right) = \\
&= O\left(n^4 m H K_\delta (m \operatorname{Mul}(n^6 m^2 H^2 K_\delta) + n^5 H K_\delta \operatorname{Mul}(n^2 H))\right),
\end{aligned}$$

where, again, the meaning of the variables are the following:

- $H = O\left(mA + m^2 F + m \log n\right)$ by Lemma 4.5,
- n is the dimension of the lattice,
- m is the degree of the algebraic number field,
- $F = \log\left(\max_{i=0}^{m-1} |f_i| + 1\right)$, where $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ is the minimal polynomial of the primitive element θ in the number field,
- $A = \max_{i=1}^{n} \mathrm{s}(b_i)$, the coefficient size of the input vectors,
- $K_\delta = \frac{1}{\log \frac{1}{\delta}}$, where δ is the parameter of the LLL algorithm between $1/4 < \delta < 1$.

For comparison, the running time for integers is the following (see e.g. in [7]):

$$\mathrm{T}_{\mathbb{Z}}(\text{LLL}) = O\left(n^4 \log B \operatorname{Mul}(n \log B) K_\delta\right),$$

where $B = \max_{i=1}^{n} \|b_i\|_2^2$.

In the following table, we compare the results for several different $\operatorname{Mul}(\cdot)$ functions in both $\mathbb{Z}[\theta]$ and $\mathbb{Z}$. For better comparison, we define $A := \log \max_{i=1}^{n} \|b_i\|_\infty$ for $\mathbb{Z}$, so we have $\log B \le 2A + \log n$.

$\operatorname{Mul}(X)$	$\mathrm{T}_{\mathbb{Z}[\theta]}(\text{LLL})$	$\mathrm{T}_{\mathbb{Z}}(\text{LLL})$
X^2	$O\left(n^{16} m^{11} (A + mF + \log n)^5 K_\delta^3\right)$	$O\left(n^6 (A + \log n)^3 K_\delta\right)$
$X^{\log_2 3}$	$O\left(n^{13.6} m^{9.4} (A + mF + \log n)^{4.2} K_\delta^{2.6}\right)$	$O\left(n^{5.6} (A + \log n)^{2.6} K_\delta\right)$
$X \log X \log\log X$	$\tilde{O}\left(n^{10}(n + m^3) m^4 (A + mF + \log n)^3 K_\delta^2\right)$	$\tilde{O}\left(n^5 (A + \log n)^2 K_\delta\right)$

In the last row, $\tilde{O}(N) = O(N \log N \log\log N)$ as in Section 3.

4.4 Notes on the LLL result

We proved that the LLL algorithm does not suffer from exponential coefficient growth even for exact algebraic numbers, and it has polynomial time complexity. However, its running time is significantly different from the integer version, not only in the presence of additional parameters (m and F), but also in the order of the basic parameters (n and A). This shows how much harder it is to contain the coefficient size ($\mathrm{s}(\cdot)$) than the normal size ($|\cdot|$), for example while $\|b_i\|_2^2 \leq nB$ for most i, we have $\mathrm{s}(b_i) = O(n^5 H^2 K_\delta)$.

However, our actual result for $\mathbb{Z}[\theta]$ is just a very pessimistic upper bound for the worst-case complexity. We strongly believe that the algorithm is much faster in practice. For example, the number of iterations in the algorithm is $N = O\left(n^3 H K_\delta\right)$, but this is only a theoretical limit, and in practice, it can often be just a few (i.e. $O(n)$) steps. Furthermore, we used L_0, the size of the shortest vector in the lattice, and we calculated a worst-case theoretical lower bound for it: $\log \frac{1}{L_0} = O\left(nmA + nm^2F + nm \log n\right)$. But in practice, there is no special reason why the shortest vector would be so extremely small. If we can make an assumption that it is constant (i.e. $O(1)$), then the running time can be reduced by several powers. It is easy to check that e.g. for basic multiplication ($\mathrm{Mul}(X) = X^2$), these two practical assumptions reduce n^{16} to n^{10}.

We suspect that the powers can be reduced even further in average. It is out of scope of the present theoretical article but is a subject of future research to perform systematic measurements on the actual running time to confirm these claims.

References

[1] H. Cohen: A Course in Computational Algebraic Number Theory. *Springer-Verlag Berlin Heidelberg*, 1996

[2] K. O. Geddes, S. R. Czapor, G. Labahn: Algorithms for Computer Algebra. *Kluwer Academic Publishers*, 1992

[3] M. Pohst, H. Zassenhaus: Algorithmic Algebraic Number Theory. *Cambridge University Press*, 1997

[4] K. Belabas: Topics in computational algebraic number theory. *Journal de théorie des nombres de Bordeaux* 16 (2004), pp. 19–63

[5] J.-F. Biasse, C. Fieker, T. Hofmann: On the computation of the HNF of a module over the ring of integers of a number field. *Journal of Symbolic Computation* 80 (2017), pp. 581–615

[6] E. H. Bareiss: Sylvester's identity and multistep integer-preserving Gaussian elimination. *Mathematics of Computation* 22 (1968), pp. 565–578

[7] A. K. Lenstra, H. W. Lenstra, L. Lovász: Factoring Polynomials with Rational Coefficients. *Mathematische Annalen* 261. (1982), pp. 515–534

[8] P. Q. Nguyen, D. Stehlé: An LLL algorithm with quadratic complexity. *SIAM Journal on Computing* 39 (2009), pp. 874–903

[9] I. Morel, D. Stehlé, G. Villard: H-LLL: Using Householder inside LLL. *Proceedings of ISSAC* (2009), pp. 271–278

[10] T. Plantard, W. Susilo, Z. Zhang: Adaptive precision floating point LLL. *C. Boyd, L. Simpson (editors) Information Security and Privacy, ACISP 2013, Lecture Notes in Computer Science*, 7959 (2013). pp. 104–117

[11] G. Saruchi, I. Morel, D. Stehlé, G. Villard: LLL reducing with the most significant bits. *ISSAC, Kobe, Japan* (2014), pp. 367–374

[12] C. Fieker, D. Stehlé: Short bases of lattices over number fields. *G. Hanrot, F. Morain, E. Thomé (editors) Algorithmic Number Theory, 9th International Symposium, Proceedings* 6197 (2010), pp. 157–173

[13] C. Lee, A. Pellet-Mary, D. Stehlé, A. Wallet: An LLL algorithm for module lattices. *Advances in Cryptology, ASIACRYPT* (2019), pp. 59–90

[14] A. Pethő, M. E. Pohst, Cs. Bertók: On multidimensional Diophantine approximation of algebraic numbers. *Journal of Number Theory* 171. (2017), pp. 422–448

[15] J. W. S. Cassels: An Introduction to the Geometry of Numbers. *Springer-Verlag Berlin Heidelberg*, 1997

[16] D. E. Knuth: The art of computer programming, Vol. 2. *Addison-Wesley*, 1998