CERTIFIED LATTICE REDUCTION

THOMAS ESPITAU AND ANTOINE JOUX

ABSTRACT. Quadratic form reduction and lattice reduction are fundamental tools in computational number theory and in computer science, especially in cryptography. The celebrated Lenstra–Lenstra–Lovász reduction algorithm (so-called LLL) has been improved in many ways through the past decades and remains one of the central methods used for reducing *integral* lattice basis. In particular, its floating-point variants—where the rational arithmetic required by Gram–Schmidt orthogonalization is replaced by floating-point arithmetic— are now the fastest known. However, the systematic study of the reduction theory of *real* quadratic forms or, more generally, of real lattices is not widely represented in the literature. When the problem arises, the lattice is usually replaced by an integral approximation of (a multiple of) the original lattice, which is then reduced. While practically useful and proven in some special cases, this method doesn't offer any guarantee of success in general. In this work, we present an adaptive-precision version of a generalized LLL algorithm that covers this case in all generality. In particular, we replace floating-point arithmetic by Interval Arithmetic to certify the behavior of the algorithm. We conclude by giving a typical application of the result in algebraic number theory for the reduction of ideal lattices in number fields.

1. INTRODUCTION

In a general setting, a *lattice* Λ is a free $\mathbf{Z}$-module of finite rank, endowed with a positive-definite bilinear form on its ambient space $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$, as presented for instance in [LS17]. In particular, this definition implies that Λ is discrete in its ambient space for the topology induced by the scalar product. This formalism encompasses the well-known *Euclidean lattices* when taking the canonical scalar product of $\mathbf{R}^d$, but also lattices arising from ideals in rings of integers of number fields. The rank of the lattice Λ is defined as the dimension of the vector space $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$. By definition of a finitely-generated free module, there exists a finite set of vectors $b_1, \ldots, b_{\mathrm{rk}\,\Lambda} \in \Lambda$ such that $\Lambda = \bigoplus_{i=1}^{\mathrm{rk}\,\Lambda} b_i \mathbf{Z}$. Such a family is called a basis of the lattice and is not unique. In fact, as soon as $\mathrm{rk}\,\Lambda \geq 2$ there are infinitely many bases of Λ. Some among those have interesting properties, such as having reasonably small vectors and low orthogonality defects. They are informally called *reduced bases* and finding them is the goal of *lattice reduction*.

Numerous algorithms arising in algebraic number theory heavily rely on lattice reduction, for example, the computation of normal forms of integral matrices (see [Jäg05] for the Hermite Normal Form and [HMM98] for the Smith Normal

Form), class group computations in a number field [GJ16, BF13], or even the enumeration of points of small height near algebraic curves [Elk00].

Even for lattices that use the canonical scalar product, there is a deep link with bilinear forms that clearly appears when considering the *Gram matrix* of a basis $\mathcal{B} = \{b_1, \ldots, b_d\}$, that is, the real symmetric matrix $\mathcal{G} = (\langle b_i, b_j \rangle)_{i,j}$.

The study of these reduction problems is not recent and goes back to the works of Lagrange and Gauss. These early works were expressed in terms of reduction of quadratic form, more precisely integral binary quadratic forms[1] and led to a method often called Gauss' algorithm. This method can be seen as a 2-dimensional extension of the Euclid algorithm for computing the greatest common divisor of two integers. In 1850, Hermite proved a general upper bound on the length of the shortest vector in a lattice, given as a function of the dimension and of a very important invariant called the determinant, which is defined in Section 2.1. This bound involves the so-called Hermite constant and has recently been rephrased in algorithmic terms [Ngu10, Hermite's Algorithms]. A century later, in 1982, Lenstra, Lenstra and Lovász designed the LLL *algorithm* [LLL82], with the polynomial factorization problem as an application, following the work of Lenstra on integer programming [Len83]. This algorithm constitutes a breakthrough in the history of lattice reduction algorithm, since it is the first to have a runtime polynomial in terms of the dimension. It was followed by many improvements lowering its complexity or improving the output's quality.

Current implementations of LLL often work with low precision approximations in order to greatly speed-up the computations. Indeed, the algorithm works surprisingly well even with such reduced precisions, even if some care needs to be taken to avoid infinite loops. Moreover, once the result is obtained, it can verified efficiently as shown in [Vil07].

We propose here an alternative strategy where we not only certify that the end-result is a reduced basis but also that the algorithm followed a valid computation path to reach it. This strongly deviates from other approaches that have been taken to obtain guaranteed lattice reduced basis. At first, this may seems irrelevant. After all, one might claim that a basis satisfying the end conditions of LLL is what is desired and that the computation path doesn't matter. However, as shown in [KV16] for Siegel-reduced bases, a reduced basis chosen uniformely at random behaves as the worst-case allowed by the final inequalities. By constrast, bases produced by the LLL algorithm are usually much better than this worst-case. This argues in favor of trying to follow the algorithm defintion exactly to better understand the phenomenon. In particular, this option might be invaluable for experiments performed toward analyzing this gap.

The present article also relies on Interval Arithmetic, a representation of reals by intervals—whose endpoints are floating-point numbers—that contain them. Arithmetic operations, in particular the basic operations $+, -, \times, \div$ can be redefined in this context. The main interest of this representation lies in its *certification* property: if real numbers are represented by intervals, the interval resulting from the evaluation of an algebraic expression contains the exact value of the evaluated expression.

---

[1] This can be viewed as the reduction of integral dimension-two lattices.

For some authors, Interval Arithmetic was introduced by R. Moore in 1962 in his Ph.D. thesis [Moo62]. For others, it can be dated back to 1958, in an article of T. Sunaga [Sun09] which describes an algebraic interpretation of the lattice of real intervals, or even sooner in 1931 as a proposal in the Ph.D. thesis [You31] of R.C. Young at Cambridge. Its main asset—calculating directly on sets—is nowadays used to deterministically determine the global extrema of a continuous function [RR88] or localizing the zeroes of a function and (dis)proving their existence [JKDW01]. Another application of Interval Arithmetic is to be able to detect lack of precision at run-time of numerical algorithms, thanks to the guarantees it provides on computations. This can, in particular, be used to design adaptive-precision numerical algorithms.

In the present paper, we propose to transform and generalize the LLL algorithm into an adaptive-precision version, which can reduce arbitrary lattices and follows a certified flow of execution. More precisely, it uses Interval Arithmetic to validate the size-reduction and exchange steps that occur within LLL.

Organisation of the paper. In Section 2, we briefly introduce reduction theory and present the $\textsc{l}^2$ variant of the LLL algorithm. Section 3 aims at describing the basics of Interval Arithmetic used in Section 4 to handle the problem of representation of real lattices. The framework of this latter section is then used in Section 5 to derive a certified reduction algorithm for real lattices. Section 6 presents an application to algorithmic number theory.

Notations and conventions.

General notations. As usual, the bold capitals $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{C}$ refer respectively to the ring of integers and the fields of rational, real and complex numbers. Given a real number x, the integral roundings *floor*, *ceil* and *round to nearest integer* are denoted respectively by $\lfloor x \rfloor, \lceil x \rceil, \lfloor x \rceil$. Note that the rounding operator is ambiguous when operating on half-integers. However, either choice when rounding is acceptable in lattice reduction algorithms. In fact, in this context, it is often enough to return an integer close to x, not necessarily the closest.

These operators are extended to operate on vectors and matrices by point-wise composition. The complex conjugation of $z \in \mathbf{C}$ is denoted by the usual bar $\bar{z}$ whereas the real and imaginary parts of a complex z are indicated by respectively $\mathfrak{R}(z)$ and $\mathfrak{I}(z)$. All logarithms are taken in base 2.

Matrices and norms. For a field $\mathbf{K}$, let us denote by $\mathbf{K}^{d \times d}$ the space of square matrices of dimension d over $\mathbf{K}$, $\mathrm{Gl}_d(\mathbf{K})$ its group of invertible elements and $\mathcal{S}_d(\mathbf{K})$ its subspace of symmetric matrices. For a complex matrix A, we write $A^\dagger$ for its conjugate transpose. For a vector v, we denote by $\|v\|_\infty$ its absolute (or infinity) norm, that is the maximum of the absolute value of its entries. We similarly define the matrix *max norm* $\|B\|_{\max} = \max_{(i,j) \in [1 \cdots d]^2} |B_{i,j}|$, for any matrix B.

Computational setting. The generic complexity model used in this work is the random-access machine (RAM) model and the computational cost is measured in bits operations. $\mathcal{M}(k)$ denotes the complexity of the multiplication of two integers of bit length at most k. It is also the cost of the multiplication of two floating-point numbers at precision k, since the cost of arithmetic over the exponents is negligible with regards to the cost of arithmetic over the mantissae.

2. BASICS OF LATTICE REDUCTION

2.1. **Orthogonalization.** Let us fix an Euclidean space $(E, \langle \cdot, \cdot \rangle)$, i.e. a real vector space E together with a positive-definite bilinear form $\langle \cdot, \cdot \rangle : E \times E \to \mathbf{R}$. As usual, two vectors $x, y \in E$ are said to be orthogonal—with respect to the form $\langle \cdot, \cdot \rangle$—if $\langle x, y \rangle = 0$. More generally a family of vectors is orthogonal if its elements are pairwise orthogonal.

Now consider $S = (b_1, \ldots, b_r)$, a family of linearly independent vectors of E. The *flag* $\mathcal{F}_S$ associated to S is the finite increasing chain of subspaces:

$$b_1 \mathbf{R} \subset b_1 \mathbf{R} \oplus b_2 \mathbf{R} \subset \cdots \subset \bigoplus_{i=1}^{r} b_i \mathbf{R}.$$

The orthogonal complement $S^\perp$ is defined as the subspace $\{x \in E \mid \forall i, \ \langle x, b_i \rangle = 0\}$. Denote by π_i the orthogonal projection on $(b_1, \ldots, b_{i-1})^\perp$, with the convention that $\pi_1 = \mathrm{Id}$. The Gram–Schmidt orthogonalization process—shorthanded as GSO—is an algorithmic method for orthogonalizing S while preserving its flag. It constructs the orthogonal set $S^* = (\pi_1(b_1), \ldots, \pi_r(b_r))$. The computation S^* can done inductively as follow:

$$\pi_1(b_1) = b_1$$

$$\forall 1 < i \leq r, \quad \pi_i(b_i) = b_i - \sum_{j=1}^{i-1} \frac{\langle b_i, \pi_j(b_j) \rangle}{\langle \pi_j(b_j), \pi_j(b_j) \rangle} b_j.$$

Define the *Gram matrix*, associated to a family of vectors $S = (b_1, \ldots, b_r)$, as the symmetric matrix of scalar products: $\mathcal{G}_S = (\langle b_i, b_j \rangle)_{(i,j) \in [1 \cdots r]^2}$. The (co)volume of S, also called its determinant, is defined as the square root of the Gram determinant $\det \mathcal{G}_S$. It can be easily computed from the Gram-Schmidt vectors S^* as:

$$\mathrm{covol}(S) = \prod_{i=1}^{r} \|\pi_i(b_i)\|$$

2.2. **Lattices and reduction.**

Definition 2.1. *A (real) lattice Λ is a finitely generated free $\mathbf{Z}$-module, endowed with a positive-definite bilinear form $\langle \cdot, \cdot \rangle$ on its ambient space $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$.*

By definition of the tensor product, there is a canonical injection that sends a vector v to $v \otimes 1$ in the ambient space and preserves linear independence. Thus, the rank of Λ as a $\mathbf{Z}$-module, is equal to the dimension of the vector space $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$.

Denoting by d the rank of the lattice, a basis *of Λ is a family $b_1, \ldots, b_d$ of elements of Λ such that $\Lambda = \bigoplus_{i=1}^{d} b_i \mathbf{Z}$.*

In the sequel, we identify Λ with its canonical image $\Lambda \otimes 1$ and thus view the lattice as an additive subgroup of its ambient space $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$. When the context makes it clear, we may omit to write down the bilinear form associated to a lattice Λ. Throughout this section, $\| \cdot \|$ stands for the Euclidean norm induced by $\langle \cdot, \cdot \rangle$, unless stated otherwise. As usual, any two bases $(b_1, \ldots, b_d)$ and $(b'_1, \ldots, b'_d)$ of Λ are related by a unimodular transformation, i.e., a linear transformation represented by a $d \times d$ integer matrix of determinant ± 1.

Lemma 2.1. *A lattice Λ is discrete for the topology induced by the given norm on its ambient space. I.e., there exists a real $\epsilon_\Lambda > 0$ such that for any pair (x, y) of elements of Λ with $x \neq y$ we have:*

$$\|x - y\| \geq \epsilon_\Lambda.$$

The largest possible value for ϵ_Λ in the above inequality is equal to the norm of the shortest non-zero vector of Λ, which is traditionally called the first minimum *or the* minimum distance *of the lattice and denoted by $\lambda_1(\Lambda)$.*

Proof. Let $\mathcal{B} = (b_1, \ldots, b_d)$ be a basis of Λ. Let $\mathcal{B}^* = (\pi_1(b_1), \ldots, \pi_d(b_d))$ be the orthogonal basis obtained by applying Gram-Schmidt orthogonalization to the canonical image of $\mathcal{B}$ in $\Lambda \otimes_{\mathbf{Z}} \mathbf{R}$. This orthogonalization is taken using as scalar product the given bilinear form.

Assume by contradiction that there exist pairs of distinct vectors with the norm of their difference arbitrarily small. Since the difference is also an element of Λ, there are non-zero elements of arbitrarily small norm in Λ. For any integer $i > 0$, choose a vector x_i in Λ with $\|x_i\|^2 \leq 2^{-i}$. Decompose x_i in the basis $\mathcal{B}^*$ as $x_i = \sum_{j=1}^{d} \chi_i^{(j)} \pi_j(b_j)$. For any pair of integers i, j we see that $|\chi_i^{(j)}|^2 \|\pi_j(b_j)\|^2 \leq \|x_i\|^2 \leq 2^{-i}$. As a consequence, each sequence $\chi^{(j)}$ converges to zero. Multiplying by the basis-change matrix, we see that the coordinates of x_i in the basis $b_1, \ldots, b_d$ also converge to zero. Since these coordinates are integral, the sequences are ultimately constant and x is also ultimately constant (and null). This contradicts the choice of x_i as a non-zero element. $\square$

2.3. The LLL reduction algorithm.

In 1982, Lenstra, Lenstra and Lovász [LLL82] proposed a notion called LLL *reduction* and a polynomial-time algorithm that computes an LLL-reduced basis from arbitrary basis of the same lattice. Their reduction notion is formally defined as follows:

Definition 2.2 (LLL reduction)**.** *A basis $\mathcal{B} = (b_1, \ldots, b_d)$ of a lattice is said to be δ-LLL-**reduced** for a parameter $1/4 < \delta \leq 1$, if the following conditions are satisfied:*

$$(1) \qquad \forall i < j, \quad |\langle b_j, \pi_i(b_i) \rangle| \leq \frac{1}{2} \|\pi_i(b_i)\|^2 \quad \text{(size-reduction condition)}$$

$$(2) \qquad \forall i, \quad \delta \|\pi_i(b_i)\|^2 \leq \left(\|\pi_{i+1}(b_{i+1})\|^2 + \frac{\langle b_{i+1}, \pi_i(b_i) \rangle}{\|\pi_i(b_i)\|^2} \right) \quad \text{(Lovász condition)}$$

In order to find a basis satisfying these conditions, it suffices to iteratively modify the current basis at any point where one of these conditions is violated. This yields the simplest version of the LLL algorithm. As in [LLL82], it is only defined for full-rank sublattice of $\mathbf{Z}^d$. It was remarked by Lovász and Scarf in [LS92] that the same algorithm also works with an arbitrary integral-valued scalar product. The method can be extended to deal with lattices described by a generating family rather than by a basis [Poh87].

Algorithm 1: The original LLL algorithm.

Parameters: $\delta \in (1/4, 1)$
Input: Initial basis $\mathcal{B} = (b_1, \ldots, b_d)$
Result: A δ-LLL-reduced basis

1 $k \leftarrow 2$;
2 Compute the $\pi_i(b_i)$'s with the GSO process (Paragraph 2.1);
3 **while** $k \leq d$ **do**
4   **for** $j = k-1$ **downto** 1 **do** $b_k \leftarrow b_k - \left\lceil \frac{\langle b_k, \pi_j(b_j) \rangle}{\|\pi_j(b_j)\|^2} \right\rfloor \cdot b_j$;
5   **if** $\delta \|\pi_{k-1}(b_{k-1})\|^2 \leq \|\pi_k(b_k)\|^2 + \langle b_k, \pi_{k-1}(b_{k-1}) \rangle^2 / \|\pi_{k-1}(b_{k-1})\|^2$ **then**
6         $k \leftarrow k+1$;
7   **else**
8   Swap b_k and b_{k-1}; Update $\pi_k(b_k)$ and $\pi_{k-1}(b_{k-1})$;
9         $k \leftarrow \max(k-1, 2)$;
10 **end**
11 **return** $(b_1, \ldots, b_d)$

2.3.1. *Decrease of the potential and complexity.* The algorithm can only terminate when the current lattice basis is LLL-reduced. Moreover, as shown in [LLL82], it terminates in polynomial time when $\delta < 1$. Indeed, consider the (square of the) product of the covolumes of the flag associated with a basis: $\prod_{i=1}^{d} \|\pi(b_i)\|^{2(n-i+1)}$, which is often called its *potential*. This value decreases by a factor at least δ^{-1} in each exchange step and is left unchanged by other operations. Indeed:

- The flag is not modified by any operation other than swaps.
- A swap between b_k and b_{k-1} only changes the sublattice spanned by the first $k-1$ vectors. The corresponding covolume $\prod_{i=1}^{k-1} \|\pi(b_i)\|^2$ decreases by a factor at least δ^{-1} and so does the potential.

Since the total number of iterations can be bounded by twice the number of swaps plus the dimension of the lattice, this suffices to conclude that it is bounded by $\mathrm{O}(d^2 \log \|B\|_{\max})$ where B is the matrix of the initial basis.

As the cost of a loop iteration is of $\mathrm{O}(dn)$ arithmetic operations on *rational* coefficients of length at most $\mathrm{O}(d \log \|B\|_{\max})$, the total cost in term of arithmetic operations is loosely bounded by $\mathrm{O}(d^6 \log^3 \|B\|_{\max})$. By being more precise in the majoration of the bit length of the integers appearing in LLL, this analysis can be improved. Kaltofen in [Kal83] bounds the complexity by

$$\mathrm{O}\left(\frac{d^5 \log^2 \|B\|_{\max}}{d + \log \|B\|_{\max}} M(d + \log \|B\|_{\max}) \right).$$

2.3.2. *A bound on the norm of reduced elements.*

Proposition 2.1. *Let $1/4 < \delta < 1$ be an admissible LLL parameter. Let $(b_1, \ldots, b_d)$ be a δ-LLL reduced basis of rank-d lattice $(\Lambda, \langle \cdot, \cdot \rangle)$. Then for any $1 \leq k \leq d$:*

$$\mathrm{covol}(b_1, \ldots, b_k) \leq \left(\delta - \frac{1}{4} \right)^{-\frac{(d-k)k}{4}} \mathrm{covol}(\Lambda)^{\frac{k}{d}}.$$

Note that this is an easy generalization of the bound on the norm of b_1 which is given in most texts. It appears among other related inequalities in [PT08]. For completeness, a proof is given in Appendix.

2.3.3. *Floating point representation.* The total cost of the LLL algorithm is dominated by the computation to handle arithmetic on rational values. A first idea of De Weger [DW87] to overcome this issue is to avoid the use of denominators by multiplying all the quantities by their common denominator. This is slightly more efficient in practice but doesn't improve the asymptotics. Another idea is to remark that the norms of the rational values remain small and to try to use approximations instead of exact values. However, directly replacing rationals in the LLL algorithm by floating-point approximations leads to severe drawbacks. The algorithm might not even terminate, and the output basis is not guaranteed to be LLL-reduced.

The first *provable* floating-point version of the algorithm is due to Schnorr in [Sch88], with complexity $O\big(d^4 \log(\|B\|_{\max})M(d + \log\|B\|_{\max})\big)$. One of the key ingredients to achieve this reduction is to slightly relax the definition of the size-reduction, in order to compensate for the approximation errors introduced by the use of floating-point arithmetic. We call *admissible* any parameters (δ, η) satisfying $1/4 < \delta < 1$, and $1/2 < \eta < \sqrt{\delta}$ and define:

Definition 2.3 ((δ, η)-LLL reduction)**.** *Let (δ, η) be admissible parameters. A basis $\mathcal{B}$ of a lattice is said to be (δ, η)-**LLL-reduced** if the following condition is satisfied:*

(3) $\quad \forall i < j, \quad |\langle b_j, \pi_i(b_i)\rangle| \leq \eta\|\pi_i(b_i)\|^2 \quad$ *(Approximate size-reduction condition)*

together with the Lovász condition, which is kept unchanged from Definition 2.2.

Using naive multiplication, the cost of Schnorr's algorithm is cubic in $\log(\|B\|_{\max})$. The introduction of approximate size reduction removes the need to know with extreme precision values close to half-integers. Instead, approximate size reduction of such values can be achieved by rounding either up or down in an arbitrary (possibly randomized) manner. In our pseudo-code, we use a function called η-CLOSEST-INTEGER to achieve this rounding, returning an integer at distance at most η of the function's argument.

2.4. **The L^2 algorithm.** The L^2 algorithm is a variant of Schnorr-Euchner version [SE94] of LLL. By contrast with the original algorithm, L^2 computes the GSO coefficients on the fly as they are needed instead of doing a full orthogonalization at the start. It also uses a lazy size reduction inspired by the Cholesky factorization algorithm. These optimizations yield an improved lattice reduction with running time

$$O\big(d^5(d + \log(\|B\|_{\max})) \log(\|B\|_{\max})\big).$$

As usual in lattice reduction, while performing the Gram-Schmidt orthogonalization of $\mathcal{B}$, we also compute QR-decomposition of B into $B^* \cdot M$ where B^* is the matrix representing the $(\pi_i(b_i))_{1 \leq i \leq d}$, and M is the upper unitriangular matrix, whose coefficients with $j \geq i$ are $M_{i,j} = \frac{\langle b_j, \pi_i(b_i)\rangle}{\|\pi_i(b_i)\|^2}$. Thus, the Gram matrix associated to the basis, i.e., $G = B^T B$ satisfies:

$$G = M^T \cdot B^{*T} \cdot B^* \cdot M = M^T \cdot D \cdot M$$

where D is a diagonal matrix whose entries are $\|\pi_i(b_i)\|^2$. We denote by R the matrix $D \cdot M$, and thus have $G = R^T \cdot M = M^T \cdot R$.

We give the pseudo-code of the Lazy Size-Reduction procedure as Algorithm 2 and of the $\textsc{l}^2$ algorithm as Algorithm 3. Both use classical formulas relating R, M and B^* to perform the computations.

2.4.1. *Precision required.* The precision required by the $\textsc{l}^2$-Algorithm is

$$d \log \left(\frac{(1+\eta)^2}{(\delta - \eta)^2} + \epsilon \right) + o(d)$$

bits for any $\epsilon > 0$, i.e., almost linear in the dimension of the lattice. Moreover, as discussed in [NS09], it appears that—even though this bound can be shown to be sharp by specific examples—experiments indicate that the number of bits required *on average* is, in fact, lower.

This phenomenom is well-known and is often used in existing algorithms and softwares in the form of a compute-and-verify paradigm. For example, this is default strategy of the well-known FPLLL [Tea16]. It relies on the fact that verifying that a lattice basis is indeed reduced is much less costly than the reduction itself, as shown in [Vil07]. In addition, it is necessary to take several conservative measures in order to prevent the implementation to enter potentially infinite loops.

The approach we propose deviates from this paradigm. Instead of guaranteeing the end-result, we want to make sure that the whole copmputation follows the mathematical definition of the algorithm. With low-precision approximations, it is unclear how this could be done. However, interval-arithmetic offers a neat solution to achieve this goal.

3. Interval Arithmetic and its certification property

Interval arithmetic is a representation of reals by intervals that contain them. For instance, one can specify a value x with an error ϵ by giving an interval of length ϵ containing x. For example, the constant π can be represented with an error of 10^{-2} by the interval $[3.14, 3.15]$. Interval arithmetic is crucial in the context of *certified* numerical computations, where reals can only be represented with finite precision. For more details, the interested reader can consult an extensive reference, such as [Moo77].

In the following, we denote by $\underline{x}$ a closed interval $[\underline{x}^-, \underline{x}^+]$. We define its *diameter* as the positive real $\underline{x}^+ - \underline{x}^-$ and its *center* as the real $\frac{1}{2}(\underline{x}^+ + \underline{x}^-)$.

Given a real-valued function $f(x_1, \ldots, x_n)$ an interval-arithmetic realization of f is an interval-valued function F such that the interval $F(\underline{x_1}, \ldots, \underline{x_n})$ contains all the values $f(x_1, \ldots, x_n)$ for $(x_1, \ldots, x_n)$ in $\underline{x_1} \times \cdots \times \underline{x_n}$.

If F always returns the smallest possible interval, it is called a *tight* realization, otherwise it is called *loose*. In practice, tight realizations can only be achieved in very simple specific cases. However, even a loose realization can suffice to certify the correctness of a computation.

Another important property of interval arithmetic is that it can be used to compare numbers in a certified way, as long as the intervals that represent them are disjoint.

3.1. **Some useful interval-arithmetic realizations.**

Algorithm 2: The lazy size reduction algorithm, η-LAZYRED.

Input: Initial basis $\mathcal{B} = (b_1, \ldots, b_d)$, with G, R and M. An integer
$\qquad 1 \leq k \leq d$.
Result: Size-reduces b_k, updates G, R, M and returns $s^{(k)}$

1 done $\leftarrow$ **false**;
2 while done = **false do**
3 $\quad$ **for** $j = 1$ **to** $k - 1$ **do**
4 $\quad\quad$ $R_{k,j} \leftarrow G_{k,j}$; **for** $i = 1$ **to** $j - 1$ **do** $R_{k,j} \leftarrow R_{k,j} - M_{j,i}R_{k,i}$;
5 $\quad\quad$ $M_{k,j} \leftarrow R_{k,j}/R_{j,j}$;
6 $\quad$ **end**
7 $\quad$ $s_1^{(k)} \leftarrow G_{k,k}$; **for** $j = 2$ **to** k **do** $s_j^{(k)} \leftarrow s_{j-1}^{(k)} - M_{k,j-1} \cdot R_{k,j-1}$;
8 $\quad$ $R_{k,k} \leftarrow s_k^{(k)}$;
9 $\quad$ **if** $(\max_{j<k} |M_{k,j}|) \leq \eta$ **then** done $\leftarrow$ **true**;
10 $\quad$ **else**
11 $\quad\quad$ **for** $i = k - 1$ **downto** 1 **do**
12 $\quad\quad\quad$ $X_i \leftarrow \eta$-CLOSEST-INTEGER$(M_{k,i})$;
13 $\quad\quad\quad$ **for** $j = 1$ **to** $i - 1$ **do** $M_{k,j} \leftarrow M_{k,j} - X_i M_{i,j}$;
14 $\quad\quad$ **end**
15 $\quad\quad$ $b_k \leftarrow b_k - \sum_{i=1}^{k} X_i b_i$; Update G accordingly;
16 $\quad$ **end**
17 end

Algorithm 3: The L^2 Algorithm.

Parameters: $\delta \in (1/4, 1), \eta \in (1/2, \sqrt{\delta})$.
Input: Initial basis $\mathcal{B} = (b_1, \ldots, b_d)$
Result: A (δ, η)-LLL-reduced basis

1 Compute $G = G(b_1, \cdots, b_d)$ in exact integer arithmetic;
2 $R_{1,1} \leftarrow G_{1,1}$;
3 $k \leftarrow 2$;
4 while $k \leq d$ **do**
5 $\quad$ Apply size reduction η-LAZYRED(k);
6 $\quad$ $k' \leftarrow k$;
7 $\quad$ **while** $(k \geq 2$ **and** $\delta R_{k-1,k-1} > s_{k-1}^{k'})$ **do** $k \leftarrow k - 1$;
8 $\quad$ $R_{k,k} \leftarrow s_k^{k'}$;
9 $\quad$ **if** $k \neq k'$ **then**
10 $\quad\quad$ **for** $i = 1$ **to** $k - 1$ **do** $M_{k,i} \leftarrow M_{k',i}$; $R_{k,i} \leftarrow R_{k',i}$;
11 $\quad\quad$ $R_{k,k} \leftarrow s_k^{k'}$;
12 $\quad\quad$ Insert $b_{k'}$ at pos k (before b_k) and update matrix G accordingly;
13 $\quad$ **end**
14 $\quad$ $k \leftarrow k + 1$;
15 end
16 return $(b_1, \cdots, b_d)$

3.1.1. *Integral representation of fixed length.* A first convenient way to represent reals at finite precision is to use integers as an approximate representation.

Definition 3.1 (Integral representation of reals)**.** *Let $x \in \mathbf{R}$ be an arbitrary real number and $n \geq 0$ a non-negative integer. Define* an integral representation at accuracy[2] n *as an interval of diameter* 2*:*

$$\underline{x}_n = [X_n - 1, X_n + 1]$$

together with a guarantee that $2^n x$ belongs to $\underline{x}_n$.

This representation is very compact, since it only requires to store the center X_n of the interval using $n + \lceil \log x \rceil$ bits. However, computing with this form of representation is not convenient. As a consequence, we only use it to represent immutable values and we convert to a different representation for computations. The reason for using the interval $[X_n - 1, X_n + 1]$ of diameter 2 rather than $[X_n - 1/2, X_n + 1/2]$ (of diameter 1) is that when $2^n x$ is very close to a half-integer, it remains possible to easily provide a valid value for X_n without computing extraneous bits of the representation of x.

3.1.2. *Fixed-point representations.* In the context of lattice reduction, it is useful to compute linear combinations with exact integral coefficients. In order to do that with approximate values initially given by centered integral representation, it is possible to use a fixed-point representation.

Definition 3.2 (Fixed point representation of reals)**.** *Let $x \in \mathbf{R}$ be an arbitrary real number and $n \geq 0$ a non-negative integer. Define a* fixed-point representation at accuracy n *of radius δ as an interval:*

$$\underline{x}_n = [X_n - \delta, X_n + \delta]$$

together with a guarantee that $2^n x$ belongs to $\underline{x}_n$.

It is easy to add or subtract such intervals by doing the computation on the center and by adding the two radii. It is also easy to multiply by an exact integer by multiplying the center by the integer and the radius by its absolute value. Integral representations are a special case of fixed-point representations, with radius equal to 1.

3.1.3. *Floating-point representation.* Another way to handle real values is to use floating point representations of the two bounds of each interval. For example, if we denote by $\lfloor x \rfloor_n$ and $\lceil x \rceil_n$ respectively the largest floating-point number below x and the lowest floating-point number above x written with n bits, the tightest floating-point representation of x with n bits of precision is the interval $I_n(x) = [\lfloor x \rfloor_n, \lceil x \rceil_n]$.

With such a representation, it becomes possible to create a realization of the elementary operations by using careful rounding when computing approximations of the bounds of the resulting interval, as shown in Figure 1. When speaking of the precision of such a representation, we simply refer to the common floating-point precision of the upper and lower bounds.

Once the elementary operations are available, they can be used to implement certified versions of any function that can classically be computed with floating point arithmetic.

---

[2]We use here the denomination of "accuracy" instead of "precision" to avoid confusions with the floating-point precision as defined in paragraph 3.1.3.

$$\left[\underline{x}^-,\underline{x}^+\right] + \left[\underline{y}^-,\underline{y}^+\right] = \left[\underline{x}^- +^- \underline{y}^-, \underline{x}^+ +^+ \underline{y}^+\right]$$

$$\left[\underline{x}^-,\underline{x}^+\right] - \left[\underline{y}^-,\underline{y}^+\right] = \left[\underline{x}^- -^- \underline{y}^-, \underline{x}^+ -^+ \underline{y}^+\right]$$

$$\left[\underline{x}^-,\underline{x}^+\right] \times \left[\underline{y}^-,\underline{y}^+\right] = \left[\min{}^-(\rho), \max{}^+(\rho)\right] \qquad \text{where} \quad \rho = \underline{x}^-\underline{y}^-, \underline{x}^+\underline{y}^-, \underline{x}^-\underline{y}^+, \underline{x}^+\underline{y}^+$$

$$\left[\underline{x}^-,\underline{x}^+\right]^{-1} = \left[\min{}^-\left(\frac{1}{\underline{x}^+}, \frac{1}{\underline{x}^-}\right), \max{}^+\left(\frac{1}{\underline{x}^+}, \frac{1}{\underline{x}^-}\right)\right]$$

$\_+^+, +^-$ *are here respectively the $+$ operator with rounding up or down. The same goes for the $-^+, -^-, \min^-, \max^+$ operators.*

FIGURE 1. Basic arithmetic operators in Interval Arithmetic

4. APPROXIMATE LATTICES

The need to reduce lattices given by approximations, especially for number-theoretic applications as been known for long. In particular, Buchmann gives in [Buc94] a bound on the required precision to achieve this goal by using a direct approximation of the input basis. However, this bound is computed in terms of a quantity called the defect that can be very large and also involves the first minimum of the lattice.

Using interval arithmetic, it becomes possible to get finer control on the precision required to perfomr the lattice reduction, even with approximate lattices.

4.1. **Approximate representation of a positive-definite matrix.** A real-valued matrix can easily be represented with the integral representation from Definition 3.1, using the same accuracy for all of its entries.

Definition 4.1 (Matrix integral representation). *Let $A = (a_{i,j})_{i,j} \in \mathbf{R}^{d \times d}$ be an arbitrary real matrix of dimension d and $n > 0$ be a fixed positive integer. A matrix of intervals*

$$\underline{A}_n = (\underline{a_{i,j}}_n)_{(i,j) \in [1 \cdots d]^2},$$

where each $\underline{a_{i,j}}_n$ is an integral representation of $a_{i,j}$ is said to integrally represent A *at accuracy n.*

We may omit the subscript n when the accuracy is clear from the context. Given a matrix A, and a matrix $B \in \underline{A}_n$, there exists a unique $d \times d$ matrix Δ with entries in $[-2, 2]$ such that $B = 2^n A + \Delta$.

In particular, we may apply this representation to symmetric matrices. In that case, we obtain the following useful lemma:

Lemma 4.1. *Let $S = (s_{i,j})_{i,j} \in \mathcal{S}_d(\mathbf{R})$ be a symmetric matrix of dimension d and $\underline{S}_n$ an integral representation of S at accuracy n. Then, for any symmetric matrix S' in $\underline{S}_n$, we have:*

$$2^n \lambda_d(S) - 2d \leq \lambda_d(S') \leq 2^n \lambda_d(S) + 2d,$$

where $\lambda_d(T)$ denotes the smallest eigenvalue of a d-dimensional symmetric matrix T.

Proof. This is a direct consequence of Weyl's inequalities for Hermitian matrices and of the relation $S' = 2^n S + \Delta$, where Δ is real symmetric with entries in $[-2, 2]$. Note that the eigenvalues of Δ all belong to $[-2d, 2d]$. □

4.2. **Representation of lattices.** In order to represent arbitrary lattices, we first need a description of their ambient space. We simply describe the ambient space V of dimension d by providing a basis $\gamma = (\gamma_1, \ldots, \gamma_d)$. Then, the scalar product $\langle \cdot, \cdot \rangle$ on V can be encoded by a Gram matrix $\mathcal{G}_\gamma = (\langle \gamma_i, \gamma_j \rangle)_{(i,j) \in [1 \cdots d]^2}$.

When the Gram matrix $\mathcal{G}_\gamma$ is integral, this already is a standard description of the lattice Γ spanned by γ. This representation appears in particular in [Coh93, Proposition 2.5.3]. We now extend this in order to represent bases and generating families of arbitrary sublattices of Γ. Let Λ be a rank $r \leq d$ sublattice of Γ given by a generating family $\ell = (\ell_1, \ldots, \ell_p)$. Since any vector in ℓ belongs to Γ, it can be expressed with integral coordinates in the basis γ. As a consequence, we can represent ℓ by a $p \times d$ integral matrix L. Moreover, the knowledge of $\mathcal{G}_\gamma$ allows us to easily compute the scalar product of any pair of vectors in Λ.

All this leads to the following definition:

Definition 4.2 (Approximate representation of a lattice)**.** *Let $\mathcal{G}_\gamma$ and L be as above and n be a non-negative integer. Denote by G the matrix of centers of an integral representation $\underline{\mathcal{G}_\gamma}_n$ at accuracy n of the Gram matrix $\mathcal{G}_\gamma$. Then the pair $(G, L) \in \mathbf{Z}^{d \times d} \times \mathbf{Z}^{p \times d}$ of integral matrices is said to* represent at accuracy n *the lattice Λ in the basis γ of Γ.*

4.2.1. *Computation of the inner product in Interval Arithmetic.* Let a and b be two vectors of Λ described by their vectors A and B of coordinates in the basis γ. We know that:
$$\langle a, b \rangle = A^T \cdot \mathcal{G}_\gamma \cdot B.$$

Thus:
$$2^n \langle a, b \rangle = A^T \cdot G \cdot B + A^T \cdot \Delta \cdot B, \text{ where } |A^T \cdot \Delta \cdot B| \leq \left(\sum_i |A_i| \right) \left(\sum_i |B_i| \right).$$

This directly gives an interval representation of $\langle a, b \rangle$.

4.3. **Lattice reduction of approximate lattices.** Suppose now that the Gram matrix $\mathcal{G}_\gamma = (\langle \gamma_i, \gamma_j \rangle)_{(i,j) \in [1 \cdots d]^2}$ representing the inner product of the ambient space $\Gamma \otimes_\mathbf{Z} \mathbf{R}$ in the basis γ is given indirectly by an algorithm or an oracle $\mathcal{O}_\gamma$ that can compute each entry at any desired accuracy. We can restate the definition of a reduced basis in this framework as:

Definition 4.3 ((δ, η)-LLL reduction)**.** *Let δ, η be admissible LLL parameters. Given an integral matrix $L \in \mathbf{Z}^{p \times d}$ that describes the vectors of a basis of a lattice Λ in the basis γ, we say that $(\mathcal{G}_\gamma, L)$ is a (δ, η)-LLL reduced basis of Λ if and only if there exists an $n_0 > 0$ such that for any $n \geq n_0$ there exists a pair $(\underline{G}_n, L)$, where $\underline{G}_n$ is an integral representation of $\mathcal{G}_\gamma$ at accuracy n, which is a (δ, η)-LLL reduced basis.*

The computational problem associated with reduction theory can then be written as:

Problem (Lattice Reduction for approximate representation)**.** *Let δ, η be admissible LLL parameters. Given as input an algorithm or oracle to compute $\mathcal{G}_\gamma$ at arbitrary precision and an integral matrix $L \in \mathbf{Z}^{p \times d}$ that describes the vectors of a generating family of a lattice Λ in the basis γ: find a basis L' of Λ such that $(\mathcal{G}_\gamma, L')$ is a (δ, η)-LLL reduced basis in the sense of Definition 4.3.*

Note that using interval arithmetic it suffices to check the (δ, η)-LLL reduction condition at accuracy n_0 to be sure it holds at any larger accuracy. Indeed, an integral representation that satisfies the condition can be refined into a more precise integral representation by scaling up the integer representing the center by an adequate power of two. This refined representation continues to satisfy the condition.

4.3.1. *Accuracy of representation and space complexity.* Let $(\underline{G}_n, L)$ be an integral representation of Λ, at accuracy n. Then, the magnitude of the entries of G is 2^n times the magnitude of the entries of $\mathcal{G}_\gamma$. Thus, $\underline{G}_n$ can be encoded using $\mathrm{O}\big(d^2(n + \log \|\mathcal{G}_\gamma\|_{\max})\big)$ bits.

5. Generalized LLL reduction with Interval Arithmetic

In this Section, we adapt lattice reduction algorithms to our setting. More precisely, we represent the information related to Gram-Schmidt vectors by interval arithmetic using a floating-point representation as described in Section 3.1.3. For the representation of the lattice itself, we consider two cases: either the underlying Gram matrix is integral, or it is given by an approximate integral representation as in Section 4.1. In the latter case, our algorithm also asks for representations with higher accuracy until it is sufficient to yield a reduced basis for the given lattice. The canonical case with the standard Euclidean scalar product is achieved by setting the Gram matrix to the (exact) identity matrix.

5.1. Interval Arithmetic L^2 reduction with fixed precision.

We first consider the simplified case where the lattice representation is fixed. It can be either exact or approximate with a given accuracy. In both cases, we fix a basis $\gamma = (\gamma_1, \ldots, \gamma_d)$ and a representation of a lattice Λ in this basis. It is respectively an exact integral representation (G, L) or an approximate representation $(\underline{G}_n, L)$ at accuracy n of $(\mathcal{G}_\gamma, L)$.

5.1.1. *Using Interval Arithmetic in* LLL. We now modify the L^2 algorithm of [NS09] in a few relevant places to make use of interval arithmetic instead of floating-point arithmetic for the Gram-Schmidt-related values. Since the description of the lattice Λ is already using intervals, it seems natural to use interval arithmetic in the lattice reduction algorithm. For completeness, when the input Gram matrix is exact, we make the updates to the Gram-Schmidt orthogonalized matrix used by LLL explicit in the algorithm (except the simple displacements). This also emphasizes a subtle difference with the case of an approximate input Gram matrix. Indeed, in that case, we update the GSO-values but recompute the errors rather than relying on the interval arithmetic to do it. This is important to gain a fine control on the error growth during updates.

In addition, when using the technique from [Poh87] to be able to deal with lattices given by a generating family instead of a basis, we make a slightly different choice than in [NS09]. Instead of moving the zero vectors that are encountered during the computation during the reduction to the start of the basis, we simply remove them. Note that with an approximate matrix, if we discover a non-zero vector whose length is given by an interval containing 0, it is not possible to continue the computation. This means that the accuracy of the input is insufficient and we abort. The core modification with interval arithmetic appears while testing the Lovász condition. If it is not possible to decide whether the test is true or false because of interval overlap, we also abort due to lack of precision. To be more precise, when testing

the Lovász condition, we also need to check that the corresponding μ coefficient is indeed smaller than η. The reason for this is that, when called with insufficient precision, the Lazy reduction routine may fail to ensure that property.

In addition, if a negative number occurs when computing the norm of a vector, it means that the given Gram matrix is not positive-definite and the algorithm returns an error accordingly.

Algorithm 4: The (interval) lazy size reduction algorithm, η-ILAZYRED.

Input: Initial basis $L = (L_1, \ldots, L_d)$, precomputed (internal) Gram matrix $Gram$, interval matrices $\underline{R}$ and $\underline{M}$, an integer $1 \leq k \leq d$.
Result: Size-reduce the k-th vector of L and update the Gram matrix $Gram$.

1   done $\leftarrow$ **false**;
2   **while** done = **false do**
3   **for** $j = 1$ **to** $k - 1$ **do**
4       $\underline{R_{k,j}} \leftarrow$ CONVERTTOFPINTERVAL$(Gram_{k,j})$;
5   **for** $i = 1$ **to** $j - 1$ **do** $\underline{R_{k,j}} \leftarrow \underline{R_{k,j}} - \underline{M_{j,i}R_{k,i}}$;
6       $\underline{M_{k,j}} \leftarrow \underline{R_{k,j}}/\underline{R_{j,j}}$;
7   **end**
8     $\underline{s_1^{(k)}} \leftarrow$ CONVERTTOFPINTERVAL$(Gram_{k,k})$;
9   **for** $j = 2$ **to** k **do** $\underline{s_j^{(k)}} \leftarrow \underline{s_{j-1}^{(k)}} - \underline{M_{k,j-1}} \cdot \underline{R_{k,j-1}}$;
10    $\underline{R_{k,k}} \leftarrow \underline{s_k^{(k)}}$;
11    $\underline{\tau} \leftarrow (\max_{j<k} \underline{M_{k,j}})$;
12   ret $\leftarrow (\underline{\tau} \leq \eta)$;
13   **if** $ret \neq$ **false then** done $\leftarrow$ **true**;
14   **else**
15   **for** $i = k - 1$ **downto** 1 **do**
16       $X_i \leftarrow \eta$-INTERVALCLOSESTINTEGER$(\underline{M_{k,i}})$;
17   **for** $j = 1$ **to** $i - 1$ **do** $\underline{M_{k,j}} \leftarrow \underline{M_{k,j}} - X_i\underline{M_{i,j}}$;
18       $L_k \leftarrow L_k - X_iL_i$;
       // Update the Gram matrix accordingly
19       $Gram_{k,k} \leftarrow Gram_{k,k} - 2X_iGram_{k,i} + X_i^2Gram_{i,i}$;
20   **for** $j = 1$ **to** i **do** $Gram_{k,j} \leftarrow Gram_{k,j} - X_iGram_{i,j}$;
21   **for** $j = i + 1$ **to** $k - 1$ **do** $Gram_{k,j} \leftarrow Gram_{k,j} - X_iGram_{j,i}$;
22   **for** $j = k + 1$ **to** d **do** $Gram_{j,k} \leftarrow Gram_{j,k} - X_iGram_{j,i}$;
23   **end**
24   **end**
25   **end**

5.1.2. *Internal precision in the exact-input case.* For the classical L^2 algorithm, Section 2.4.1 states that the precision that is needed for the computations only depends on the dimension of the lattice. It is natural to ask a similar question about the algorithm $\overline{\text{LL}}$: can the required internal accuracy be bounded independently of

---

Algorithm 5: The $\overline{\text{LL}}$ Algorithm.

Parameters: $\delta \in (1/4, 1), \eta \in (1/2, \sqrt{\delta})$ admissible LLL parameters, $\ell \in \mathbf{N}$ the internal precision used for floating-point representation.

Input: Exact representation (G, L) or approximate representation $(\underline{G_n}, L)$ of a lattice given by p generating vectors in dimension d.

Result: A (δ, η) LLL-reduced basis L' (with $\dim(L)$ vectors).

---

1 $k \leftarrow 2$;

// Compute the Gram matrix of the basis represented by L

2 **for** $i = 1$ **to** p **for** $j = 1$ **to** i **do**

3   **if** *Exact* **then** $GramL_{i,j} \leftarrow L_i^T G L_j$;

4   **else** $GramL_{i,j} \leftarrow$ Interval of center $L_i^T G_n L_j$ and radius $\|L_i\|_1 \|L_j\|_1$;

5 **end**

6 $\underline{R_{1,1}} \leftarrow \textsc{ConvertToFPinterval}(GramL_{1,1})$;

7 **while** $k \leq p$ **do**

    // Size-reduce L_k with interval on the family $(L_1, \ldots, L_{k-1})$

8 η-ILazyRed(k, *Exact*);

9   **if** *Exact* = **false then for** $j = 1$ **to** k **do**

10   Update radius of $GramL_{k,j}$ to $\|L_k\|_1 \|L_j\|_1$ (rounded up with ℓ significant bits)

11   **end**

12 $k' \leftarrow k$;

13   **while** $k \geq 2$ **do**

14   ret $\leftarrow \left(\underline{M_{k',k-1}} \leq \eta \right)$ and $\left(\underline{\delta} \cdot \underline{R_{k-1,k-1}} > \underline{s_{k-1}^{(k')}} \right)$;

15   **if** *ret* = **true then** $k \leftarrow k - 1$;

16   **else if** *ret* = **false then break**;

17   **else return** ErrorPrecision ;

18   **end**

19   **if** $k \neq k'$ **then**

20   **for** $i = 1$ **to** $k - 1$ **do** $\underline{M_{k,i}} \leftarrow \underline{M_{k',i}}$; $\underline{R_{k,i}} \leftarrow \underline{R_{k',i}}$;

21 $\underline{R_{k,k}} \leftarrow \underline{s_k^{k'}}$;

22 $\overline{L_{tmp}} \leftarrow L_{k'}$;

23   **for** $i = k'$ **downto** $k + 1$ **do** $L_i \leftarrow L_{i-1}$;

24 $L_k \leftarrow L_{tmp}$; Move values in $GramL$ accordingly;

25   **else**

26 $\underline{R_{k,k}} \leftarrow \underline{s_k^{(k')}}$;

27   **if** $0 \in \underline{R_{k,k}}$ *and* $L_k \neq 0$ **then return** ErrorAccuracy ;

28   **if** $\underline{R_{k,k}} < 0$ **then return** ErrorNonPosDefinite ;

29   **end**

30   **if** $L_k = 0$ **then**

       // Remove zero vector from L

31   **for** $i = k$ **to** $p - 1$ **do** $L_i \leftarrow L_{i+1}$;

32 $p \leftarrow p - 1$; $k \leftarrow k - 1$; Move values in $GramL$ accordingly;

33   **end**

34 $k \leftarrow \max(k + 1, 2)$;

35 **end**

36 **return** (L)

---

the entries appearing in the matrices G and L. When G is exact, i.e., integral, the adaptation is straightforward and we obtain the following result.

Theorem 5.1. *Let (δ, η) be admissible* LLL *parameters. Let $c > \log \frac{(1+\eta)^2}{\delta - \eta^2}$ and let $(\Lambda, \langle \cdot, \cdot \rangle)$ denote a rank-d lattice, exactly described by the pair (G, L). Let B denotes the maximum entry in absolute value in $L^T G L$. Then, the $\overline{\text{LL}}$ of Figure 5 used with $\ell = cd + o(d)$ outputs a (δ, η)-*LLL-reduced basis in time $O\big(d^3 \log B(d + \log B)\mathcal{M}(d)\big)$. Furthermore, if τ denotes the number of main loop iterations, the running time is $O(d(\tau + d \log dB)(d + \log B)\mathcal{M}(d))$.*

In fact, the bound on ℓ is made explicit in [NS09]. More precisely, it states that for any arbitrary $C > 0$ and an $\epsilon \in]0, 1/2]$, it suffices to have:

$$\ell \geq 10 + 2\log_2 d - \log_2 \min(\epsilon, \eta - 1/2) + d(C + \log_2 \rho) \quad \text{where } \rho = \frac{(1 + \eta)^2 + \epsilon}{\delta - \eta^2}.$$

For example, choosing $C = \epsilon = \eta - 1/2$ it suffices to have:

$$\ell \geq T(d, \delta, \eta) = 10 + 2\log_2 d - \log_2 (\eta - 1/2) + (\eta - 1/2 + \log_2 \rho)\, d.$$

When δ is close to 1 and η to $1/2$, the constant before d becomes smaller than 1.6.

5.1.3. *Dealing with approximate inputs.* When dealing with lattices given in an approximate form, i.e., by a representation $(\underline{G}_n, L)$ at accuracy n of $(\mathcal{G}_\gamma, L)$, the analysis of the algorithms differs in three main places:

- When bounding the number of rounds τ, we can no longer assume that the potential is an integer. As a consequence, in order to keep a polynomial bound on τ, we need to provide a lower bound on the possible values of the potential, rather than rely on the trivial lower bound of 1 for an integral-valued potential.
- Since the notion of LLL-reduction is only well-defined for a positive definite G, we need to make sure that $\underline{G}_n$ is positive-definite during the algorithm. Otherwise, it should output an error; Algorithm 5 returns an error that $\underline{G}_n$ is incorrect whenever it encounters a vector with a negative norm.
- When $\underline{G}_n$ is approximate, the scalar products between lattice vectors can no longer be exactly computed. Thus, we need to able to make sure that the errors are small enough to be compatible with the inner precision used for Gram-Schmidt values. At first glance, this might seem easy. However, when using update formulas to avoid recomputation of scalar products, the estimates on errors provided by interval arithmetic can grow quite quickly. In fact,it would prevent the update strategy from working. The key insight is to remark that since the centers of the intervals are represented by integers, any computation on them is exact and we can use update formulas to compute them. However, it is essential to recompute the radii of the intervals, i.e., the errors, to prevent them from growing too quickly.

Number of rounds. Since interval arithmetic allows up to emulate exact computations as long as no failures are detected, we can analyze the number of rounds by assuming that all computations on non-integral values are done using an exact arithmetic oracle. In this context, the number of rounds can be studied by considering the potential as usual. Remember that the initial setting where LLL operates

on a basis the potential is defined as

$$D(B) = \prod_{i=1}^{d} \operatorname{covol}\big(B_{[1\ldots i]}\big).$$

The key argument is that it decreases by a multiplicative factor whenever an exchange is performed.

However, in our context, the starting upper bound and the ending lower bound are different from the integer lattice setting. The initial upper bound needs to account from the presence of the positive definite matrix. So if the lattice is described by a pair $(\mathcal{G}_\gamma, L)$ the upper bound becomes:

$$D(B)^2 \leq \big(d^2 \|\mathcal{G}_\gamma\|_{\max} \|L\|_{\max}^2\big)^{d(d+1)/2}.$$

More importantly, it is no longer possible to claim that the potential is an integer. Instead, we derive a lower bound by considering the smallest eigenvalue of $\mathcal{G}_\gamma$ and find:

$$D(B)^2 \geq \lambda_d(\mathcal{G}_\gamma)^{d(d+1)/2}.$$

As a consequence, if we let τ denote the number of rounds of the algorithm, we can conclude that:

$$\tau \leq \mathrm{O}\big(d^2(\log(\|L\|_{\max}) + \log(\|\mathcal{G}_\gamma\|_{\max}/\lambda_d(\mathcal{G}_\gamma)) + \log(d)\big).$$

When the lattice is given by a generating family L rather than a basis B, we need a slightly different invariant. Following [NS09], we define d_i to be the product of the first i non-zero values $\|b_j^*\|$. Note that they are not necessarily consecutive, since zeroes may occur anywhere. We then let:

$$D'(L) = \left(\prod_{i=1}^{\dim L} d_i\right) \cdot \left(\prod_{i, b_i^* = 0} 2^i\right).$$

This generalized potential is needed for the proof of Theorem 5.2. Note that, for lattices given by a basis, the two definitions coincide.

Necessary accuracy for the scalar products. In order to preserve the correctness of the algorithm when computing with internal precision ℓ, we need to check that all conversions of scalar product values, using the calls to CONVERTTOFPINTERVAL in Algorithms 4 and 5, have sufficient precision. For a pair of lattice elements, described by vectors L_i and L_j, the relative precision on the value of their scalar product is:

$$\frac{\|L_i\|_1 \|L_j\|_1}{|L_i^T G_n L_j|}.$$

When the vectors are close to orthogonal with respect to the scalar product given by G_n, the error can be arbitrarily large. However, by carefully following the analysis of Theorem 3 in [NS09, Section 4.1], we can show that this Theorem remains true in our context. This suffices to ensure the correctness part of Theorem 5 of [NS09]. The first check is to verify that quantity called err_1 in the proof of the Theorem remains upper bounded by $2^{-\ell}$. Since the value is defined as the error on the scalar product of the vectors number i and 1 divided by the norm of the first vector, we have:

$$err_1 \leq \frac{\|L_i\|_1 \|L_1\|_1}{|L_1^T G_n L_1|} \leq \frac{\max_i \|L_i\|_1^2}{\lambda_d(G_n)} \leq \frac{d \max_i \|L_i\|^2}{\lambda_d(G_n)} \leq \frac{d \max_i \|b_i\|^2}{\lambda_d(G_n)^2}.$$

Thus:

$$err_1 \leq \frac{d^3 \|G_n\|_{\max} \|L\|_{\max}^2}{\lambda_d(G_n)^2} \leq \frac{d^3(2^n \|\mathcal{G}_\gamma\|_{\max} + 1)\|L\|_{\max}^2}{(2^n \lambda_d(\mathcal{G}_\gamma) - 2d)^2}.$$

As a consequence, it suffices to have:

$$n \geq \ell + \mathrm{O}(\log(\|L\|_{\max}) + \log(\|\mathcal{G}_\gamma\|_{\max}/\lambda_d(\mathcal{G}_\gamma)) + \log(d)).$$

L^2 *with approximate inputs.* To complete the above properties on the number of rounds and necessary accuracy, it suffices to remark that the only additional line of code in the approximate L^2 is the recomputation of interval radii on line 10. Since it suffices to know the ℓ high-order bits of the values, this recomputation can fully be done using arithmetic on ℓ. Indeed, during the computations of $\|L_i\|_1$ no cancellation occurs. As a consequence, we get the following adaptation of Theorem 5.1. For completeness, we give here the case where the lattice is initially given by a generating family of p vectors, has rank d and lives in an ambient space of dimension D.

Theorem 5.2. *Let (δ, η) be such that $1/4 < \delta < 1$ and $1/2 < \eta < \sqrt{\delta}$. Let $c > \log \frac{(1+\eta)^2}{\delta - \eta^2}$. Assume that we are given as input $(\Lambda, \langle \cdot, \cdot \rangle)$ a rank-d lattice $(\mathcal{G}, L)$ described by $p \geq d$ generating vectors in a ambient space of dimension $D \geq d$. Further assume that it is approximately represented at accurary N by the pair $(\underline{G_N}, L)$ and let B denote the maximum entry in absolute value in $L^T \mathcal{G} L$. Let $\ell = cd + o(d)$ and*

$$N \geq \ell + \log(B/\lambda_D(\mathcal{G})) + \log(d).$$

Then, the $\overline{\mathrm{LL}}$ of Figure 5 outputs a (δ, η)-LLL-reduced basis in time

$$O\big(DN\big(d^2 N + p(p - d)\big)\mathcal{M}(d)\big).$$

Furthermore, if τ denotes the number of main loop iterations, the running time is $O(DN(dN + \tau)\mathcal{M}(d))$.

5.2. L^2 reduction with adaptive precision and accuracy.

5.2.1. *Adaptive precision.* Since by construction the $\overline{\mathrm{LL}}$ Algorithm can detect that the choice for internal precision ℓ is insufficient to correctly reduce the lattice Λ. The procedure can be wrapped in a loop that geometrically increases precision ℓ after each unsuccessful iteration. This yields an *adaptive precision* reduction algorithm ADAPTIVE-LLL. Since the complexity of floating-point multiplication is superlinear, the use of a geometric precision growth guarantees that the total complexity of this lattice reduction is asymptotically dominated by its final iteration.[3]

Moreover, the cost of operations in the floating-point realization of interval arithmetic is at most four times the cost of floating-point arithmetic at the same precision. Depending on the internal representation used, this constant can even be improved. As a consequence, for lattices that can be reduced with a low-enough precision, it can be faster to use interval arithmetic than floating-point arithmetic with the precision required by the bound from Section 2.4.1.

---

[3]In practice, for lattices of rank few hundreds it appears nonetheless that the computational cost of the previous iterations lies between 20% and 40% of the total cost.

5.2.2. *Adaptive accuracy.* We now turn to the setting of Section 4.3, where an algorithm or oracle $\mathcal{O}_\gamma$ can output an integral representation of the Gram matrix $\mathcal{G}_\gamma = (\langle \gamma_i, \gamma_j \rangle)_{(i,j) \in [1 \cdots r]^2}$ at arbitrary accuracy n. In that context, we need to determine both the necessary accuracy and internal precision. When running Algorithm 5 with some given accuracy and precision, three outcomes are possible:

- Either the reduction terminates in which case the lattice is LLL-reduced, which implies that both accuracy and precision are sufficient.
- The Lovász condition fails to be tested correctly, which indicates an insufficient precision. In that case, we need to test whether the precision is lower than theoretical bound $T(d, \delta, \eta)$ given after Theorem 5.1 or not. In the latter case, we know that the accuracy needs to be increased.
- The algorithm detects a non-zero vector whose norm is given by an interval containing 0. This directly indicates insufficient accuracy.

Depending on the result of Algorithm 5, we increase the precision or the accuracy and restart. The corresponding pseudo-code is given in Algorithm 6. Since the precision and accuracy both follow a geometric growth, the computation is dominated by its final iteration. In particular, we may use the complexity bound given by Theorem 5.2.

Note that when we increase the accuracy in Algorithm 6, we also reset the precision to its minimal value. This is a matter of preference that doesn't affect the asymptotic complexity. In practice, it seems to be preferable.

It is important to note that we do need to precompute the eigenvalues of the Gram matrix, since Algorithm 6 automatically detects the needed accuracy.

5.3. **Possible generalizations.** The adaptative strategy we describe for LLL can be generalized to other lattice reduction algorithm. In particular, enumeration algorithms are possible within our framework, which allows the implementation of the BKZ algorithm of [Sch87].

It would be interesting to study a generalization to sieving techniques to adapt them to approximate lattices.

6. Application to Algebraic Number Theory

We now present a direct application of our lattice reduction strategy in algorithmic number theory. Namely, we consider some interesting lattices sitting inside number fields: *ideal lattices.*

6.1. **Number fields, integers and ideal lattices.**

Number fields. A number field $\mathbf{K}$ is a finite-dimensional algebraic extension of $\mathbf{Q}$. It can be described as:
$$\mathbf{K} \cong \mathbf{Q}[X]\big/(P) = \mathbf{Q}(\alpha),$$
where P is a monic irreducible polynomial of degree d in $\mathbf{Z}[X]$ and where α denotes the image of X in the quotient.

Let $(\alpha_1, \ldots, \alpha_d) \in \mathbf{C}^d$ denote the distinct complex roots of P. Then, there are d distinct ring-embeddings of $\mathbf{K}$ in $\mathbf{C}$. We define the i-th embedding $\sigma_i : \mathbf{K} \to \mathbf{C}$ as the field homomorphism sending α to α_i.

It is classical to distinguish embeddings induced by real roots, a.k.a., *real embeddings* from embeddings coming from (pairs of conjugate) complex roots, called *complex embeddings.* Those arising from complex roots called complex embeddings.

Algorithm 6: The ADAPTIVE-LLL algorithm.

Parameters: $\delta \in (1/4, 1), \eta \in (1/2, \sqrt{\delta}), \ell_0 \in \mathbf{N}$ initial precision of the
algorithm for floating-point representation, n_0 initial
accuracy for representing the scalar product, $g > 1$ geometric
growth factor.

Input: γ a basis of a lattice $(\Gamma, \langle \cdot, \cdot \rangle)$, and $\mathcal{O}_\gamma(n)$ an oracle that compute
the integral representation of the inner product $\langle \cdot, \cdot \rangle$ at accuracy n.

Input: A generating family represented by L in γ of a sublattice $\Lambda \subset \Gamma$.

Result: A (δ, η) LLL-reduced basis of Λ represented as $L' \in \mathbf{Z}^{\mathrm{rk}(\Lambda) \times \mathrm{rk}(\Lambda)}$.

```
// Set initial values for accuracy and precision
// T(d, δ, η) is the theoretical bound given after Theorem 5.1
```

1 $\ell \leftarrow \ell_0$;

2 $n \leftarrow n_0$;

3 $G \leftarrow \mathcal{O}_\gamma(n)$;

4 succeed $\leftarrow$ **false**;

5 **repeat**

6   retcode $\leftarrow \overline{\mathrm{LL}}(G, L)$;

7   **if** *retcode=ErrorNonPosDefinite* **then return** ErrorNonPosDefinite;

8   **if** *retcode=*OK **then** succeed $\leftarrow$ **true**;

9   **else if** *retcode=ErrorPrecision* **then**

10 $\ell' \leftarrow \ell$;

11 $\ell \leftarrow \min(\lceil g\,\ell \rceil, T(d, \delta, \eta), n)$;

12   **if** $\ell' = \ell$ **then** retcode $\leftarrow$ ErrorAccuracy;

13   **end**

14   **if** *retcode=ErrorAccuracy* **then**

15 $\ell \leftarrow \ell_0$;

16 $n \leftarrow \lceil g\,n \rceil$;

17 $G \leftarrow \mathcal{O}_\gamma(n)$;

18   **end**

19 **until** *succeed* $=$ **true**;

20 **return** L

Assume that P has r_1 real roots and r_2 pairs of conjugate complex roots, with $d = r_1 + 2r_2$. Since the embeddings corresponding to conjugate roots are related by conjugation on $\mathbf{C}$, we can either keep a single complex root in each pair or replace each pair by the real and imaginary part of the chosen root. This leads to the *Archimedean embedding* σ defined as:

$$
\begin{array}{rrl}
\sigma: & \mathbf{K} & \longrightarrow \quad \mathbf{R}^d \\
& x & \longmapsto \quad \left(\sigma_1(x), \ldots, \sigma_{r_1}(x), \sqrt{2}\Re(\sigma_{r_1+1}(x)), \sqrt{2}\Im(\sigma_{r_1+1}(x)), \ldots \right)^T
\end{array}
$$

This embedding allows us to define a real symmetric bilinear form on $\mathbf{K}$:

$$
\langle a, b \rangle_\sigma = \sigma(a) \cdot \sigma(b) = \sum_{i=1}^{d} \sigma_i(a)\overline{\sigma_i(b)}.
$$

The second equality explains the presence of the normalization factors $\sqrt{2}$ in the definition of σ. Note that the form is positive definite, thus endowing $\mathbf{K}$ with an Euclidean structure.

Integers. Any element γ of $\mathbf{K}$ has a minimal polynomial, defined as the unique monic polynomial of least degree among all polynomials of $\mathbf{Q}[X]$ vanishing at γ. The algebraic number γ is said to be *integral* if its minimal polynomial lies in $\mathbf{Z}[X]$. The set of all integers in $\mathbf{K}$ forms a ring, called the *ring of integers* of $\mathbf{K}$ and denoted $\mathfrak{o}_\mathbf{K}$. It is also a free $\mathbf{Z}$-module of rank d. A basis $(w_1, \dots, w_d)$ of $\mathfrak{o}_\mathbf{K}$ (as a $\mathbf{Z}$-module) is called an integral basis of $\mathbf{K}$.

As a consequence, using the bilinear form $\langle \cdot, \cdot \rangle_\sigma$, we can view $\mathfrak{o}_\mathbf{K}$ as a lattice.

Ideals. An ideal of $\mathfrak{o}_\mathbf{K}$ is defined as an $\mathfrak{o}_\mathbf{K}$-submodule of $\mathfrak{o}_\mathbf{K}$. In particular, it is a $\mathbf{Z}$-submodule of rank d. Every ideal I can be described by a two-element representation, i.e. expressed as $I = \alpha\mathfrak{o}_\mathbf{K} + \beta\mathfrak{o}_\mathbf{K}$, with α and β in $\mathfrak{o}_\mathbf{K}$. Alternatively, every ideal can also be described by a $\mathbf{Z}$-basis formed of d elements.

6.2. **Lattice reduction for ideals.** With the above notations, we can directly use our lattice reduction algorithm to reduce an ideal lattice. More precisely, given an integral basis $(w_1, \dots, w_d)$ and a two-element representation of I by α and β, we proceed as follows:

(1) Define the Gram matrix $\mathcal{G}_w$ with entries $\langle w_i, w_j \rangle_\sigma$. It can be computed to any desired precision from approximations of the roots of P. The roots themselves can be computed, using, for example, the Gourdon-Schönhage algorithm [Gou96].
(2) Let L be the matrix formed of the (integral) coordinates of $(\alpha w_1, \dots, \alpha w_d)$ and $(\beta w_1, \dots, \beta w_d)$ in the basis $(w_1, \dots, w_d)$.
(3) Directly apply Algorithm 6 to $(\mathcal{G}_w, L)$.

The same thing can be done, *mutantis mudantis*, for an ideal described by a $\mathbf{Z}$-basis.

A well-known special case. For some number fields, the Gram matrix is $\mathcal{G}_w$ is integral. In that case, the use of Algorithm 6 isn't necessary and one can directly work with an exact lattice. This is described for the special case of reducing the full lattice corresponding to the ring of integers in [Bel04, Section 4.2] for totally real fields. It can be generalized to CM-fields, since they satisfy the same essential property of having an integral Gram matrix. The same application is also discussed in [Coh93, Section 4.4.2].

Non integral case. For the general case where the Gram matrix is real, [Bel04] propose to multiply by 2^e and round to the closest integer. It also gives a bound on the necessary accuracy e as the logarithm of (the inverse of) the smallest diagonal entry in the Cholesky decomposition of the Gram matrix. In some sense, this is similar to our approach. However, without any auxiliary information on this coefficient, it is proposed to continue *increasing e as long as it is deemed unsatisfactory.*

By contrast, termination of our algorithm guarantees that lattice reduction is completed and that the output basis is LLL-reduced.

References

[Bel04]   Karim Belabas. Topics in computational algebraic number theory. *J. Théor. Nombres Bordeaux*, 16:19–63, 2004.

[BF13]   Jean-François Biasse and Claus Fieker. Improved techniques for computing the ideal class group and a system of fundamental units in number fields. *The Open Book Series*, 1(1):113–133, 2013.

[Buc94]   Johannes A. Buchmann. Reducing lattice bases by means of approximations. In *Algorithmic Number Theory, First International Symposium, ANTS-I, Ithaca, NY, USA, May 6-9, 1994, Proceedings*, pages 160–168, 1994.

[Coh93]   Henri Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[DW87]   Benne M. M. De Weger. Solving exponential Diophantine equations using lattice basis reduction algorithms. *J. Number theory*, 26:325–367, 1987.

[Elk00]   Noam D. Elkies. Rational points near curves and small nonzero $|x^3 - y^2|$ via lattice reduction. *Algorithmic Number Theory: 4th International Symposium, ANTS-IV Leiden, The Netherlands, July 2-7, 2000. Proceedings*, pages 33–63, 2000.

[GJ16]   Alexandre Gélin and Antoine Joux. Reducing number field defining polynomials: an application to class group computations. In *Algorithmic Number Theory Symposium XII*, volume 19 of *LMS Journal of Computation and Mathematics*, pages 315–331, 2016.

[Gou96]   Xavier Gourdon. Combinatoire, algorithmique et géométrie des polynomes. *PhD thesis*, pages 27–49, 1996.

[HMM98]   George Havas, Bohdan S. Majewski, and Keith R. Matthews. Extended GCD and Hermite normal form algorithms via lattice basis reduction. *Experimental Mathematics*, 7(2):125–136, 1998.

[Jäg05]   Gerold Jäger. Reduction of Smith normal form transformation matrices. *Computing*, 74(4):377–388, 2005.

[JKDW01] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics.* Springer Verlag, 2001.

[Kal83]   Erich Kaltofen. On the complexity of finding short vectors in integer lattices. In J. A. van Hulzen, editor, *Computer Algebra, EUROCAL '83, European Computer Algebra Conference*, volume 162 of *Lecture Notes in Computer Science*, pages 236–244. Springer, 1983.

[KV16]   Seungki Kim and Akshay Venkatesh. The behavior of random reduced bases, 2016.

[Len83]   Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.

[LLL82]   Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[LS92]   László Lovász and Herbert E. Scarf. The generalized basis reduction algorithm. *Math. Oper. Res.*, 17(3):751–764, 1992.

[LS17]   Hendrik W. Lenstra, Jr. and Alice Silverberg. Lattices with symmetry. *Journal of Cryptology*, 30(3):760–804, Jul 2017.

[Moo62] Ramon E. Moore. *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Stanford, 1962.

[Moo77] Ramon E. Moore. *Methods and applications of interval analysis*. 1977.

[Ngu10] Phong. Q. Nguyen. Hermite's constant and lattice algorithms. In Phong. Q. Nguyen and Brigitte Vallée, editors, *The LLL algorithm*, chapter 2. Springer, 2010.

[NS09] Phong. Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. of Computing*, 39(3):874—-903, 2009.

[Poh87] Michael E. Pohst. A modification of the LLL reduction algorithm. *Journal of Symbolic Computation*, 4(1):123–127, 1987.

[PT08] Gabor Pataki and Mustafa Tural. On sublattice determinants in reduced bases, 2008.

[RR88] Helmut Ratschek and Jon Rokne. *New computer methods for global optimization*. Halsted Press, New York, NY, USA, 1988.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[Sch88] Claus-Peter Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9(1):47–62, 1988.

[SE94] Claus-Peter Schnorr and Michael Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[Sun09] Teruo Sunaga. Theory of an interval algebra and its application to numerical analysis. *Japan J. Indust. Appl. Math.*, 26, 10 2009.

[Tea16] The FPLLL Development Team. fplll, a lattice reduction library. Available at https://github.com/fplll/fplll, 2016.

[Vil07] Gilles Villard. Certification of the qr factor r, and of lattice basis reducedness, 2007.

[You31] Rosalind C. Young. *The algebra of many-values quantities*. PhD thesis, Cambridge, 1931.

Appendix A. Proof of Proposition 2.1

We now show the more general statement for a (δ, η)-LLL reduced basis $(b_1, \ldots, b_d)$ of $(\Lambda, \langle \cdot, \cdot \rangle)$. Namely that for any $1 \leq k \leq d$ we have:

$$\mathrm{covol}(b_1, \ldots, b_k) \leq \left(\delta - \eta^2\right)^{-\frac{(d-k)k}{4}} \mathrm{covol}(\Lambda)^{\frac{k}{d}}.$$

Proof. Using the Lovász condition at index $1 \leq i < d$, we write:

$$\delta \|\pi_i(b_i)\|^2 \leq \|\pi_i(b_{i+1})\|^2 = \|\pi_{i+1}(b_{i+1})\|^2 + \mu_{i,i+1}^2 \|\pi_i(b_i)\|^2$$

Thanks to the size-reduction condition, this implies:

$$(4) \qquad \forall i \in \{1, \ldots, d-1\}, \quad \|\pi_i(b_i)\|^2 \leq \left(\delta - \eta^2\right)^{-1} \|\pi_{i+1}(b_{i+1})\|^2.$$

Let K denote $\left(\delta - \eta^2\right)^{-1/2}$ and ℓ_i be the norm of the vector $\pi_i(b_i)$. Then, Equation (4) becomes:

$$\forall i \in \{1, \ldots, d-1\}, \quad \ell_i \leq K \ell_{i+1}.$$

Recall that $\text{covol}(b_1, \ldots, b_k) = \prod_{i=1}^{k} \ell_i$. This implies that for any $j > k$:

$$\text{covol}(b_1, \ldots, b_k) \leq \prod_{i=1}^{k} K^{j-i} \ell_j = K^{k(2j-k-1)/2} \cdot \ell_j^k.$$

Thus:

$$\text{covol}(b_1, \ldots, b_k)^d = \left(\prod_{i=1}^{k} \ell_i\right)^d \leq \left(\prod_{i=1}^{k} \ell_i\right)^k \prod_{j=k+1}^{d} K^{k(2j-k-1)/2} \cdot \ell_j^k$$

$$\leq \left(\prod_{i=1}^{d} \ell_i\right)^k K^{\sum_{j=k+1}^{d} k(2j-k-1)/2}$$

$$\leq \text{covol}(\Lambda)^k K^{\frac{d(d-k)k}{2}}.$$

$\square$

Sorbonne Université, LIP 6, CNRS UMR 7606, Paris, France
Email address: t.espitau@gmail.com

Chaire de Cryptologie de la Fondation SU, Sorbonne Université, Institut de Mathématiques de Jussieu–Paris Rive Gauche, CNRS, INRIA, Univ Paris Diderot. Campus Pierre et Marie Curie, F-75005 Paris, France
Email address: antoine.joux@m4x.org