Acta Cryst A
FOUNDATIONS
ADVANCES

ISSN 2053-2733

A new method for lattice reduction using directional and hyperplanar shearing

Cyril Cayron*

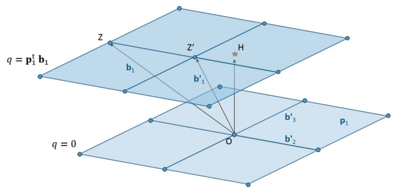
Laboratory of Thermo Mechanical Metallurgy (LMTM), PX Group Chair, EPFL, Rue de la Maladière 71b, Neuchâtel, 2000, Switzerland. *Correspondence e-mail: cyril.cayron@epfl.ch

Received 26 January 2021
Accepted 21 October 2021

Edited by L. Palatinus, Czech Academy of Sciences, Czech Republic

Keywords: lattice reduction; hyperplane; left inverse; algorithm.

A geometric method of lattice reduction based on cycles of directional and hyperplanar shears is presented. The deviation from cubicity at each step of the reduction is evaluated by a parameter called 'basis rhombicity' which is the sum of the absolute values of the elements of the metric tensor associated with the basis. The levels of reduction are quite similar to those obtained with the Lenstra–Lenstra–Lovász (LLL) algorithm, at least up to the moderate dimensions that have been tested (lower than 20). The method can be used to reduce unit cells attached to given hyperplanes.



1. Introduction

In a recent paper (Cayron, 2021), we proposed a method to determine a unit cell attached to any hyperplane $\mathbf{p}$. A hyperplane $\mathbf{p}$ is a plane of dimension $N-1$ in a space of dimension N. Its Miller indices p_i permit it to be built geometrically in the direct basis by considering its intersection points with the ith axes (in $1/p_i$). Equivalently the letter $\mathbf{p}$ represents the vector of coordinates p_i in the reciprocal basis; this vector is normal to the hyperplane. The unit cell attached to the hyperplane $\mathbf{p}$ is made of one short vector $\mathbf{b}_1$ pointing to a node of the first layer parallel to the plane $\mathbf{p}$, *i.e.* such that the scalar product $\mathbf{p}^{\mathrm{t}}\mathbf{b}_1 = 1$, and of $N-1$ short vectors $\{\mathbf{b}_2, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ lying in the plane $\mathbf{p}$, *i.e.* such that the scalar product $\mathbf{p}^{\mathrm{t}}\mathbf{b}_i = 0$, where 't' means 'transpose'. The first vector is a solution of Bézout's identity, and the $N-1$ vectors are solutions of the integer relation, both with the coordinates p_i. Even if the vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ determined by the algorithm are already quite short, they can be reduced even more, *i.e.* it is possible to find shorter vectors $\{\mathbf{b}'_1, \ldots, \mathbf{b}'_i, \ldots, \mathbf{b}'_N\}$ defining a smaller and more orthogonal unit cell of the same volume associated with the same hyperplane $\mathbf{p}$, *i.e.* fulfilling the same Bézout's identity and integer relation. Reducing the length of the vectors in a lattice is related to the general problem called 'lattice reduction'.

Let us explain it in a general way. Given a lattice $\mathcal{L}$ spanned (freely) by N vectors $\mathbf{b}_i$, lattice reduction consists of finding new relatively short, nearly orthogonal vectors $\mathbf{b}'_i$ spanning the same lattice $\mathcal{L}$. The reduced and initial bases are linked by integers z_{ij} such that $\mathbf{b}'_i = \sum_{j=1}^{N} z_{ij}\mathbf{b}_j$ and $\mathcal{L} = \{\mathbb{Z}\mathbf{b}'_i\} = \{\mathbb{Z}\mathbf{b}_j\}$, where the $\{\mathbb{Z}\}$ means all linear combinations with integer coefficients. The number of vectors cannot be larger than the space dimension. The coefficients z_{ij} form a unimodular matrix $\mathbf{Z}$ (integer matrix of determinant ± 1), and the relation between the vectors of the bases is

$$\begin{bmatrix} \mathbf{b}'_1 \\ \vdots \\ \mathbf{b}'_i \\ \vdots \\ \mathbf{b}'_N \end{bmatrix} = \mathbf{Z} \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_j \\ \vdots \\ \mathbf{b}_N \end{bmatrix}$$

with $\mathbf{Z} \in \mathbb{Z}^{NN}$ and $\det(\mathbf{Z}) = \pm 1$, where $\mathbf{b}'_i$ and $\mathbf{b}_j$ refer to the vectors themselves, not to their coordinates. Strictly speaking, it is the basis whose vectors generate the lattice that is reduced at constant volume, and not the lattice itself since this remains the same. The most popular algorithm to determine a reduced basis is the Lenstra–Lenstra–Lovász (LLL) algorithm, which relies on Gram–Schmidt orthogonalization (Appendix *A*). It is usual in lattice reduction problems to present the vectors $\mathbf{b}_j$ as the rows of a matrix. In crystallography, we generally write the coordinates in columns and keep the row notation for planes, *i.e.* for vectors of the reciprocal space. In order to avoid any confusion, we will write $\mathbf{b}^{\rm t}$ for a row vector $\mathbf{b}$. With this notation, in a space of dimension N, a vector $\mathbf{b}$ is an $N \times 1$ matrix, and $\mathbf{b}^{\rm t}$ a $1 \times N$ matrix. All the vectors in this paper are written in a Cartesian orthonormal basis and their coordinates are integers. The relation between the reduced and initial bases can be written in the form of a matrix product $\mathbf{B}' = \mathbf{ZB}$, where

$$\mathbf{B}' = \begin{bmatrix} \mathbf{b}'^{\rm t}_1 \\ \vdots \\ \mathbf{b}'^{\rm t}_i \\ \vdots \\ \mathbf{b}'^{\rm t}_N \end{bmatrix}$$

and

$$\mathbf{B} = \begin{bmatrix} \mathbf{b}^{\rm t}_1 \\ \vdots \\ \mathbf{b}^{\rm t}_i \\ \vdots \\ \mathbf{b}^{\rm t}_N \end{bmatrix}$$

are matrices of $\mathbb{Z}^{NN}$. A typical low-dimensional example of lattice reduction is the set of three vectors in three dimensions, $\mathbf{b}^{\rm t}_1 = [1, 1, 1]$, $\mathbf{b}^{\rm t}_2 = [-1, 0, 2]$ and $\mathbf{b}^{\rm t}_3 = [3, 5, 6]$. They form the matrix

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 2 \\ 3 & 5 & 6 \end{bmatrix}.$$

The reduced lattice basis is

$$\mathbf{B}' = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ -1 & 0 & 2 \end{bmatrix}.$$

One can check that the three row vectors are $\mathbf{b}'_1 = -4\mathbf{b}_1 - \mathbf{b}_2 + \mathbf{b}_3$, $\mathbf{b}'_2 = 5\mathbf{b}_1 + \mathbf{b}_2 - \mathbf{b}_3$ and $\mathbf{b}'_3 = \mathbf{b}_2$. The integer coefficients of linearity could be found by calculating $\mathbf{Z} = \mathbf{B}'\mathbf{B}^{-1}$.

A direct lattice reduction algorithm, such as LLL, permits the lattice to be reduced but does not preserve the unit cell attached to a given hyperplane $\mathbf{p}$. We are thus looking for an intermediate method such that the vector $\mathbf{b}'_1$ continues to point towards a node of the first layer, and the other vectors $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_i, \ldots, \mathbf{b}'_N\}$ remain in the hyperplane $\mathbf{p}$. An intuitive solution to reduce $\mathbf{b}_1$ consists of applying a simple shear parallel to the hyperplane $\mathbf{p}$, as illustrated in Fig. 4 of Cayron (2021). One could then think of applying the LLL algorithm to reduce the other vectors $\{\mathbf{b}_2, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ lying in the hyperplane $\mathbf{p}$, but it is also possible to reverse the problem and use the intermediate simple shear method to develop a simple geometric algorithm of lattice reduction. This method, which we call 'cubification', is different from LLL because it does not require Gram–Schmidt orthogonalization. It is also well adapted to determine a reduced unit cell attached to a hyperplane $\mathbf{p}$, as shown by Cayron (2021). It consists of applying simple shears parallel to the directions and to the hyperplanes of the lattice. Here, the term 'shear' should be understood in its general meaning: for a vector space $\mathcal{V}$ and a subspace $\mathcal{W}$, a shear of a vector $\mathbf{v} \in \mathcal{V}$ fixing $\mathcal{W}$ translates $\mathbf{v}$ in a direction parallel to $\mathcal{W}$. If $\mathcal{V}$ is the direct sum $\mathcal{V} = \mathcal{W} \oplus \mathcal{W}'$, we write $\mathbf{v} = \mathbf{w} + \mathbf{w}'$, then the image of $\mathbf{v}$ by the shear S is $S(\mathbf{v}) = \mathbf{w} + \mathbf{w}' + M(\mathbf{w}')$ where M is a linear mapping from $\mathcal{W}'$ onto $\mathcal{W}$. Directional and hyperplanar shears correspond to the case where the dimensions of the subspaces $\mathcal{W}$ are 1 and $N-1$, respectively.

In general, a parameter called 'orthogonality defect' is used to evaluate the degree of reduction. It is defined by P/V where P is the product of the norms of the basis vectors, $P = \prod_{i \le N} \|\mathbf{b}_i\|$, and V is the volume of the cell formed by the vectors, $V = \det(\mathbf{b}_1, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N)$, which is an invariant of the reduction process. Another parameter to evaluate the norms could be $S = \sum_{i \le N} \|\mathbf{b}_i\|^2 = \sum_{i \le N} \mathbf{b}^{\rm t}_i \mathbf{b}_i$. In this paper, instead of using P/V, the degree of 'cubicity' of a basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ will be evaluated by calculating the 'basis rhombicity' defined from the Euclidean scalar products between the vectors:

$$R = \sum_{i,j} |\mathcal{M}_{ij}| = \sum_{i \le N} \|\mathbf{b}_i\|^2 + 2 \sum_{i<j \le N} \|\mathbf{b}^{\rm t}_i \mathbf{b}_j\| \qquad (1)$$

where $\mathcal{M}$ is the metric tensor given

$$\mathcal{M} = \begin{bmatrix} \mathbf{b}^{\rm t}_1 \\ \vdots \\ \mathbf{b}^{\rm t}_i \\ \vdots \\ \mathbf{b}^{\rm t}_N \end{bmatrix} (\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N)$$

$$= \begin{bmatrix} \mathbf{b}^{\rm t}_1\mathbf{b}_1 & \ldots & \mathbf{b}^{\rm t}_1\mathbf{b}_j & \ldots & \mathbf{b}^{\rm t}_1\mathbf{b}_N \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ \mathbf{b}^{\rm t}_i\mathbf{b}_1 & \ldots & \mathbf{b}^{\rm t}_i\mathbf{b}_j & \ldots & \mathbf{b}^{\rm t}_i\mathbf{b}_N \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ \mathbf{b}^{\rm t}_N\mathbf{b}_1 & \ldots & \mathbf{b}^{\rm t}_N\mathbf{b}_j & \ldots & \mathbf{b}^{\rm t}_N\mathbf{b}_N \end{bmatrix}.$$

The ‘basis rhombicity’ contains the information on both the norms and the angles between the vectors. A lowest ‘rhombicity’ indicates a more cubic cell. Note that the term ‘rhombicity’ has a specific meaning in a branch of mathematics that deals with symmetric second-rank tensors in three-dimensional Euclidean space, but that is not the one given in the present paper. The ‘basis rhombicity’ R was preferred to the parameter P/V for two reasons:

(*a*) From a theoretical point of view, although it seems to be common knowledge, we realized that minimizing the norms of the vectors in high dimensions is not exactly equivalent to improving the orthogonality between them. The reader can look at the simple example in dimension 4 in Appendix *B*, which presents two bases of the same lattice, with the same norms S and P, but one with a better orthogonality, *i.e.* lower R, than the other one. We will also show in Section 4.3 an example in dimension 20 in which, for the same lattice, one reduced basis has a better orthogonality (lower R) but a worse norm (larger P and S) than another reduced basis.

(*b*) From a practical point of view, we noticed that the ‘cubification’ method leads to lower norms in terms of P or S when R is used as driving criterion, and not P or S themselves.

Fig. 1(*a*) gives an example of lattice reduction with the LLL algorithm. The matrix representing the basis to be reduced is nearly the identity except that the last column containing the Nth coordinates of the vectors is constituted of relatively high integers. These types of matrices are often used because they appear in the ‘knapsack problems’ (given a set of items, each with a weight and a value, one has to determine the number of each item to include in the knapsack so that the total weight should not exceed a limit and the value is maximized). Geometrically, the initial basis is highly elongated along the Nth axis. Its initial values are R = 453988268, S = 61580172. They decrease to R = 531, S = 99 with the LLL-reduced basis given in Fig. 1(*b*).

The principle of directional shear will be presented in Section 2. It helps to obtain a reduced lattice with significantly lower R and S values, although higher than with LLL. The hyperplanar shear will be explained in Section 3; it permits R and S to be decreased further. In Section 4, it will be shown how cycling directional and hyperplanar shears permits values of R and S to be obtained that are comparable with those of LLL.

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&75\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&436\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1586\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1030\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1921\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&569\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&-721\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1183\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1570\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&-6665\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&123\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&890\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&6\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&742\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&33\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&888\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&14\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&769\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1234\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&-852
\end{bmatrix}$$

(*a*)

$$\begin{bmatrix}
-1&0&-1&1&1&-1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&-1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&-1&0&1&0&-1&1&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&-1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&-1\\
0&0&-1&0&0&-1&1&1&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&-1&-1&0&0&1&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&1&-1&1&1&0&0&0&0&0&1&0&0&0&0&0&0\\
-1&0&0&0&0&-1&0&-1&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&0&0&-1&0&0&1&1&0&0&0&0&0&0&0\\
0&-1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&-1\\
0&-1&1&0&0&0&0&-1&0&0&0&0&0&0&1&0&0&0&0&0\\
0&-1&0&1&0&0&0&0&-1&0&1&0&0&0&0&0&0&0&0&-1\\
-1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&-1\\
0&1&0&0&0&0&0&1&0&0&1&-1&0&0&0&0&0&0&0&0\\
0&0&0&-1&0&1&-1&-1&0&0&0&1&0&0&1&0&0&0&0&0\\
-1&0&0&1&0&0&0&0&0&0&-1&0&1&0&0&0&1&0&0&0\\
0&0&0&0&-1&0&-1&-1&0&0&-1&0&0&0&1&0&0&1&0&0\\
-1&0&-1&0&-1&-1&0&0&0&0&0&0&0&1&0&0&0&0&0&-1\\
0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&1\\
-1&0&0&1&0&0&-1&0&0&1&-1&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

(*b*)

Figure 1
Example of the LLL algorithm with a 20×20 matrix representing a list of 20 vectors whose coordinates are written in rows. (*a*) Input list. (*b*) Output list determined with the function *LatticeReduce* of *Mathematica*. (*a*) Basis before reduction; the values of the rhombicity $R = \sum_N \|\mathbf{b}_i\|^2 + 2\sum_{i<j\leq N} \|\mathbf{b}_i^{\rm t}\mathbf{b}_j\|$ and of the sum of the squares of the norms $S = \sum_N \|\mathbf{b}_i\|^2$ are R = 453988268, S = 61580172. (*b*) Basis after reduction; the parameters decreased to R = 531, S = 99.

2. Directional shearing

2.1. Lagrange’s division

Let us consider two vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ such that $\|\mathbf{b}_i\| \leq \|\mathbf{b}_j\|$. We introduce the rational number $q = (\mathbf{b}_i^{\rm t}\mathbf{b}_j)/(\mathbf{b}_i^{\rm t}\mathbf{b}_i)$ from the orthogonal projection of $\mathbf{b}_j$ on $\mathbf{b}_i$ (Fig. 2). Practically, as in LLL, q is encoded by a floating-point number. The vector $q\mathbf{b}_i$ is rational and can be approximated by the integer vector $\lfloor q \rceil \mathbf{b}_i$, where $\lfloor q \rceil$ is the integer closest to q computed by $\lfloor q \rceil = {\rm int}({\rm round}(q))$. The reduced vector $\mathbf{r} = \mathbf{b}_j - \lfloor q \rceil \mathbf{b}_i$ belongs to the lattice spanned by $\mathbf{b}_i$ and $\mathbf{b}_j$, and its norm is such that $\|\mathbf{r}\| \leq \|\mathbf{b}_j\|$ if the coordinates of $\mathbf{b}_i$ and $\mathbf{b}_j$ are such that $|q| \geq \frac{1}{2}$, *i.e.* $\lfloor q \rceil \neq 0$. In the limit case $|q| = \frac{1}{2}$, the triangle made by $(\mathbf{b}_i, \mathbf{b}_j, \mathbf{r})$ is isosceles, *i.e.* $\|\mathbf{r}\| = \|\mathbf{b}_j\|$. Note that, in some cases, the norm of $\mathbf{r}$ that is lower than that of $\mathbf{b}_j$ may even be lower than that of $\mathbf{b}_i$. The vector $\mathbf{r}$ can be considered as the remainder of the vector division of $\mathbf{b}_j$ by $\mathbf{b}_i$.

Now, we consider a basis in N dimensions made of N integer vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ initially sorted by norms, from the lowest to the highest norms, *i.e.* such that $\|\mathbf{b}_1\| \leq \ldots \leq \|\mathbf{b}_i\| \leq \|\mathbf{b}_{i+1}\| \ldots \leq \|\mathbf{b}_N\|$. The function ‘Lagrange’s division’ consists of applying vector divisions to the pairs of vectors $(\mathbf{b}_i, \mathbf{b}_j)$ of the list. It starts with the vectors $\mathbf{b}_i = \mathbf{b}_1$ and $\mathbf{b}_j = \mathbf{b}_2$. Two cases should be distinguished in the algorithm: if $\lfloor q \rceil = 0$, nothing changes in the list and the next pair of vectors $(\mathbf{b}_i, \mathbf{b}_j)$ is considered by iteration with a loop with i containing a loop with j; and if $\lfloor q \rceil \neq 0$, the list is modified, and two algorithm variants are proposed:

Variant *Append*: the vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ are deleted from the list, and the vectors $\mathbf{r}$ and $\mathbf{b}_i$ are appended at the end of the list.

Variant *Insert*: if $\|\mathbf{r}\| \leq \|\mathbf{b}_i\|$, $\mathbf{r}$ replaces $\mathbf{b}_i$, and $\mathbf{b}_i$ replaces $\mathbf{b}_j$ in the list; else, $\mathbf{r}$ replaces $\mathbf{b}_j$ in the list.

The process is repeated recursively; the input for the function ‘Lagrange’s division’ is the new list of vectors

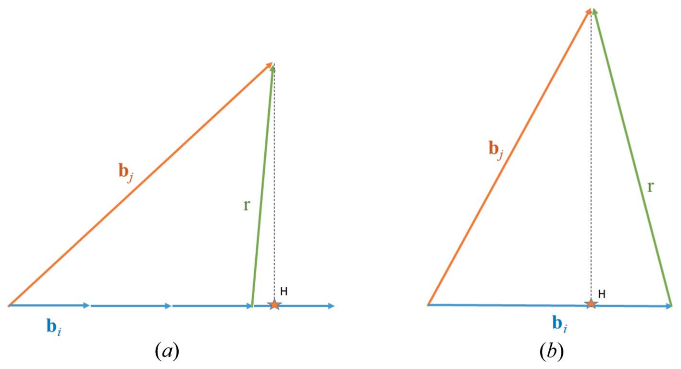


Figure 2
Directional shear of $\mathbf{b}_j$ along the direction $\mathbf{b}_i$. Case where (*a*) $\lfloor q \rceil = \lfloor (\mathbf{b}_i^t \mathbf{b}_j)/(\mathbf{b}_i^t \mathbf{b}_i) \rceil = 3$, and (*b*) $\lfloor q \rceil = 1$. The orthogonal projection point is noted H and marked by a little orange star.

(without sorting them). The recursion stops when all the values $\lfloor q \rceil$ become null for all the pairs of vectors in the basis. The method is quite similar to Lagrange's division described by Nguyen & Vallée (2010).

The variant *Insert* gives good results in a short time. The rhombicity and the sum of the squares of the norms of the list in Fig. 1(*a*) that were initially $R = 453988268$, $S = 61580172$ are reduced to $R = 540$, $S = 134$. These values are not far from those obtained with the LLL algorithm ($R = 531$, $S = 99$). With *Append*, the list of Fig. 1(*a*) is reduced 'only' to $R = 1199$, $S = 337$, but, as will be shown in the next sections, this will leave more action for the hyperplanar shearing, and better final reduction will be obtained at the end of the process for dimensions approximately $N \geq 15$.

2.2. Simplification

Lagrange's division reduces the vectors by pairs without considering the basis as a whole. Now, if one accepts to slightly but only temporarily degrade the value of S of the basis, the rhombicity R can be further improved as follows. Let us consider again a list of integer vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_i, \ldots, \mathbf{b}_N\}$ sorted by norms from the lowest to the highest norms. For a pair of vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ in the list such that $\|\mathbf{b}_i\| \leq \|\mathbf{b}_j\|$, we calculate the vector $\mathbf{r} = \mathbf{b}_j - \operatorname{sign}(\mathbf{b}_i^t \mathbf{b}_j)\mathbf{b}_i$, where $\operatorname{sign}(\mathbf{b}_i^t \mathbf{b}_j) = 1$ if $\mathbf{b}_i^t \mathbf{b}_j > 0$, -1 if $\mathbf{b}_i^t \mathbf{b}_j < 0$ and 0 if $\mathbf{b}_i^t \mathbf{b}_j = 0$. Then, we calculate whether or not replacing $\mathbf{b}_i$ or $\mathbf{b}_j$ by $\mathbf{r}$ allows the value of the rhombicity R to be decreased. If the answer is positive, the change is made. Here again, two algorithm variants are proposed

Variant '*Append*': if replacing $\mathbf{b}_i$ by $\mathbf{r}$ allows the value of R to be decreased, the vector $\mathbf{b}_i$ is deleted and the vector $\mathbf{r}$ is appended at the end of the list. If not, the vector $\mathbf{b}_j$ is deleted and the vector $\mathbf{r}$ is appended at the end of the list.

Variant '*Insert*': if replacing $\mathbf{b}_i$ by $\mathbf{r}$ allows the value of R to be decreased, the vector $\mathbf{b}_i$ is replaced by $\mathbf{r}$ at its position i; else, $\mathbf{b}_j$ is replaced by $\mathbf{r}$ at its position j. The new list of vectors is then sorted again following the increasing norms.

The variant '*Insert*' is chosen by default, except for random matrices for which the variant '*Append*' should be preferred, as will be discussed in Section 4. The process of simplification is repeated recursively until R cannot be reduced anymore. Simplification permits the values obtained in Section 2.1 to be decreased a little more. For the list of Fig. 1(*a*), from the lattice reduced by Lagrange's division with $R = 1199$, $S = 337$, the lattice is further reduced to $R = 1084$, $S = 330$ by simplification with the variant *Insert*. At this step, the rhombicity cannot be further reduced, even by combining Lagrange's division and simplification. In the rest of the paper, the process described in Section 2 will be called 'directional shearing'.

3. Hyperplanar shearing

3.1. The hyperplane normal

Let us consider again a list of integer vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ initially sorted by norms, *i.e.* such that $\|\mathbf{b}_1\| \leq \ldots \leq \|\mathbf{b}_j\| \leq \|\mathbf{b}_{j+1}\| \ldots \leq \|\mathbf{b}_N\|$. We isolate the first vector $\mathbf{b}_1$ and the subspace of dimension $N-1$ (hyperplane) constituted by the vectors $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$. The coordinates of the integer vector $\mathbf{p}_1$ that is normal to this hyperplane can be calculated as follows. We write the coordinates of vectors $\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N$ in columns to form the $N \times (N-1)$ matrix

$$\mathbf{S}_1 = \begin{bmatrix} b_{1,2} & \ldots & b_{1,j} & \ldots & b_{1,N} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ b_{i,2} & \ldots & b_{i,j} & \ldots & b_{i,N} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ b_{N,2} & \ldots & b_{N,j} & \ldots & b_{N,N} \end{bmatrix}$$

where $b_{i,j}$ means the ith coordinate of the vector $\mathbf{b}_j$.

If we insert in the matrix a first column made of any vector of the set $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$, let us say the vector $\mathbf{b}_j$, then the new set of vectors becomes linearly dependent and the determinant of the $N \times N$ matrix is null:

$$\det \begin{bmatrix} b_{1,j} & b_{1,2} & \ldots & b_{1,j} & \ldots & b_{1,N} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ b_{i,j} & b_{i,2} & \ldots & b_{i,j} & \ldots & b_{i,N} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ b_{N,j} & b_{N,2} & \ldots & b_{N,j} & \ldots & b_{N,N} \end{bmatrix} = 0.$$

Let us write this determinant by its cofactor expansion along the first column. The minors, *i.e.* the determinants of $\mathbf{M}_{1,k}$, the $(N-1) \times (N-1)$ submatrices of $\mathbf{S}_1$ obtained by deleting the kth row, form a vector

$$\mathbf{p}_1 = \begin{bmatrix} +\det(\mathbf{M}_{1,1}) \\ -\det(\mathbf{M}_{1,2}) \\ \vdots \\ (-1)^{k+1}\det(\mathbf{M}_{1,k}) \\ \vdots \\ (-1)^{N+1}\det(\mathbf{M}_{1,N}) \end{bmatrix}$$

that fulfils the property $\mathbf{p}_1^t \mathbf{b}_j = 0, \forall j \in [2, \ldots, N]$. In other words, $\mathbf{p}_1$ is the normal to the hyperplane $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$

that we were looking for. Its norm equals the area of the hypersurface formed by the vectors $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$. The reader can check that in three dimensions $\mathbf{p}_1 = \mathbf{b}_2 \wedge \mathbf{b}_3$. The coordinates of $\mathbf{p}_1$ are the Miller indices.

Note 1. The calculation of the coordinates of $\mathbf{p}_1$ from the determinants of the square matrices $\mathbf{M}_{1,k}$ may appear complicated and computationally expensive, and one may think about other methods. It can be noticed that the coordinates of $\mathbf{p}_1$ are the solution of $\mathbf{p}_1^{\rm t}\mathbf{S}_1 = {\rm NullRow}(N-1)$, the null row vector, or equivalently $\mathbf{S}_1^{\rm t}\mathbf{p}_1 = {\rm NullColumn}(N-1)$, the null column vector, both of dimension $N-1$. This system of equations is underdetermined since it is constituted of $N-1$ equations with N unknown. It can be solved by matrix inversion by imposing a specific value 0 or 1 to one of the coordinates of $\mathbf{p}_1$, but such an approach becomes numerically unstable and leads to incorrect solutions in high dimensions $N \geq 20$. A more classical way would be to compute Gaussian elimination taking care with the choices of the pivot positions to avoid instabilities, but the complexity is $O(N^3)$, which is comparable with that required to calculate N determinants of square matrices of dimension $N-1$.

3.2. Hyperplanar shear

Let us consider a cell of the lattice $\mathcal{L}$ attached to the hyperplane $\mathbf{p}_1$ generated by the vectors $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$, *i.e.* $\mathbf{p}_1^{\rm t}\mathbf{b}_j = 0, \forall j \in [2, \ldots, N]$. There are many equivalent cells, but we are looking for a quasi-reduced one. First, we replace the sublattice $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ by its reduced form $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$ obtained by directional shearing, as described in Section 2. If this reduction in dimension $N-1$ is not possible, the sublattice $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ is not changed, *i.e.* $\mathbf{b}'_j = \mathbf{b}_j$. All the vectors $\mathbf{b}'_j$ belong to the hyperplane $\mathbf{p}_1$; we say that they are in the layer $q = 0$ of the plane $\mathbf{p}_1$. Only the vector $\mathbf{b}_1$ points to a node of the layer q of the hyperplane $\mathbf{p}_1$ with $q \in \mathbb{Z}$ and $q > 0$. Note that $q = 1$ for a unit cell. The set $\{\mathbf{b}_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$ is a cell attached to the hyperplane (Cayron, 2021). Another vector of the lattice $\mathcal{L}$ pointing to the layer q such as $\mathbf{b}_1$ but shorter than $\mathbf{b}_1$ can be determined as follows. We note O the origin of the lattice, and Z the point such that $\mathbf{OZ} = \mathbf{b}_1$, as illustrated in Fig. 3. We call H the orthogonal projection of O on the layer q of the hyperplane $\mathbf{p}_1$. It is such that $\mathbf{OH} \parallel \mathbf{p}_1$ and $q = \mathbf{p}_1^{\rm t}\mathbf{OH} = \mathbf{p}_1^{\rm t}\mathbf{b}_1$. Thus, $\mathbf{OH} = [(\mathbf{p}_1^{\rm t}\mathbf{b}_1)/(\mathbf{p}_1^{\rm t}\mathbf{p}_1)]\mathbf{p}_1$. Its coordinates are not integer but remain rational.

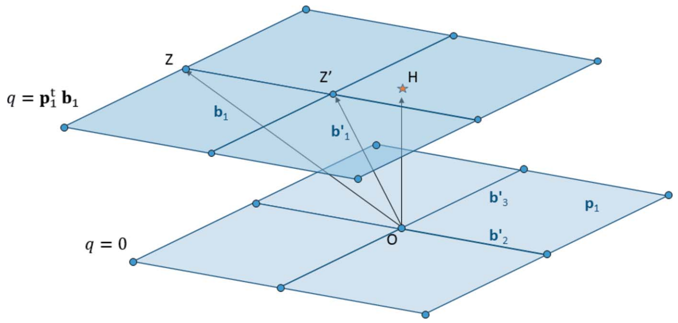


Figure 3
Hyperplanar shear parallel to $\mathbf{p}_1$. The lattice is 'stratified' into different layers parallel to $\mathbf{p}_1$. The layer to which the vector $\mathbf{b}_1$ points is given by the integer $q = \mathbf{p}_1^{\rm t}\mathbf{b}_1$. The hyperplanar shear is made by calculating the point H (marked by a little orange star) which is the orthogonal projection of the origin O onto the layer q. The node Z such that $\mathbf{b}_1 = \mathbf{OZ}$ can be translated towards another node Z' closer to H (see text). The vector $\mathbf{b}'_1 = \mathbf{OZ}'$ has a lower norm and a better 'orthogonality' with the hyperplane $\mathbf{p}_1$.

The vector $\mathbf{ZH} = -\mathbf{OZ} + \mathbf{OH}$ is a vector of the hyperplane $\mathbf{p}_1$, which means that it can be written as a linear combination of the vectors $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$. In order to get its coordinates, we use again the $N \times (N-1)$ matrix formed by writing the reduced vectors in columns, *i.e.*

$$\mathbf{S}'_1 = \begin{bmatrix} b'_{1,2} & \ldots & b'_{1,j} & \ldots & b'_{1,N} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ b'_{i,2} & \ldots & b'_{i,j} & \ldots & b'_{i,N} \\ \vdots & \ldots & \vdots & \ldots & \vdots \\ b'_{N,2} & \ldots & b'_{N,j} & \ldots & b'_{N,N} \end{bmatrix}.$$

The $N-1$ local coordinates of $\mathbf{ZH}$ in the basis $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$ are given by $\mathbf{ZH}_{\rm loc} = (\mathbf{S}'_1)^{-1}_{\rm Left}\mathbf{ZH}$ where $(\mathbf{S}'_1)^{-1}_{\rm Left}$ is the left inverse of the matrix $\mathbf{S}'_1$. We recall that a left inverse of a non-square matrix $\mathbf{M}$ is $\mathbf{M}^{-1}_{\rm Left} = (\mathbf{M}^{\rm t}\mathbf{M})^{-1}\mathbf{M}^{\rm t}$. The vector $\mathbf{ZH}_{\rm loc} = \{z_2, z_3, \ldots, z_N\}$ is an $N-1$-dimensional rational vector in the $N-1$ subspace. A lattice point Z' close to H that belongs to the same layer is given by $\mathbf{Z'H}_{\rm loc} = \{\lfloor z_2 \rceil, \lfloor z_3 \rceil, \ldots, \lfloor z_N \rceil\}$. The vector $\mathbf{ZZ}'_{\rm loc} = \mathbf{ZH}_{\rm loc} - \mathbf{Z'H}_{\rm loc}$ is calculated and re-expressed in the N-dimensional space by $\mathbf{ZZ}' = \mathbf{S}'_1 \cdot \mathbf{ZZ}'_{\rm loc}$. The vector $\mathbf{b}'_1 = \mathbf{OZ}' = \mathbf{OZ} + \mathbf{ZZ}'$ is a reduced form of the vector $\mathbf{b}_1$. At this step the cell $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ attached to the hyperplane $\mathbf{p}_1$ has been reduced; the new vectors defining this cell are $\{\mathbf{b}'_1, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$. This is the method used by Cayron (2021).

Note 2. The calculation of the $N-1$ local coordinates of $\mathbf{ZH}$ in the basis $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$ from the $N \times (N-1)$ matrix $\mathbf{S}'_1$ may appear complicated and computationally expensive, and one may think about other methods. One may notice that the coordinates of $\mathbf{ZH}$ form an $N-1$ vector $\mathbf{X}$ that is the solution of $\mathbf{S}'_1\mathbf{X} = \mathbf{ZH}$. The system of equations is overdetermined since it is constituted of N equations with $N-1$ unknown (the coordinates of $\mathbf{X}$). One could ignore one of the equations (*i.e.* remove one of the rows of $\mathbf{S}'_1$) to solve the system by matrix inversion, but such an approach becomes numerically unstable and leads to incorrect solutions for high dimension $N \geq 20$. This problem is induced by the projection. Let us explain it with an arbitrary example in three dimensions. We consider $\mathbf{b}'^{\,\rm t}_2 = [1211, 1423, 1]$ and $\mathbf{b}'^{\,\rm t}_3 = [-8921, 2389, 1]$, two vectors nearly perpendicular to the z axis, and the vector $\mathbf{ZH}$ that is in the plane $(\mathbf{b}'_2, \mathbf{b}'_3)$. If we work with the coordinates (x, y) of $\mathbf{ZH}$ to write it as a linear combination of $\mathbf{b}'_2$ and $\mathbf{b}'_3$, a solution is found without any problem. However, if the coordinates (x, z) or (y, z) of $\mathbf{ZH}$ are used, then the system becomes 'unbalanced', and it would become completely unsolvable if 0 were chosen in place of 1 for the z coordinates of the vectors $\mathbf{b}'_2$ and $\mathbf{b}'_3$. Geometrically, the instability

Table 1
Two cubification methods – the values of the options are given in Table 2.

Method 1	Method 2
Cubification (list, *opt.*):	Cubification (list, *opt.*):
newlist = Sort_by_norm (list)	newlist = Sort_by_norm (list)
newlist = Directional shearing (newlist, *opt.*)	newlist = Hyperplanar shearing (newlist)
newlist = Sort_by_norm (list)	newlist = Directional shearing (newlist, *opt.*)
newlist = Hyperplanar shearing (newlist)	newlist = Hyperplanar shearing (newlist)
If R (newlist) < R (list):	**If** R (newlist) < R (list):
Return Cubification (newlist, *opt.*)	**Return** Cubification (newlist, *opt.*)
Else Return list	**Else Return** list

Table 2
Method and option to be used depending on the type of square matrix.

We consider 'large' a matrix of dimension $N \geq 15$. For some large heterogeneous matrices a first step with hyperplanar shearing may be required before starting method 1, as indicated in parentheses.

Type of list of vectors	Cubification method	Variant for the directional reduction	
		Lagrange's division	Simplification
Small columnar matrix	Method 1	*Insert*	*Insert*
Large columnar matrix		*Append*	*Insert*
Large heterogeneous matrix	(Hyperplanar shearing +) method 1	*Insert*	*Insert*
Random matrix	Method 2	*Append*	*Append*

comes from the projection along a direction that makes the rhombus ($\mathbf{b}'_2$, $\mathbf{b}'_3$) appear nearly on its edge, as a segment. To avoid this problem, one could solve the overdetermined system by Gaussian elimination, taking care with the choices of the pivot positions to avoid instabilities, but the complexity would be comparable with that required to calculate the left inverses of matrices.

The function 'hyperplanar shear' works as follows. It starts with the list $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ and it tries to reduce $\mathbf{b}_1$ by a shear on the hyperplane $\{\mathbf{b}_2, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$, as described previously. If the basis rhombicity is reduced when $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ is substituted by $\{\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N\}$, the vector $\mathbf{b}'_1$ is moved to the end of the list, and the function is called again with $\{\mathbf{b}'_2, \ldots, \mathbf{b}'_j, \ldots, \mathbf{b}'_N, \mathbf{b}'_1\}$ as input. If the rhombicity is not reduced, the function keeps the initial list $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ and tries to reduce the vector $\mathbf{b}_2$ by a shear on the hyperplane $\{\mathbf{b}_1, \mathbf{b}_3, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ *etc.* The process stops when all the vectors $\mathbf{b}_i$ of the list $\{\mathbf{b}_1, \ldots, \mathbf{b}_j, \ldots, \mathbf{b}_N\}$ are screened but none of the vectors $\mathbf{b}'_i$ permits the basis rhombicity to be reduced any further. This series of hyperplanar shears will be called 'hyperplanar shearing'.

Both directional and hyperplanar shearing imply orthogonal projections followed by numerical rounding in which rational numbers are replaced by their closest integers, which is actually very similar to the operations required in the Gram–Schmidt procedure. The lattice of Fig. 1(*a*) that was previously reduced by directional shearing becomes even more reduced by hyperplanar shearing: the rhombicity and sum of the squares of the norms decreased to R = 451 and S = 113. These values are closer to those obtained by LLL, and they will be improved even more by alternating directional and hyperplanar shearing, as detailed in the next section.

4. Cycling directional and hyperplanar shearing

4.1. Methods and options

The directional and hyperplanar shearing steps can now be repeated in cycles until the rhombicity cannot be decreased anymore. This method is called 'cubification'. There is not a unique way to perform a cubification as it can be started by the directional shearing or by the hyperplanar shearing. It also depends on the variant of the algorithms chosen for Lagrange's division (Section 2.1) and for the simplification (Section 2.2). By trial and error, we could identify two cubification methods (Table 1).

The chosen algorithm variant depends on the type of matrix that is to be reduced (Table 2). We refer to 'columnar matrix' as a list of vectors whose matrix (the vectors are written in rows) contains many zeros, and at least one column contains many non-null and generally moderate integer values (here 3 or 4 digits). A typical example is the matrix given in Fig. 1(*a*). We noticed that for matrices of dimensions approximately $N \geq 15$, Lagrange's division in its *Append* variant gives better results than with '*Insert*'. A 'heterogeneous matrix' is a matrix that contains many zeros, and at least one row and one column with many non-null and moderate integer values. We noticed that for some cases of large heterogeneous matrices, with approximately $N \geq 15$, the first directional reduction may go beyond the recursion limit of our computer; when this happens, applying first a hyperplanar shearing solves the problem. A 'random matrix' is a matrix whose values are randomly computed with integers between 0 and 100. Limits larger than 100, for example 1000, in large random matrices $N \geq 15$ lead to too high integer values in intermediate calculations and error messages. A 'columnar random matrix' is here an identity matrix in which the last column is replaced by random integers in the range 0–100. Columnar random matrices are classified as random matrices and are treated with method 2.

4.2. Computer program and comparisons

We wrote a computer program called *Cubification* in Python 3.8 using the Numpy library to perform the matrix calculations (scalar products, matrix products, inverses *etc.*), generate the random numbers, vectors and matrices, and calculate the reduced lattices. All the results presented in the paper were obtained with a laptop computer equipped with an Intel Core i7-4600 CPU 2.1 GHz, 64-bit Windows system, with a RAM of 8 GB. The recursion limit in our Python program has been fixed to 10 000. We compared the results obtained with our program with those obtained by the LLL method computed in

Python 3 by Yonashiro (2020) in a program called *OLLL*. All the *OLLL* calculations were made with α = 3/4. For specific matrices, such as that of Fig. 1, we also used the function *ReduceLattice* of *Mathematica*. On this example we checked that *OLLL* and *Mathematica* give the same result; the only difference is that the calculations are nearly instantaneous with *Mathematica*, whereas they are longer (a few seconds) with Python language (*OLLL* and *Cubification*). This shows that it is difficult to compare the time efficiency of lattice reduction algorithms with computer programs written by different people in different languages. Thus, the execution times will just be given for indication.

4.3. Results on non-random matrices

The cubification algorithm gives results quite similar to those of LLL. For example, the lattice of Fig. 1(*a*) could be reduced in three cycles (in 3.0 s); the output list of vectors is given in Fig. 4. The final basis is characterized by R = 285, S = 87; these values are lower than those obtained by LLL (R = 531, S = 99). Souvignier (2021) showed that with the Schnorr–Euchner variant of LLL it is possible to get a reduced basis with R = 335, S = 83, and then, by computing the vectors of norm 4, selecting 18 of them and associating them with two vectors of norms 3, he could obtain a reduced basis with R = 294, S = 78. These solutions are significantly better than those obtained by *Mathematica.* Compared with the result obtained by cubification, they have a lower norm S (also a lower norm P), but a larger rhombicity R. This example shows that improving only the norms of the vectors does not always permit a better orthogonality (and vice versa) to be obtained, as also shown in Appendix *B*.

For heterogeneous matrices, we have tested only five 20 × 20 matrices, and all of them show that LLL and cubification give similar results (not shown here).

4.4. Results on random matrices

We have tested the performances of *Cubification* (method 2) and *OLLL* programs on columnar random matrices and full random matrices. We used matrices of dimensions 10 × 10, 12 × 12 and 14 × 14. Fifty matrices have been generated for each type. The performances on the norms and orthogonalities were measured by the reduction factors R(input)/R(output) and S(input)/S(output). The higher the reduction factors, the better the algorithm. The results are given in Table 3.

$$\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
-1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Figure 4
Cubification by method 1 of the lattice of Fig. 1(*a*). The vectors are written in rows, as in Fig. 1. The reduced basis has values R = 285, S = 87.

Table 3
Reduction factors obtained on columnar and full random matrices of dimensions 10 × 10, 12 × 12 and 14 × 14 by testing 50 matrices.

The mean deviation estimated by various tests is for *OLLL* around ± 20% for a 10 × 10 matrix and it decreases down to ±5% for a 14 × 14 matrix. It seems to be larger for *Cubification* (±25% and ±10%, respectively).

Reduction factor	R(input)/R(output)	S(input)/S(output)
Columnar random matrices 10 × 10		
OLLL	2780	1000
Cubification	3600	1060
Columnar random matrices 12 × 12		
OLLL	3120	1060
Cubification	4100	1090
Columnar random matrices 14 × 14		
OLLL	3630	1160
Cubification	4370	1070
Full random matrices 10 × 10		
OLLL	14.3	5.2
Cubification	16.9	5.4
Full random matrices 12 × 12		
OLLL	14.1	5.0
Cubification	15.2	4.6
Full random matrices 14 × 14		
OLLL	13.6	4.7
Cubification	14.3	4.1

For these moderate dimensions, the reduction of the rhombicity is systematically better with *Cubification* than with the *OLLL* algorithm. The norms seem however less reduced by *Cubification* for large full random matrices. The execution times of *Cubification* are 0.1, 0.3 and 0.5 s for 10 × 10, 12 × 12 and 14 × 14 columnar random matrices, respectively, and 0.2, 0.7 and 1.3 s for 10 × 10, 12 × 12 and 14 × 14 full random matrices, respectively. They are slightly shorter than with *OLLL*. We also performed some experiments in higher dimensions. The mean execution times are 14 and 30 s for 30 × 30 columnar and full random matrices, respectively. They are shorter than with *OLLL*, but ten times longer than those reported with other optimized Python programs (Papachristoudis *et al.*, 2015). The way the algorithm is implemented, the choice of types of variables, the use of different libraries, the memory management, all play a crucial role in the execution times. In this paper, the code was not optimized to reach the best performances in execution times; its aim was only to show that simple shears along directions and hyperplanes may be interesting tools for lattice reduction.

5. Conclusion and perspectives

A method of lattice reduction called 'cubification' is proposed. It is geometrically simple; it is based on the complementary actions of directional shearing and hyperplanar shearing. These two kinds of shears were initially introduced to reduce the unit cells attached to given hyperplanes (Cayron, 2021). In contrast to LLL, the cubification algorithm does not require the calculations of Gram–Schmidt bases. The 'driving force' of the reduction is the 'basis rhombicity', a parameter that encompasses the information on the norms and angles of the basis vectors. A computer program called *Cubification* was written in Python 3.8. The results are comparable with those of LLL, at least up to moderate dimensions ($N \le 20$). The Python program *Cubification* is freely available from the author on request.

We foresee margins of progression for the algorithm of cubification. The two methods described in Section 4.1 were determined by trial and error; better strategies to alternate the directional and hyperplanar shears seem possible, for example by cross-calling the two processes without necessarily screening all the vectors in the basis. We could also try to generalize the $N \to N-1$ decrease of dimensions already used in the hyperplanar shearing step with the help of the left inverse matrices to work in spaces of dimensions $N-1$, $N-2$ *etc*.

APPENDIX *A*
Brief overview of the LLL algorithm

The most popular algorithm to tackle the lattice reduction problem was proposed nearly 40 years ago by Lenstra–Lenstra–Lovász (Lenstra *et al.*, 1982), and it is still considered as the main reference in the domain. It should be noted that the LLL algorithm does not give in general a Hermite–Minkowski reduced basis for which the vectors have minimal lengths (Ryshkov, 1976), but 'only' a basis made of short and nearly orthogonal vectors that constitutes a good, approximate solution that is very useful for many applications. It was initially designed to give in polynomial-time a good solution for factorizing polynomials with rational coefficients, and it is also nowadays applied for finding rational approximations to real numbers, and for solving the integer linear programming problems in fixed dimensions; it is applied in global positioning systems (GPS), data detection and communication systems. It is so important that a complete book has been devoted to it (Nguyen & Vallée, 2010). The reader can also consult Wübben *et al.* (2011). We just give here some of its key points. At the core of LLL is the Gram–Schmidt orthogonalization routine in which one attaches to any basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_k, \ldots, \mathbf{b}_{m \le N}\}$ an orthogonal basis $\{\mathbf{b}_1^*, \ldots, \mathbf{b}_k^*, \ldots, \mathbf{b}_m^*\}$ by a series of projections $\mathbf{b}_k^* = \mathbf{b}_k - \sum_{i<k} u_{i,k}\mathbf{b}_i$ with $u_{i,k} = (\mathbf{b}_k \cdot \mathbf{b}_i^*)/(\mathbf{b}_i^* \cdot \mathbf{b}_i^*)$. The vectors $\mathbf{b}_k^*$ are not integer anymore (*i.e.* 'reticular' in crystallographic language); they remain however rational. Practically, as the numerators and denominators may become huge numbers, floating-point numbers are used for $u_{i,k}$. The LLL algorithm works in two steps repeated iteratively. The first step is the quasi-orthogonalization. The vectors $\mathbf{b}_i$ are replaced by $\mathbf{b}_i - \lfloor u_{i,k} \rceil \mathbf{b}_k$, for k between 1 and $i-1$, where $\lfloor u_{i,k} \rceil$ means the nearest integer of $u_{i,k}$. The Gram–Schmidt basis should be actualized during the process. The second step relies on a criterion to determine whether or not the vectors $\mathbf{b}_i$ and $\mathbf{b}_i$ should be swapped: the swap is made when $\|\mathbf{b}_{i+1}^* + u_{i,i+1}\mathbf{b}_i^*\|^2 < \alpha\|\mathbf{b}_i^*\|^2$, where α is a constant arbitrarily chosen between $\frac{1}{4}$ and 1 (and fixed once for all). Often, the value $\alpha = \frac{3}{4}$ is chosen. The constant α influences the strength of reduction in the algorithm and by that also the number of required iterations; greater values lead to stronger reductions; it has an effect on the final norms of the reduced vectors, and more precisely it permits the product of the squared norms $\prod_{i=1}^{N} \|\mathbf{b}_i\|^2$ to be bound.

APPENDIX *B*
Example of a lattice with two reduced bases of the same norms but different orthogonalities

This appendix provides an example showing that minimizing the norms of the vectors of a lattice does not necessarily permit their orthogonality to be improved. Let us consider the lattice spanned by the four vectors $\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4\}$ whose coordinates are written in rows by

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

The squares of the norms of the four vectors are 2, 2, 2, 3. The parameters that can be used to evaluate the 'norm' of the basis $\mathbf{B}$ are the sum of the squares of norms $S(\mathbf{B}) = 9$ and the products of the squares of norms $P^2(\mathbf{B}) = 24$. This basis is already reduced if one considers only the norm of $\mathbf{B}$. The output of the LLL algorithm is thus the same basis. However, the same lattice may also be given by the vectors $\mathbf{b}_1' = \mathbf{b}_1$, $\mathbf{b}_2' = \mathbf{b}_1 - \mathbf{b}_2$, $\mathbf{b}'_3 = \mathbf{b}_3$, $\mathbf{b}_4' = \mathbf{b}_3 - \mathbf{b}_4$, written in rows:

$$\mathbf{B}' = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & 1 & -1 & 0 \end{bmatrix}.$$

The squares of the norms of the four vectors are 2, 2, 2, 3, and the new basis $\mathbf{B}'$ is characterized by the parameters $S(\mathbf{B}') = 9$ and $P^2(\mathbf{B}') = 24$. The two bases $\mathbf{B}$ and $\mathbf{B}'$ generate the same lattice and have the same 'norm', but their orthogonalities are different. Their metric tensors are

$$\mathcal{M}(\mathbf{B}) = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 3 \end{bmatrix}$$

and

$$(\mathbf{B}') = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 3 \end{bmatrix},$$

respectively. Their rhombicities are $R(\mathbf{B}) = 21$ and $R(\mathbf{B}') = 15$, respectively. The basis $\mathbf{B}'$ is thus more 'orthogonal' than the basis $\mathbf{B}$. This example shows that the term 'orthogonality defect' usually attributed to the parameter P/V may not be very appropriate. Since the value $R - S$ gives the

Euclidean scalar products of the vectors with the other ones, the parameter $(R - S)/S$ seems better adapted to characterize the 'orthogonality defect'. The cubification method described in the paper aims at reducing both the norms and the orthogonalites of the vectors, which is why the rhombicity R was used as a driving force in the algorithm. The basis $\mathbf{B}'$ of the example was found by cubification.

Acknowledgements

Professor Roland Logé is warmly acknowledged for the freedom given to our research that sometimes goes beyond metallurgy. I would also like to thank the reviewers, and particularly Professor Souvignier, who showed me the efficiency of the Schnorr–Euchner method on the same examples with the same parameters R and S as those introduced in the paper. The paper was modified and improved thanks to his comments. Professor Palatinus is also thanked for his advice and for putting me in contact with Professor Souvignier.

Funding information

Open access funding provided by Ecole Polytechnique Federale de Lausanne.

References

Cayron, C. (2021). *Acta Cryst.* A**77**, 453–459.

Lenstra, A. K., Lenstra, H. W. Jr & Lovász, L. (1982). *Math. Ann.* **261**, 515–534.

Nguyen, P. Q. & Vallée, B. (2010). *The LLL Algorithm. Survey and Applications.* Berlin, Heidelberg: Springer-Verlag.

Papachristoudis, D. G., Halkidis, S. T. & Stephanides, G. (2015). *Int. J. Appl. Comput. Math.* **1**, 327–342.

Ryshkov, S. S. (1976). *J. Math. Sci.* **6**, 651–671.

Souvignier, B. (2021). Personal communication.

Wübben, D., Seethaler, D., Jaldén, J. & Matz, G. (2011). *IEEE Signal Process. Mag.* **28**, 70–91.

Yonashiro, N. (2020). *OLLL, a Python3 Implementation of LLL*, available at https://github.com/orisano/olll.