Complexity of an extended lattice reduction algorithm

Wilberd van der Kallen

December 1998

1 Summary

We consider the complexity of a Lenstra Lenstra Lovász lattice reduction algorithm ([LLL]) in which the vectors are allowed to be linearly dependent and in which one also asks for the matrix of the transformation from the given generators to the reduced basis. The main problem will be to show that the entries of the transformation matrix remain bounded through the algorithm, with a reasonable bound. Here the difficulty is of course that due to the dependence of the generators the transformation is not determined by the basis. To remedy this we work with two inner products and apply the LLL methods to both.

2 Description of `GramLatticeReduce`

Let $e_1,\ldots,e_n$ be the standard basis of $\mathbb{R}^n$. The input of `GramLatticeReduce` is the Gram matrix $gram = (\langle e_i, e_j\rangle)_{i,j=1}^n$ of a positive semidefinite inner product $\langle\ ,\ \rangle$ on $\mathbb{R}^n$. We assume *gram* has integer entries. We are concerned with the lattice $\mathbb{Z}^n$. The output of `GramLatticeReduce` is an integer *rank* and an integer matrix b of determinant one. To explain its properties we need some more notation. The ordinary inner product on $\mathbb{R}^n$ is denoted $(\ ,\ )$. Call v isotropic if $\langle v, v\rangle = 0$. Put $isodim = n - rank$ and let b_i^* denote the i-th Gram-Schmidt vector in the following sense. We have $b_i^* \in (b_i + \sum_{j=1}^{i-1} \mathbb{R} b_i)$ and if $1 \le j < i$, $j \le isodim$ then $(b_i^*, b_j) = 0$, but if $1 \le j < i$, $j > isodim$ then $\langle b_i^*, b_j\rangle = 0$. With those notations the output satisfies:

1. The first *isodim* rows b_i of b are isotropic.

2. With respect to $(\ ,\ )$ the first *isodim* rows of b form an LLL reduced basis of $\sum_{j=1}^{isodim} \mathbb{Z} b_i$.

3. With respect to $\langle\ ,\ \rangle$ the last *rank* rows of b form an LLL reduced basis of the lattice they span, and this lattice contains no nonzero isotropic vector.

4. For $1 \leq i \leq isodim$, $i < j \leq n$ we have $|(b_i^*, b_j)| \leq 1/2(b_i^*, b_i)$.

Remark 2.1 Variations are possible, depending on what one is really after. If one is only interested in the isotropic vectors, one may weaken condition 3 to

3' The last *rank* rows of b form a basis of the lattice they span, and this lattice contains no nonzero isotropic vector. Furthermore, $|\langle b_i^*, b_j \rangle| \leq 1/2\langle b_i^*, b_i \rangle$ for $n - rank + 1 \leq i \leq n$, $i < j \leq n$.

Similarly, one may wish to replaces condition 2 with

2' For $1 \leq i \leq isodim$, $i < j \leq isodim$ we have $|(b_i^*, b_j)| \leq 1/2(b_i^*, b_i)$.

These changes do not affect our analysis in any essential way. One just has to change the wording, not the formulas. And in the algorithm one has to leave out some swaps.

3 Description of `ExtendedLatticeReduce`

Given generators $b_1, \ldots, b_n$ of a sublattice of $\mathbb{Z}^m$, the algorithm `ExtendedLatticeReduce` basically just calls `GramLatticeReduce` with as input the Gram matrix $((b_i, b_j))_{i,j=1}^n$.

4 Sketch of the algorithm

We assume the reader is familiar with [LLL] and also with the implementation of the LLL algorithm in integer arithmetic, as described in [C].

Most of the time we are given

- An integer matrix b of determinant one,

- Integers k, $kmax$, $1 \leq k \leq kmax \leq n$,
- An integer $isodim \geq 0$, so that the first $isodim$ rows b_i of b span the isotropic subspace of $\sum_{j=1}^{kmax} \mathbb{R}b_j$.

(Initialize with $k = kmax = 1$ and $isodim = 0$.)

Let $\mathrm{pr}_{\mathrm{iso}}$ be the orthogonal projection according to $(\ ,\ )$ of $\mathbb{R}^n$ onto $\sum_{j=1}^{isodim} \mathbb{R}b_j$ and put

$$(v, w)_{\mathrm{mix}} = (\mathrm{pr}_{\mathrm{iso}} v, \mathrm{pr}_{\mathrm{iso}} w) + \langle v, w \rangle.$$

Let $\mu_{i,j}$ be defined for $i > j$ so that

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^*.$$

The first standard assumption is then that, with respect to $(\ ,\ )_{\mathrm{mix}}$, the first $k-1$ rows of b form an LLL reduced basis of $\sum_{j=1}^{k-1} \mathbb{Z}b_j$, except that one does *not* require

$$|b_i^* + \mu_{i,i-1} b_{i-1}^*|_{\mathrm{mix}}^2 \geq 3/4 |b_{i-1}^*|_{\mathrm{mix}}^2$$

when $i = isodim + 1$. And the second standard assumption is that, as in [C], the first $kmax$ rows of b form a basis of $\sum_{j=1}^{kmax} \mathbb{Z}e_i$.

We run the LLL algorithm with respect to $(\ ,\ )_{\mathrm{mix}}$, except that one never swaps b_{isodim} with $b_{isodim+1}$. This roughly amounts to running two LLL algorithms, one for $(\ ,\ )$ and one for $\langle\ ,\ \rangle$. That is how one implements it and that is how we would have told it if we had not needed $(\ ,\ )_{\mathrm{mix}}$ for the complexity analysis. One runs LLL until k tries to go to $kmax + 1$. If $kmax = n$ we are through. If $kmax < n$ and $\sum_{j=1}^{kmax+1} \mathbb{Z}b_j$ contains no more isotropic vectors than $\sum_{j=1}^{kmax} \mathbb{Z}b_j$, then we simply increase $kmax$ by one.

In the remaining case we have to work until we are back in the standard situation with $k = 2$ and with both $isodim$ and $kmax$ one bigger. This is what `trickledown` is for. We postpone its discussion. At the end of `trickledown` we also make that $|\mu_{i,j}| \leq 1/2$ for $j < i \leq kmax$. This may not be necessary (and indeed we did not do this in earlier versions) but it can do little harm and definitely simplifies the estimates below.

5 Estimates

We want to give estimates as in [LLL]. Thus let $B \geq 2$ so that the entries of *gram* are at most B. Our main result is that all through the algorithm all entries have bit length $\mathcal{O}(n \log(nB))$. We do not care about the constants in this estimate. We leave to the reader the easy task of estimating the number of operations in the algorithm in the manner of [LLL].

5.1 Determinants

Let $gram_{\mathrm{mix}}$ be the Gram matrix $((e_i, e_j)_{\mathrm{mix}})$ with respect to $e_1, \ldots, e_{kmax}$. Its entries are at most $B + 1$. With Hadamard this gives

$$|\det(gram_{\mathrm{mix}})| \leq (\sqrt{n}(B+1))^n$$

and the same estimate holds for its subdeterminants. We claim that the determinant of $gram_{\mathrm{mix}}$ is an integer, so that we also get this upper bound for the entries of $gram_{\mathrm{mix}}^{-1}$. To see the claim, consider as in [P] the inner product $(\ ,\ )_\epsilon$ given by $(v, w)_\epsilon = \epsilon(v, w) + \langle v, w \rangle$. Its Gram matrix has a determinant which is a polynomial $\det_\epsilon$ of ϵ with integer coefficients. One may also compute $\det_\epsilon$ with respect to a basis which is obtained from $e_1, \ldots, e_{kmax}$ through an orthogonal transformation matrix. By diagonalizing the Gram matrix of $\langle\ ,\ \rangle$ we see that $\det(gram_{\mathrm{mix}})$ is the coefficient of ϵ^{isodim} in $\det_\epsilon$. □

Lemma 5.2 *For $v \in \mathbb{R}^n$ one has*

$$(v, v)_{\mathrm{mix}} \leq n(B+1)(v, v)$$

and for $v \in \sum_{j=1}^{kmax} \mathbb{R} e_j$ one has

$$(v, v) \leq n(\sqrt{n}(B+1))^n (v, v)_{\mathrm{mix}}.$$

Proof

The supremum of $\{\, (v, v)_{\mathrm{mix}} \mid (v, v) = 1 \,\}$ is the largest eigenvalue of the gram matrix of $(\ ,\ )_{\mathrm{mix}}$ with respect to $e_1, \ldots, e_n$. The largest eigenvalue is no larger than the trace of this matrix. So it is at most $n(B+1)$. Similarly the largest eigenvalue of $gram_{\mathrm{mix}}^{-1}$ it is at most $n(\sqrt{n}(B+1))^n$. □

5.3 Vectors

Now put

$$diso_i = \prod_{j=1}^{i} (b_j^*, b_j^*)$$

for $i \leq isodim$ and

$$d_i = \prod_{j=1}^{i} \langle b_{j+isodim}^*, b_{j+isodim}^* \rangle$$

for $i \leq rank$. As far as d_i is concerned we may compute modulo isotropic vectors, or also with $(\ ,\ )_{\mathrm{mix}}$. Indeed

$$\langle b_{j+isodim}^*, b_{j+isodim}^* \rangle = (b_{j+isodim}^*, b_{j+isodim}^*)_{\mathrm{mix}}$$

for $1 \leq j \leq rank$. Both $diso_i$ and d_j are integers and they descend when applying LLL.

One may also compute $\det(gram_{\mathrm{mix}})$ with the b_i^* basis, as the transition matrix has determinant one. From that one sees that it is just $diso_{isodim} d_{rank}$. So we get $diso_{isodim} \leq (\sqrt{n}(B+1))^n$. In fact, for $i \leq isodim$ one has the same estimate

$$diso_i \leq (\sqrt{n}(B+1))^n$$

because i was equal to $isodim$ earlier in the algorithm and LLL only makes it go down. Similarly $d_{rank} \leq (\sqrt{n}(B+1))^n$, but actually we know from [LLL] that

$$d_i \leq B^i.$$

(This is not spoiled by `trickledown` which also makes d_i descend.)

Lemma 5.4 *Let $1 \leq i \leq kmax$. Then*

$$(\sqrt{n}(B+1))^{-n} \leq (b_i^*, b_i^*)_{\mathrm{mix}} \leq (\sqrt{n}(B+1))^n$$

and if $|\mu_{ij}| \leq 1/2$ for $1 \leq j < i$ then

$$(b_i, b_i)_{\mathrm{mix}} \leq n(\sqrt{n}(B+1))^n$$

□

Remark 5.5 These estimates are not as sharp as those in [LLL] but that can not be helped: It is no longer true that $(b_1^*, b_1^*) \leq B$. Consider for instance the quadratic form $\langle v, v \rangle = \sum_{i=1}^{n-1} (Nx_i - x_{i+1})^2$ for some large integer N. The shortest nonzero isotropic vector in $\mathbb{Z}^n$ is $(1, N, N^2, \ldots, N^n)$.

5.6 Preserved estimates

Lemma 5.7 *The following estimates hold between applications of* `trickledown`*.*

1. $diso_i \leq (\sqrt{n}(B+1))^n$ *for* $i \leq isodim$,
2. $d_i \leq B^i$ *for* $i \leq rank$,
3. $(b_i, b_i)_{\mathrm{mix}} \leq n(\sqrt{n}(B+1))^n$ *for* $i \neq k$,
4. $(b_k, b_k)_{\mathrm{mix}} \leq n^2 4^n (\sqrt{n}(B+1))^{3n}$,
5. $|\mu_{i,j}| \leq 1/2$ *for* $1 \leq j < i < k$,
6. $|\mu_{k,j}| \leq 2^{n-k}\sqrt{n}(\sqrt{n}(B+1))^n$ *for* $1 \leq j < k$,
7. $|\mu_{i,j}| \leq \sqrt{n}(\sqrt{n}(B+1))^n$ *for* $1 \leq j < i > k$.

Proof

That these are preserved under LLL follows as in [LLL], so one has to check that they hold right after `trickledown`. Given our earlier estimates this is straightforward. □

6 Description of `trickledown`

Before we can do estimates concerning `trickledown` we must describe it. One starts with having $k = kmax + 1 \leq n$. Consider the lattice generated by $b_1, \ldots, b_{kmax+1}$ where $b_{kmax+1} = e_{kmax+1}$. By assumption this lattice contains a nonzero vector v with $(v, v)_{\mathrm{mix}} = 0$. Modulo $\mathbb{R}v$ the vector b_k is linearly dependent on the b_i with $i < k$. Changing the basis of $\mathbb{Z}b_{k-1} + \mathbb{Z}b_k$ we can achieve that modulo $\mathbb{R}v$ the vector b_{k-1} is linearly dependent on the b_i with $i < k-1$. Then lower k by one and repeat until $k = isodim+1$, where *isodim* is the one from before the present `trickledown`. At that point b_k is itself isotropic and we increase *isodim* by one and pass to a new $(\ ,\ )_{\mathrm{mix}}$. After subtracting suitable multiples of b_j with $j < i$ from b_i for all i we arrive at the situation where $|\mu_{i,j}| \leq 1/2$ and we leave `trickledown` with $k = 2$ (or $k = \max(isodim, 2)$) and with *kmax* increased by one.

7 Estimates during `trickledown`

We look in more detail. Upon entering `trickledown` we freeze the old *isodim*, *kmax* and the b_i^*, even though the b_i will change. We also do not change $(\ ,\ )_{\mathrm{mix}}$. Let $\mu_{i,0}$ stand for (e_{kmax+1}, b_i) and let $\mu_{i,j}$ stand for $(b_j^*, b_i)_{\mathrm{mix}}/(b_j^*, b_j^*)_{\mathrm{mix}}$ if $j > 0$. Note that initially $|\mu_{i,j}| \leq 1$ for $i \leq kmax$, $0 \leq j \leq kmax$. We will estimate $|\mu_{i,j}|$ as k descends.

Say $k > isodim + 1$ and modulo $\mathbb{R}v$ the vector b_k is linearly dependent on the b_i with $i < k$. Let us compute with b_k, b_{k-1} modulo $V = \mathbb{R}v + \sum_{i=1}^{k-2} \mathbb{R}b_i$. We have $b_k \equiv \mu_{k,k-1} b_{k-1}^*$ and $b_{k-1} \equiv b_{k-1}^*$ modulo V. With the extended euclidean algorithm of [C] we find an integer matrix $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$ of determinant one so that $\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} 1 \\ \mu_{k,k-1} \end{pmatrix} = \begin{pmatrix} 0 \\ -1/r_k \end{pmatrix}$ where r_k is the index of $\mathbb{Z}$ in $\mathbb{Z} + \mathbb{Z}\mu_{k,k-1}$. More specifically, one has $\begin{pmatrix} \delta & -\beta \\ -\gamma & \alpha \end{pmatrix} \begin{pmatrix} 0 \\ -1/r_k \end{pmatrix} = \begin{pmatrix} 1 \\ \mu_{k,k-1} \end{pmatrix}$ so $\beta = r_k$ and $\alpha = -r_k \mu_{k,k-1}$. By [C] we have $|\gamma| \leq |\mu_{k,k-1} r_k|$ and $|\delta| \leq r_k$. (Here we assume for simplicity that $\mu_{k,k-1}$ is not zero.)

Now put $c_{k-1} = \alpha b_{k-1} + \beta b_k$ and $c_k = \gamma b_{k-1} + \delta b_k$. The algorithm will tell us to replace b_k with c_k and b_{k-1} with c_{k-1}. We want to estimate the resulting new $\mu_{i,j}$, which we call $\nu_{i,j}$. For i different from k, $k-1$ nothing changes. Further $|\nu_{k-1,j}| = |\alpha \mu_{k-1,j} + \beta \mu_{k,j}| \leq r_k |\mu_{k,k-1} \mu_{k-1,j}| + r_k |\mu_{k,j}|$ and $|\nu_{k,j}| = |\gamma \mu_{k-1,j} + \delta \mu_{k,j}| \leq r_k |\mu_{k,k-1} \mu_{k-1,j}| + r_k |\mu_{k,j}|$, which is the same bound.

Remark 7.1 During `trickledown` the d_i are repeatedly replaced by divisors. As we are recording the $\mu_{i,j}$ with $j > isodim$ as fractions with a denominator $d_{j-isodim}$, this means that one has to update the numerators too. By remembering the r_k one can postpone all this updating until the end of `trickledown`, processing the product of the corrections, rather than each correction separately.

Lemma 7.2 *As k descends we have*

1. $|\mu_{i,j}| \leq 1$ *for* $k > i > j \geq 0$,
2. $|\mu_{k,j}| \leq \sqrt{B}(\sqrt{n}(B+1))^{n/2} \prod_{i=k+1}^{kmax+1} (2r_i)$ *for* $k > j \geq 0$,
3. $|\mu_{i,j}| \leq 2^n n^{n/4} (B+1)^n$ *for* $k \leq i > j \geq 0$.

Proof

Initially we have $k = kmax + 1$ and we estimate $|\mu_{k,j}|^2 \leq B(b_j^*, b_j^*)_{\text{mix}}^{-1} \leq B(\sqrt{n}(B+1))^n$. Now assume the estimates are true for the present k. We get $|\nu_{k-1,j}| \leq r_k|\mu_{k,k-1}\mu_{k-1,j}| + r_k|\mu_{k,j}| \leq 2r_k \max_j |\mu_{k,j}|$ which takes care of $|\nu_{k-1,j}|$. As $\prod_{k=isodim+2}^{kmax+1} r_k^2$ is the ratio by which d_{rank} drops during `trickledown`, it is at most B^{rank}. So $|\nu_{k,j}| \leq \sqrt{B}(\sqrt{n}(B+1))^{n/2}2^n B^{rank/2}$. □

Remark 7.3 Experiments show it is wise to insert a reduce step to make that $|\mu_{k,j}| \leq 1/2$ for $j \neq 0$.

7.4 Increasing *isodim*

When k has reached $isodim + 1$ it is time to increase *isodim* by one and pass to a new $(\ ,\ )_{\text{mix}}$. But first use the estimates of the $\mu_{i,j}$ to estimate $(b_i, b_i)_{\text{mix}}$ and $(\mu_{i,0}e_{kmax+1}, \mu_{i,0}e_{kmax+1})_{\text{mix}}$, next $(b_i - \mu_{i,0}e_{kmax+1}, b_i - \mu_{i,0}e_{kmax+1})$ by means of Lemma 5.2, and finally (b_i, b_i).

Now change *isodim*, *kmax*, $(\ ,\ )_{\text{mix}}$. We have to compute the new $\mu_{j,isodim}$. They can be estimated, as we have an estimate for (b_j, b_j) and for $(b_{isodim}^*, b_{isodim}^*)_{\text{mix}}^{-1}$.

Finally we reduce to the case $|\mu_{i,j}| \leq 1/2$ for $i > j$. During this reduction the maximum of the $|\mu_{i,j}|$ gets at most 2^n as large by the argument in [LLL].

We have seen that all the integers that are encountered have bit length $\mathcal{O}(n \log(nB))$.

8 Implementation in Mathematica

```
(*:Name: NumberTheory‘LLLalgorithm‘ *)

(*:Summary:  The extended Lenstra Lenstra Lovasz algorithm finds     *)
(*           a lattice basis consisting of "short" vectors,         *)
(*           together with a transformation that relates the new    *)
(*           basis with the original set of generators. Instead of  *)
(*           giving a generating set directly, one may also give    *)
(*           its Gram matrix. When the generators are linearly      *)
(*           dependent, a reduced basis of the lattice of relations *)
(*           ("null space lattice") is also obtained.               *)
```

```
(*:Author:        Wilberd van der Kallen, 1998                          *)

(*:Mathematica Version: 3.0 *)
(*:Package Version: 1.111 *)

(*:Warnings:                                                            *)
(* For us a lattice in euclidean n-space need not be of full rank n. *)

(*:Sources:  For the original LLL paper, see                            *)
(* Mathematische Annalen 261, 515-534 (1982).                          *)
(* For the case of dependent vectors, see                              *)
(* M. Pohst, A modification of the LLL-algorithm,                      *)
(* Journal of Symbolic Computation 4 (1987), 123--128.                 *)
(* Much of the code below relies on:                                   *)
(* Henri Cohen, A course in computational Algebraic Number Theory,     *)
(* Graduate Texts in Mathematics 138, Springer 1993.                   *)

(* :Discussion:  This version is a rather minimal version with few     *)
(* options. It may serve to document the algorithm.                    *)
(*                                                                     *)
(* We add ExtendedLatticeReduce and GramLatticeReduce  to the          *)
(* builtin function LatticeReduce.                                     *)
(*                                                                     *)
(*           ExtendedLatticeReduce[matrix]                             *)
(*                                                                     *)
(* computes a reduced basis of two lattices. One is the row space      *)
(* lattice spanned by the rows of the matrix, the other is the null    *)
(* space lattice, consisting of vectors v that have integer entries    *)
(* and satisfy v.matrix=0. (Contrary to the function NullSpace, we     *)
(* are concerned with multiplication from the left here.)              *)

BeginPackage["LLLalgorithm`"]

ExtendedLatticeReduce::usage:=
"ExtendedLatticeReduce[{v1, v2, ...}] gives
{reduced basis,transformation}
where the transformation is an integral matrix T of determinant plus or
minus one such that the rows of T.{v1, v2, ...} form a reduced basis,
```

```
possibly preceded by zero vectors. These zero vectors occur if the
integral vectors vi are linearly dependent.  In that case the
corresponding top part of T is a reduced basis of the ‘null space
lattice’.";

GramLatticeReduce::usage:=
"GramLatticeReduce[{{vi.vj}}] gives {rank of lattice,transformation}
where the transformation is an integral matrix T of determinant plus or
minus one such that the rows of T.{v1, v2, ...} form a reduced basis,
possibly preceded by zero vectors. These zero vectors occur if the
vectors vi are linearly dependent.\n
The Gram matrix {{vi.vj}} should have rational entries.";

RatioLLLCondition::usage:="RatioLLLCondition is an option in
ExtendedLatticeReduce and in GramLatticeReduce.
It tells which ratio to use in the criterion for an LLL
reduced basis. The default is 3/4, but for some applications one should
take it closer to one.\n
RatioLLLCondition->{3/4,19/20,99/100} causes
three passes, with ratios 3/4, 19/20, 99/100 respectively."

(*:Examples:

In[1]:= <<LLLalgorithm.m

In[2]:= generatingset={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

In[3]:= {reducedbasis,transformation}=ExtendedLatticeReduce[generatingset]

Out[3]= {{{2, 1, 0}, {-1, 1, 3}},

>    {{1, -1, -1, 1}, {1, -2, 1, 0}, {-1, -1, 1, 0}, {2, 1, -1, 0}}}

In[4]:= transformation.generatingset

Out[4]= {{0, 0, 0}, {0, 0, 0}, {2, 1, 0}, {-1, 1, 3}}

In[5]:= Take[%,-Length[reducedbasis]]
```

```
Out[5]= {{2, 1, 0}, {-1, 1, 3}}

In[6]:= %==reducedbasis

Out[6]= True

In[7]:= GramLatticeReduce[generatingset.Transpose[generatingset]]==
{Length[reducedbasis],transformation}

Out[7]= True

In[8]:= reducedbasisnullspace=Take[transformation,Length[transformation]-
         Length[reducedbasis]]

Out[8]= {{1, -1, -1, 1}, {1, -2, 1, 0}}

In[9]:= reducedbasisnullspace.generatingset

Out[9]= {{0, 0, 0}, {0, 0, 0}}

*)

Unprotect[ExtendedLatticeReduce,GramLatticeReduce];

Options[ExtendedLatticeReduce]={RatioLLLCondition->3/4};

Options[GramLatticeReduce]={RatioLLLCondition->3/4};

Begin["LLLalgorithm`Private`"]

(* The following will be local to this Private block:
answer           m
b                mat
basis            matextendedgcd
bbb              moreratios
coefficient      n
coefficientiso
```

```
cond              notLLLcond
d                 notLLLcondiso
diso              post
                  px
error             py
extendedLLL       q
finished          r
four              rank
gcd               rat
gram              ratiolist
gramLLL           red
gramorg           rules
i                 setratio
incGS             swap
init              swaptail
inner             swaptailiso
integerbody       t
isodim            testgramm
                  testLLLcondition
j                 testm
k                 three
kmax              trickledown
l                 v
la                val
LLLbranch         w
LLLratio          x
LLLratiolist      y
*)

(*                      MAIN INTERFACE                                  *)

ExtendedLatticeReduce[m_,rules___Rule]:=
(
LLLratio=(RatioLLLCondition/.{rules}/.Options[ExtendedLatticeReduce]);
extendedLLL[m]
)

GramLatticeReduce[m_,rules___Rule]:=
```

```
(
LLLratio=(RatioLLLCondition/.{rules}/.Options[GramLatticeReduce]);
gramLLL[m]
)

extendedLLL[m_]:=
CheckAbort[(
           testm[m];
           integerbody;answer={Take[b.basis,-rank],b}
           )
           ,$Failed
          ]

gramLLL[m_]:=
CheckAbort[(
           testgramm[m];
           integerbody;answer={rank,b}
           )
           ,$Failed
          ]

(*                    SUBROUTINES                                    *)

(* ----- Purpose of testm:
Test if main argument of ExtendedLatticeReduce is legal.
Compute gram matrix.
-------  *)

testm[m_]:=
(
basis=m;
(* begin check argument *)
error:=(Message[ExtendedLatticeReduce::argint,basis];Abort[]);
If[MatrixQ[basis],
   Map[(If[ #[[0]]===Integer,,error])&,basis,{2}]
  ,
  error
  ];
```

```
(* end check argument *)
gram=basis.Transpose[basis];
If[MatrixQ[gram],,error]
(* To catch "matrices" like {{}}, whose transpose is not a matrix. *)
);

(* ----- Purpose of testgramm:
Test if main argument of GramLatticeReduce is legal.
(But positive semi definiteness will have to wait.)
Remove denominators in gram matrix.
-------  *)

testgramm[m_]:=
(
gramorg=gram=m;
(* begin check argument *)
error:=(Message[GramLatticeReduce::argrat,gram];Abort[]);
If[MatrixQ[gram],
   Map[(If[ #[[0]]===Integer || #[[0]]===Rational,,error])&,gram,{2}]
  ,
  error
  ];
If[Length[gram]!=Length[gram[[1]]],error];
If[Length[Union@@(gram-Transpose[gram])]>1,error];
(* end check argument *)
If[Count[gram,_Rational,{2}]>0,
   gram=LCM@@(Union@@(Map[Denominator,gram,{2}]))gram
  ];
);

(* ----- Purpose of integerbody:
Body of integer arithmetic algorithm.
-------  *)

integerbody:=
(
For[init;
    ,
```

```
    moreratios
    ,
    moreratios=(LLLratiolist=!={})
    ,
    For[setratio,!finished,,testLLLcondition]; (* main loop *)
    ];
post;
)

(* ----- Purpose of init:
Initializing for integer arithmetic.
-------  *)

init:=(
n=Length[gram]; (* Number of vectors *)
b=Hold@@(IdentityMatrix[n]); (* Will be the transformation matrix *)
d=Hold@@(Range[n+1]);  (* Will be a list of denominators *)
la=Hold@@(Table[0,{i,n},{j,i-1}]); (* Will be list of numeratos *)
(* The Hold wrapper is used here because Part does not have Attribute  *)
(* HoldFirst. We want to use Part frequently, without re-evaluating    *)
(* all entries each time. This starts to matter when n gets large.     *)
diso=d; (* Will also be a list of denominators *)
rank=isodim=0;(* isodim is dimension of isotropic subspace *)
LLLratiolist=ratiolist[LLLratio];(* The quality ratios. The default is 3/4. *)
moreratios=True;
kmax=0;(* Number of vectors for which tables are filled. *)
k=1; (* The index of the vector on which attention is focussed *)
incGS;
);

(* ----- Purpose of ratiolist:
Interface for the option RatioLLLCondition.
-------  *)

ratiolist[r_Rational]:={r};

ratiolist[{r__Rational}]:={r};
```

```
ratiolist[r_]:=(Message[RatioLLLCondition::argrat,r];Abort[]);

(* ----- Purpose of setratio:
Process the option RatioLLLCondition and help initializing.
-------  *)

setratio:=(
LLLratio=First[LLLratiolist];
If[(1/4<LLLratio<1)
  ,
  three=Numerator[LLLratio];
  four=Denominator[LLLratio];
  LLLratiolist=Rest[LLLratiolist];
  ,
  Message[RatioLLLCondition::argrat,LLLratio];Abort[]
  ];
k=2;
finished=False;
If[(k<=n)
  ,
   incGS
  ,
   finished=True
  ]
);

(* ----- Purpose of post:
Removing the Hold wrapper.
-------  *)

post:=
(
b=List@@b;
)

(* ----- Purpose of incGS:
If b[[k]] is new, extend tables.
If existence of other isotropic vector is detected, go find it and
```

```
reorganize, changing "everything".
-------  *)

incGS:=
(
If[k>kmax
  ,
  kmax=k;
  Do[coefficient[k,j]
    ,{j,isodim+1,k}
    ];
  Do[coefficientiso[k,j]
    ,{j,isodim}
    ];
  If[d[[k+1]]<=0,trickledown,rank++]
  ];
);

(* ----- Purpose of inner:
The inner product described by the gram matrix.
-------  *)

inner[v_,w_]:=v.gram.w;(* used only in one place *)

(* ----- Purpose of coefficientiso[k,j]:
Computes la[[k,j]] or diso[[k+1]] assuming tables are up to date for smaller
indices. It assumes j<=isodim, as it works with the euclidean inner product
on the rows of the transformation matrix b.
-------  *)

coefficientiso[k_,j_]:=
(
q=Dot[b[[k]],b[[j]]];
Do[q=(diso[[i+1]]q-la[[k,i]]la[[j,i]])~Quotient~diso[[i]],{i,j-1}];
If[j<k,la[[k,j]]=q,diso[[k+1]]=q]
);

(* ----- Purpose of coefficient[k,j]:
```

```
Computes la[[k,j]] or d[[k+1]] assuming tables are up to date for smaller
indices. It assumes j>isodim, as it works with the inner product given
by the gram matrix.
-------  *)

coefficient[k_,j_]:=
(
q=inner[b[[k]],b[[j]]];
Do[q=(d[[i+1]]q-la[[k,i]]la[[j,i]])~Quotient~d[[i]],{i,isodim+1,j-1}];
If[j<k,la[[k,j]]=q,d[[k+1]]=q]
);

(* ----- Purpose of testLLLcondition:
After straightening b[[k]] with respect to b[[k-1]], move to testing
the relevant LLL condition, if any.
-------  *)

testLLLcondition:=
(
red[k-1];
If[k>isodim+1
   ,
   LLLbranch[notLLLcond];
   ,
   LLLbranch[(k!=isodim+1)&&notLLLcondiso]
  ]
);

(* ----- Purpose of LLLbranch[cond]:
Depending on cond, take branch in LLL algorithm.
-------  *)

LLLbranch[cond_]:=(* given k *)
If[cond
  ,
  swap;
  k=Max[2,k-1]
  ,
```

```
  For[l=k-2,l>0,l--,red[l]];
  k++;
  If[k<=n,incGS,finished=True]
  ];


(* ----- Purpose of notLLLcond:
Tell if the LLL condition is violated for the inner product given by the
gram matrix.
   ----- Purpose of notLLLcondiso:
Tell if the LLL condition is violated for the euclidean inner product
on the rows of the transformation matrix b.
-------  *)

notLLLcond:=(* given k *)
four*d[[k+1]]d[[k-1]]<(three*d[[k]]^2-four*la[[k,k-1]]^2);

notLLLcondiso:=(* given k *)
four*diso[[k+1]]diso[[k-1]]<(three*diso[[k]]^2-four*la[[k,k-1]]^2);

(* ----- Purpose of red[l]:
Subtract appropriate multiple of b[[l]] from b[[k]] and update table la
accordingly.
-------  *)

red[l_]:=(* given k *)
(
t=If[l>isodim,d[[l+1]],diso[[l+1]]];
If[Abs[2la[[k,l]]]>t
  ,
  q=Quotient[2la[[k,l]]+t,2t];
  b[[k]]=b[[k]]-q b[[l]];
  la[[k,l]]=la[[k,l]]-q t;
  Do[la[[k,i]]=la[[k,i]]-q la[[l,i]];,{i,l-1}]
  ]
);

(* ----- Purpose of swap:
```

```
Swap b[[k-1]] with b[[k]] and update tables la, d, diso
accordingly.
-------  *)

swap:=(* given k *)
(
{b[[k]],b[[k-1]]}={b[[k-1]],b[[k]]};
Do[{la[[k-1,j]],la[[k,j]]}={la[[k,j]],la[[k-1,j]]},{j,k-2}];
q=la[[k,k-1]];
If[k>isodim
   ,
   swaptail
   ,
   swaptailiso
   ]
);

(* ----- Purpose of swaptail:
Finish updating tables la and d in a swap.
-------  *)

swaptail:=(* given k, q *)
(
bbb=(d[[k-1]]d[[k+1]]+q^2)~Quotient~d[[k]];
Do[
   t=la[[i,k]];
   la[[i,k]]=(d[[k+1]]la[[i,k-1]]-q t)~Quotient~d[[k]];
   la[[i,k-1]]=(bbb t+q la[[i,k]])~Quotient~d[[k+1]]
   ,{i,k+1,kmax}
  ];
d[[k]]=bbb
);

(* ----- Purpose of swaptailiso:
Finish updating tables la and diso in a swap.
-------  *)

swaptailiso:=(* given k, q *)
```

```
(
bbb=(diso[[k-1]]diso[[k+1]]+q^2)~Quotient~diso[[k]];
Do[
   t=la[[i,k]];
   la[[i,k]]=(diso[[k+1]]la[[i,k-1]]-q t)~Quotient~diso[[k]];
   la[[i,k-1]]=(bbb t+q la[[i,k]])~Quotient~diso[[k+1]]
   ,{i,k+1,kmax}
  ];
diso[[k]]=bbb
);

(* ----- Purpose of matextendedgcd[x,y]:
Compute the gcd and the matrix implicit in the euclidean algorithm.
-------  *)

matextendedgcd[x_,y_]:=
({gcd,{px,py}}=ExtendedGCD[x,y];{{-y~Quotient~gcd,x~Quotient~gcd},{px,py}});

(* ----- Purpose of trickledown:
To compute another vector which is isotropic with respect to the inner product
given by the gram matrix and to put it before the nonisotropic vectors.
To adapt b, isodim, k, la, d, diso to the new situation.
-------  *)

trickledown:=
(
(*                                                                    *)
If[d[[k+1]]<0,Message[GramLatticeReduce::neg,gramorg];Abort[]];
rat=Hold@@(Table[1,{kmax}]); (* Hold@@ is not essential, cf. comment below *)
Do[
   k--;
   mat=matextendedgcd[d[[k+1]],la[[k+1,k]]];
   rat[[k+1]]=mat[[1,2]];
   {b[[k]],b[[k+1]]}=mat.{b[[k]],b[[k+1]]};
   Do[{la[[k,j]],la[[k+1,j]]}=mat.{la[[k,j]],la[[k+1,j]]},{j,k-1}];
   For[l=k-1,l>0,l--,red[l]];
  ,
  {rank}
```

```
  ];
isodim++;(* "null space part" = isotropic part of b is increased *)
rat=Rest[FoldList[Times,1,rat]]; (* cumulative volume ratios *)
(* Only now do we update d and la *)
Do[
   d[[k+1]]=(d[[k]])~Quotient~(rat[[k]]^2);
   q=rat[[k]]rat[[k-1]];
   Do[la[[j,k]]=Quotient[la[[j,k-1]],q],{j,k+1,kmax}];
   ,
   {k,kmax,isodim+1,-1}
  ];
d[[isodim+1]]=1;
Do[coefficientiso[k,isodim],{k,isodim,kmax}];
Do[red[l],{k,isodim,kmax},{l,k-1,1,-1}];
k=Max[isodim,2]
);


(*                MORE OF THE  INTERFACE                          *)


ExtendedLatticeReduce[m__,{rules___},more___]:=
ExtendedLatticeReduce[m,rules,more]

ExtendedLatticeReduce[___]:=(
Message[ExtendedLatticeReduce::argw,val];$Failed);

GramLatticeReduce[m__,{rules___},more___]:=
GramLatticeReduce[m,rules,more]

GramLatticeReduce[___]:=(
Message[GramLatticeReduce::argw];$Failed);



(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)
(*                          Error messages.                             *)
```

```
ExtendedLatticeReduce::argint:=
"Argument ‘1‘ of ExtendedLatticeReduce should be a matrix with integral
entries."

ExtendedLatticeReduce::argw:="ExtendedLatticeReduce called with wrong
number of arguments. One argument is expected.";

GramLatticeReduce::argrat:=
"Argument ‘1‘ of GramLatticeReduce should be a symmetric matrix
with rational entries."

GramLatticeReduce::argw:="GramLatticeReduce called with wrong number of
arguments. One argument is expected.";

GramLatticeReduce::neg:="Gram matrix ‘1‘ has negative eigenvalue; it should
be positive semi definite."

RatioLLLCondition::argrat:=
"Value ‘1‘ of RatioLLLCondition should be a rational number
between 1/4 and 1, or a list of such numbers."

(* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - *)

End[]

Protect[ExtendedLatticeReduce,GramLatticeReduce,RatioLLLCondition];

EndPackage[]

Null
```

9 Implementation in GP/PARI

```
\\                                                                      \\
\\  Extended Lenstra Lenstra Lovasz algorithm.                          \\
\\                                                                      \\
\\  NAME extendedlll                                                    \\
\\                                                                      \\
\\  DESCRIPTION                                                         \\
\\                                                                      \\
\\  By default extendedlll computes LLL reduced bases of both the lattice  \\
\\  L spanned by the generating set and the lattice R of integer        \\
\\  relations between the generators. It is an extended algorithm in    \\
\\  that it returns the needed transformation as second entry.          \\
\\                                                                      \\
\\  SYNOPSIS                                                            \\
\\                                                                      \\
\\  extendedlll(m[,isgram[,reducecolumnspaceoff[,reducerelationsoff]]])  \\
\\                                                                      \\
\\  Here isgram, reducecolumnspaceoff, reducerelationsoff are 0 or 1.   \\
{\

\\  PROGRAM  \\

extendedlll(X,\
\\                              X is matrix with integer entries        \\

\\         OPTIONS                                                      \\

           isgram,\
\\                              isgram is an option to indicate,        \\
\\                              by means of value 1,                    \\
\\                              that the matrix X should be understood  \\
\\                              as Gram matrix. If isgram is 0 or absent,  \\
\\                              then the columns of the matrix X are    \\
\\                              understood as generators of the lattice L. \\
           reducecolumnspaceoff,\
\\                              option to indicate if nonreduced basis of  \\
\\                              the lattice L is acceptable.            \\
```

```
          reducerelationsoff,\
\\                            option to indicate if nonreduced basis of  \\
\\                            the relation lattice R is acceptable.      \\

\\        LOCAL VARIABLES                                                 \\

          B,\
\\           table of denominators                                        \\
          Gram,\
\\              Gram matrix                                               \\
          H,\
\\           transformation matrix                                        \\
          Lam,\
\\             table of numerators                                        \\
          R,
\\          table of volume ratios                                        \\

          A,C,D,Matr11,Matr12,Matr21,Matr22,Q,Q2,T1,T3,Tmp,Tmp2,V1,V3,\
          isodim,k,kk,kmax,l,ll,n,nxt,notfinished,rnk,s,s1,s2)\

\\        LOCAL LOOP VARIABLES ii,j                                       \\

=\
\\  INITIALIZATION  \\

\\ Check matrix argument. \\
if(type(X)!="t_MAT",print("extendedlll: argument is no matrix");1/0,);
Q2=matsize(X);
k=Q2[2];
for(ii=1,Q2[1],\
    for(j=1,k,\
        if(type(X[ii,j])!="t_INT",\
          print("extendedlll: matrix entry is no integer");1/0\
          ,\
          )\
       )\
   );
\\ Get Gram matrix. \\
```

```
if(isgram,\
  Gram=X;
  if(Gram==Gram~,\
    ,\
    print("extendedlll: Gram matrix is not symmetric.");1/0\
\\  Actually one may wish to use that the entries below the diagonal do   \\
\\  not affect the computation, and thus forget the symmetry requirement. \\
    );
  ,\
  Gram=(X~)*X\
  );
s1=if((1-reducecolumnspaceoff),1,0);
s2=if((1-reducerelationsoff),1,0);
notfinished=1;
n=matsize(Gram)[1];
H=matid(n);
\\ The array B must be long enough to keep the isotropic part separate. \\
B=vector(n+2,ii,1);
Lam=H;
rnk=0;
isodim=0;
kmax=0;
k=1;
kmax=k;
\\ Gram Schmidt coefficients of first generator. \\
B[k+2]=(H[,k]~)*Gram[,k];
s=sign(B[k+2]);
if(s<=0,\
\\ It is a relation. The rank of the lattice R of relations increases. \\
  isodim=isodim+1;
  if(s,\
    print("extendedlll: Gram matrix is not positive semi-definite.");1/0\
    ,\
    );
  B[isodim+2]=1;
\\ Gram Schmidt coefficients of relation. Here the ordinary inner \\
\\ product is relevant, not the one given by Gram. For the        \\
\\ latter, relations are isotropic. Whence the name isodim.       \\
```

```
  B[k+1]=(H[,k]~)*(H[,k]);
  ,\
\\ It is not a relation, and rank of lattice L increases. \\
  rnk=rnk+1\
  );
nxt=0;
k=2;
if((k<=n),nxt=1,notfinished=0);
\\  MAIN LOOP  \\
while(notfinished,\
      if(nxt,\
        if(k>kmax,\
\\        Add generator.     \\
          kmax=k;
\\        Gram Schmidt coefficients of new generator.        \\
          for(j=isodim+1,k,\
              Q=(H[,j]~)*Gram[,k];
              for(ii=isodim+1,j-1,Q=(B[ii+2]*Q-Lam[k,ii]*Lam[j,ii])/B[1+ii]);
              if(j<k,Lam[k,j]=Q,B[k+2]=Q)\
             );
          s=sign(B[k+2]);
          if(s<=0,\
\\          New relation expected.     \\
            if(s,\
              print("extendedlll: Gram matrix has negative eigenvalue.");1/0\
              ,\
              );
            R=vector(kmax,ii,1);
\\          Push generator down into subspaces.          \\
            for(ii=1,rnk,\
                k=k-1;
\\              Extended Euclid.              \\
                A=B[k+2];C=Lam[k+1,k];Matr21=1;D=A;
                if(C,\
                  V1=0;V3=C;
                  while(V3,\
                        Q2=divrem(D,V3);Q=Q2[1];T3=Q2[2];
                        T1=Matr21-Q*V1;Matr21=V1;D=V3;V1=T1;V3=T3\
```

```
                    );
                  Matr22=(D-A*Matr21)/C;
                  ,\
                  Matr22=0\
                 );
               Matr11=-C/D;Matr12=A/D;
\\             Matrix which pushes generator one down is           \\
\\             Matr=[Matr11,Matr12;Matr21,Matr22];                 \\
\\             To estimate its 11 entry, note                       \\
\\             Matr11 == - (C/A)*Matr12                            \\

\\             Volume ratio between old and new sublattice.              \\
               R[k+1]=Matr12;
\\             Push one down.              \\
               Tmp=Matr11*H[,k]+Matr12*H[,k+1];
               H[,k+1]=Matr21*H[,k]+Matr22*H[,k+1];H[,k]=Tmp;
\\             Update part of Lam.              \\
               for(j=1,k-1,\
                   Q=Matr11*Lam[k,j]+Matr12*Lam[k+1,j];
                   Lam[k+1,j]=Matr21*Lam[k,j]+Matr22*Lam[k+1,j];Lam[k,j]=Q\
                  )\
              );
\\          Add new relation.          \\
            isodim=isodim+1;
\\          Cumulative volume ratios.          \\
            for(ii=2,kmax,R[ii]=R[ii]*R[ii-1]);
\\          Only now do we update B and Lam.          \\
\\          Thus we avoided a lot of updating.       \\
            forstep(k=kmax,isodim+1,-1,\
                    B[k+2]=(B[1+k])/(R[k]^2);
\\                  These are integers, implying a bound on R[k].          \\
                    Q=R[k]*R[k-1];
                    for(j=k+1,kmax,Lam[j,k]=(Lam[j,k-1])/(Q));
                   );
            B[isodim+2]=1;
\\          Gram Schmidt coefficients of new relation.          \\
            for(k=1,isodim,\
                kk=isodim;ll=k;Q=(H[,kk]~*(H[,ll]));
```

```
                for(ii=1,ll-1,Q=(B[ii+1]*Q-Lam[kk,ii]*Lam[ll,ii])/B[ii]);
                if(ll<kk,Lam[kk,ll]=Q,B[kk+1]=Q)\
               );
\\          B[isodim+1] <= (order kmax leading minor of matid(n)+Gram)     \\
            for(k=isodim+1,kmax,\
                kk=k;ll=isodim;Q=(H[,kk]~)*(H[,ll]);
                for(ii=1,ll-1,Q=(B[ii+1]*Q-Lam[kk,ii]*Lam[ll,ii])/B[ii]);
                if(ll<kk,Lam[kk,ll]=Q,B[kk+1]=Q)\
               );
            k=max(isodim,2)\
            ,\
\\          It is not a relation, and rank of lattice L increases.         \\
            rnk=rnk+1\
            )\
          ,\
          );
        nxt=0\
        ,\
\\      Test for swap.      \\
\\      First reduce generator k with generator k-1.      \\
        l=(k-1);A=if(l>isodim,B[l+2],B[l+1]);
        if(abs(2*Lam[k,l])>A,\
          Q=((2*Lam[k,l]+A)\(2*A));
          H[,k]=H[,k]-Q*H[,l];
          Lam[k,l]=Lam[k,l]-Q*A;
          for(ii=1,l-1,Lam[k,ii]=Lam[k,ii]-Q*Lam[l,ii])\
          ,\
          );
\\      Compute if condition for swap is satisfied.      \\
        cond=if(k>isodim+1,\
\\              Test involves no relations.               \\
                s1&&(4*B[k+2]*B[k]<(3*B[1+k]^2-4*Lam[k,k-1]^2))\
                ,\
\\              Test involves relations.              \\
                s2&&((k!=isodim+1)&&\
                4*B[k+1]*B[k-1]<(3*B[k]^2-4*Lam[k,k-1]^2))\
                );
        if(cond,\
```

```
\\        Swap.        \\
          Tmp=H[,k-1];H[,k-1]=H[,k];H[,k]=Tmp;
          for(j=1,k-2,Q=Lam[k,j];Lam[k,j]=Lam[k-1,j];Lam[k-1,j]=Q);
          Q=Lam[k,k-1];
          if(k>isodim,\
\\          More Gram Schmidt coefficients to be updated.          \\
            u=(B[k]*B[k+2]+Q^2)/B[1+k];
            for(ii=k+1,kmax,\
                A=Lam[ii,k];
                Lam[ii,k]=(B[k+2]*Lam[ii,k-1]-Q*A)/B[1+k];
                Lam[ii,k-1]=(u*A+Q*Lam[ii,k])/B[k+2]\
               );
            B[1+k]=u\
            ,\
\\          Gram Schmidt coefficients that involve relations.          \\
            u=(B[k-1]*B[k+1]+Q^2)/B[k];
            for(ii=k+1,kmax,\
                A=Lam[ii,k];
                Lam[ii,k]=(B[k+1]*Lam[ii,k-1]-Q*A)/B[k];
                Lam[ii,k-1]=(u*A+Q*Lam[ii,k])/B[k+1]\
               );
            B[k]=u\
            );
          k=max(2,k-1)\
          ,\
\\        No swap required. Reduce generator k with         \\
\\        generators k-2 down to 1.                         \\
          forstep(l=k-2,1,-1,\
                  A=if(l>isodim,B[l+2],B[l+1]);
                  if(abs(2*Lam[k,l])>A,\
                    Q=((2*Lam[k,l]+A)\(2*A));
                    H[,k]=H[,k]-Q*H[,l];
                    Lam[k,l]=Lam[k,l]-Q*A;
                    for(ii=1,l-1,Lam[k,ii]=Lam[k,ii]-Q*Lam[l,ii])\
                    ,\
                    )\
                 );
\\        Go up.        \\
```

```
          k=k+1;
          if(k<=n,nxt=1,notfinished=0)\
          )\
        )\
   );
[if(isgram,\
\\  If isgram is 1, then the first entry of the output is the rank of L. \\
    rnk,\
\\  Otherwise the columns of the first entry form a basis of L. \\
\\  This basis is reduced unless reducecolumnspaceoff is 1.     \\
    vecextract(X*H,\
               vector(matsize(X)[1],ii,ii),\
               vector(rnk,ii,n-rnk+ii))\
              )\
,\
\\ The second entry of the output is the required transformation,  \\
\\ as acting on the matrix whose columns span L.                   \\
H
]}
```

References

[C] H. Cohen, *A course in computational Algebraic Number Theory*, Graduate Texts in Mathematics 138, Springer 1993.

[LLL] A.K. Lenstra, H.W. Lenstra Jr. and L. Lovász, *Factoring polynomials with rational coefficients,* Math. Ann. **261** (1982), 515–534.

[P] M. Pohst, *A modification of the LLL-algorithm,* J. Symb. Comp. **4** (1987), 123–128.