Accelerating lattice reduction algorithms with floating-point arithmetic

Damien Stehlé

http://perso.ens-lyon.fr/damien.stehle/

LIP – CNRS/ENSL/INRIA/UCBL/U. Lyon

MaGiX@LiX, September 2011

Goals and plan of the talk

Goals:

- To describe efficient techniques for lattice reduction.
- To illustrate how numerical linear algebra can be rigorously used to accelerate an algebraic computation.

Plan of the talk:

1. Reminders on Euclidean lattices.
2. Using floating-point arithmetic within lattice algorithms.
3. The fplll library.

Goals and plan of the talk

Goals:

- To describe efficient techniques for lattice reduction.
- To illustrate how numerical linear algebra can be rigorously used to accelerate an algebraic computation.

Plan of the talk:

1. Reminders on Euclidean lattices.
2. Using floating-point arithmetic within lattice algorithms.
3. The fplll library.

Euclidean lattices

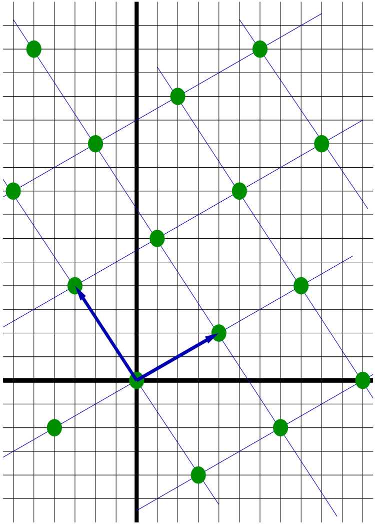
Lattice $\equiv \{\sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$.

If the $\mathbf{b}_i$'s are linearly independent, they are called a basis.

Bases are not unique, but can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$

Lattice reduction:
find a nice basis, given an arbitrary one.



Euclidean lattices

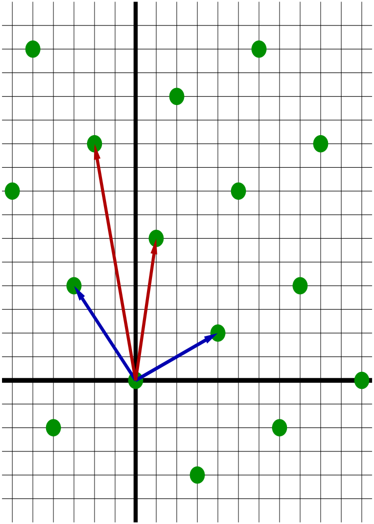
Lattice $\equiv \{\sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$.

If the $\mathbf{b}_i$'s are linearly independent, they are called a basis.

Bases are not unique, but can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$

Lattice reduction:
find a nice basis, given an arbitrary one.



Euclidean lattices

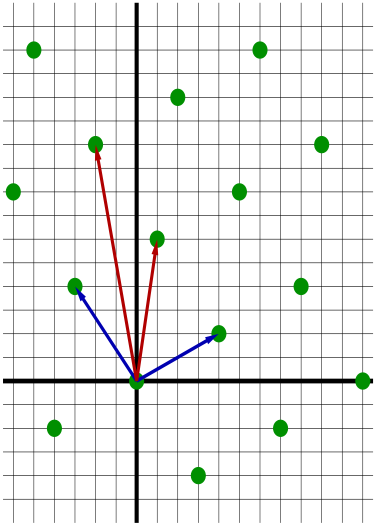
Lattice $\equiv \{\sum_{i \leq n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$.

If the $\mathbf{b}_i$'s are linearly independent, they are called a basis.

Bases are not unique, but can be obtained from each other by integer transforms of determinant ± 1:

$$\begin{bmatrix} -2 & 1 \\ 10 & 6 \end{bmatrix} = \begin{bmatrix} 4 & -3 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$

Lattice reduction:
find a nice basis, given an arbitrary one.



Lattice invariants and lattice reduction

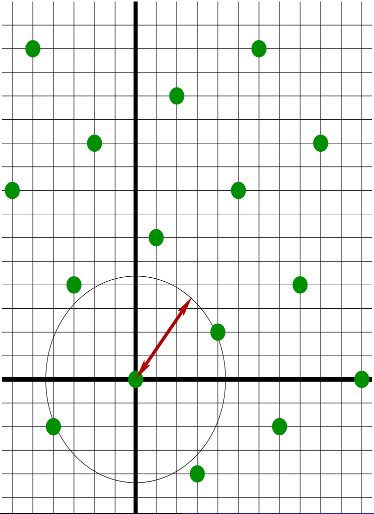
Minimum:
$\lambda(L) = \min\left(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0}\right)$.

Lattice determinant:
$\det L = \left|\det(\mathbf{b}_i)_i\right|$, for any basis.

Minkowski's theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$.

Lattice reduction:
Find basis $(\mathbf{b}_i)_i$ s.t. $\mathrm{HF}(B)$ is small, with

$$\mathrm{HF}(B) := \frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}.$$



Lattice invariants and lattice reduction

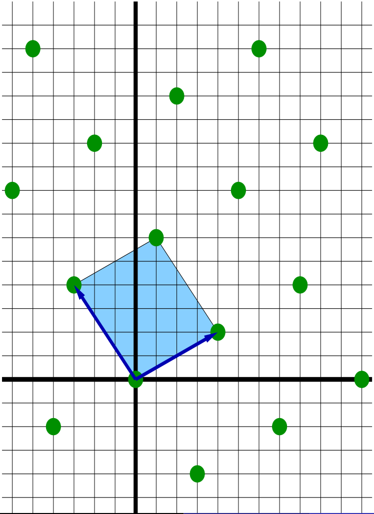
Minimum:
$\lambda(L) = \min\left(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0}\right)$.

Lattice determinant:
$\det L = \left|\det(\mathbf{b}_i)_i\right|$, for any basis.

Minkowski's theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$.

Lattice reduction:
Find basis $(\mathbf{b}_i)_i$ s.t. $\mathrm{HF}(B)$ is small, with

$$\mathrm{HF}(B) := \frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}.$$



Lattice invariants and lattice reduction

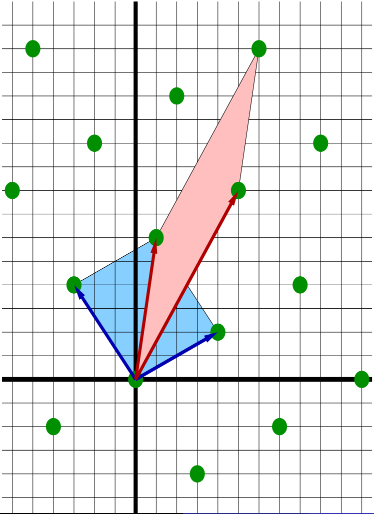
Minimum:
$\lambda(L) = \min\ (\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$.

Lattice determinant:
$\det L = |\det(\mathbf{b}_i)_i|$, for any basis.

Minkowski's theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$.

Lattice reduction:
Find basis $(\mathbf{b}_i)_i$ s.t. $\mathrm{HF}(B)$ is small, with

$$\mathrm{HF}(B) := \frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}.$$



Lattice invariants and lattice reduction

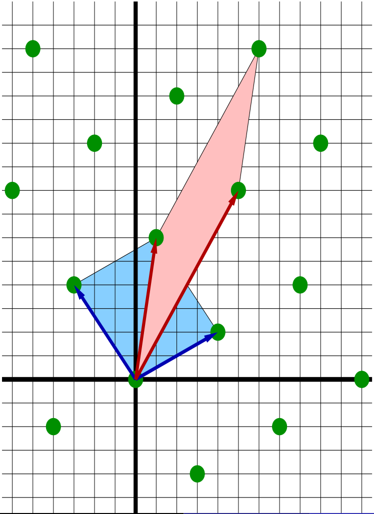
Minimum:
$\lambda(L) = \min \left(\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0}\right)$.

Lattice determinant:
$\det L = \left|\det(\mathbf{b}_i)_i\right|$, for any basis.

Minkowski's theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$.

Lattice reduction:
Find basis $(\mathbf{b}_i)_i$ s.t. $\mathrm{HF}(B)$ is small, with

$$\mathrm{HF}(B) := \frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}.$$



Lattice invariants and lattice reduction

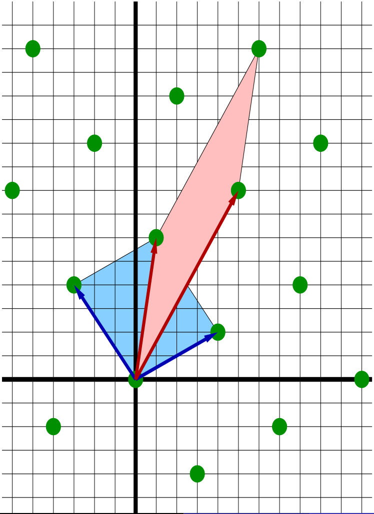
Minimum:
$\lambda(L) = \min\ (\|\mathbf{b}\| : \mathbf{b} \in L \setminus \mathbf{0})$.

Lattice determinant:
$\det L = |\det(\mathbf{b}_i)_i|$, for any basis.

Minkowski's theorem:
$\lambda(L) \leq \sqrt{n} \cdot (\det L)^{1/n}$.

Lattice reduction:
Find basis $(\mathbf{b}_i)_i$ s.t. $\mathrm{HF}(B)$ is small, with

$$\mathrm{HF}(B) := \frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}.$$



Main computational problems

- SVP_γ: Given a basis of L, find $\mathbf{b} \in L$ with
$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$
- BDD_γ: Given a basis of L and $\mathbf{t}$ with
$$\mathrm{dist}(\mathbf{t}, L) \leq \gamma^{-1} \cdot \lambda(L),$$
find $\mathbf{b} \in L$ closest to $\mathbf{t}$.
- And many variants: CVP_γ, SIVP_γ, uSVP_γ, etc.
- Very hard for small γ: CVP, SIVP, uSVP, and SVP are NP-hard under (randomized) reductions.
- "Easy" for exponential γ.

Main computational problems

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with
$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$
- BDD$_\gamma$: Given a basis of L and $\mathbf{t}$ with
$$\mathrm{dist}(\mathbf{t}, L) \leq \gamma^{-1} \cdot \lambda(L),$$
find $\mathbf{b} \in L$ closest to $\mathbf{t}$.
- And many variants: CVP$_\gamma$, SIVP$_\gamma$, uSVP$_\gamma$, etc.
- Very hard for small γ: CVP, SIVP, uSVP, and SVP are NP-hard under (randomized) reductions.
- "Easy" for exponential γ.

Main computational problems

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with
$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$
- BDD$_\gamma$: Given a basis of L and $\mathbf{t}$ with
$$\mathrm{dist}(\mathbf{t}, L) \leq \gamma^{-1} \cdot \lambda(L),$$
find $\mathbf{b} \in L$ closest to $\mathbf{t}$.
- And many variants: CVP$_\gamma$, SIVP$_\gamma$, uSVP$_\gamma$, etc.

- Very hard for small γ: CVP, SIVP, uSVP, and SVP are NP-hard under (randomized) reductions.
- "Easy" for exponential γ.

Main computational problems

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with
$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$
- BDD$_\gamma$: Given a basis of L and $\mathbf{t}$ with
$$\mathrm{dist}(\mathbf{t}, L) \leq \gamma^{-1} \cdot \lambda(L),$$
find $\mathbf{b} \in L$ closest to $\mathbf{t}$.
- And many variants: CVP$_\gamma$, SIVP$_\gamma$, uSVP$_\gamma$, etc.

- Very hard for small γ: CVP, SIVP, uSVP, and SVP are NP-hard under (randomized) reductions.
- "Easy" for exponential γ.

Main computational problems

- SVP$_\gamma$: Given a basis of L, find $\mathbf{b} \in L$ with
$$0 < \|\mathbf{b}\| \leq \gamma \cdot \lambda(L).$$
- BDD$_\gamma$: Given a basis of L and $\mathbf{t}$ with
$$\mathrm{dist}(\mathbf{t}, L) \leq \gamma^{-1} \cdot \lambda(L),$$
  find $\mathbf{b} \in L$ closest to $\mathbf{t}$.
- And many variants: CVP$_\gamma$, SIVP$_\gamma$, uSVP$_\gamma$, etc.

- Very hard for small γ: CVP, SIVP, uSVP, and SVP are NP-hard under (randomized) reductions.
- "Easy" for exponential γ.

All known algorithms rely on some kind of lattice reduction.

Why do we care about lattices?

Lattices tend to pop out every time one wants to use linear algebra but is restricted to discrete transformations.

- Computer algebra: factorisation of rational polynomials, reconstruction of algebraic numbers.

  Given α algebraic of degree n, the shortest vector in the lattice

$$L := L[(\mathbf{b}_i)_i], \quad \text{with } B = \begin{bmatrix} C & C\alpha & C\alpha^2 & \dots & C\alpha^n \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

  leads to the minimal polynomial of α (for some large C).

Why do we care about lattices?

Lattices tend to pop out every time one wants to use linear algebra but is restricted to discrete transformations.

- Computer algebra: factorisation of rational polynomials, reconstruction of algebraic numbers.
  Given α algebraic of degree n, the shortest vector in the lattice

$$L := L[(\mathbf{b}_i)_i], \quad \text{with } B = \begin{bmatrix} C & C\alpha & C\alpha^2 & \dots & C\alpha^n \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

  leads to the minimal polynomial of α (for some large C).

Why do we care about lattices?

- Cryptography: cryptanalyses of variants of RSA.
  Coppersmith's methods [J. Crypto'98] allow the computation of all unexpectedly small roots of polynomials.
  Example [HerMay'10]: $n = 60$, entries up to $> 30,000$ bits.
- Communications theory: MIMO, GPS.

$$\mathbf{m} \in \mathbb{Z}^n \mapsto \mathbf{y} = H \cdot \mathbf{m} + \mathbf{e} \in \mathbb{R}^n.$$

  Knowing H and $\mathbf{y}$, find $\mathbf{m}$.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Why do we care about lattices?

- Cryptography: cryptanalyses of variants of RSA.
  Coppersmith's methods [J. Crypto'98] allow the computation of all unexpectedly small roots of polynomials.
  Example [HerMay'10]: $n = 60$, entries up to $> 30,000$ bits.
- Communications theory: MIMO, GPS.

$$\mathbf{m} \in \mathbb{Z}^n \mapsto \mathbf{y} = H \cdot \mathbf{m} + \mathbf{e} \in \mathbb{R}^n.$$

  Knowing H and $\mathbf{y}$, find $\mathbf{m}$.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Why do we care about lattices?

- Cryptography: cryptanalyses of variants of RSA.
  Coppersmith's methods [J. Crypto'98] allow the computation of all unexpectedly small roots of polynomials.
  Example [HerMay'10]: $n = 60$, entries up to $> 30,000$ bits.
- Communications theory: MIMO, GPS.

$$\mathbf{m} \in \mathbb{Z}^n \mapsto \mathbf{y} = H \cdot \mathbf{m} + \mathbf{e} \in \mathbb{R}^n.$$

  Knowing H and $\mathbf{y}$, find $\mathbf{m}$.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Why do we care about lattices?

- Cryptography: cryptanalyses of variants of RSA.
  Coppersmith's methods [J. Crypto'98] allow the computation of all unexpectedly small roots of polynomials.
  Example [HerMay'10]: $n = 60$, entries up to $> 30,000$ bits.
- Communications theory: MIMO, GPS.

  $$\mathbf{m} \in \mathbb{Z}^n \mapsto \mathbf{y} = H \cdot \mathbf{m} + \mathbf{e} \in \mathbb{R}^n.$$

  Knowing H and $\mathbf{y}$, find $\mathbf{m}$.
- Combinatorial optimisation, algorithmic group theory, algorithmic number theory, computer arithmetic, etc.

Several types of lattice reduction

	HKZ	BKZ_k	LLL
Hermite factor	$\sqrt{n}$	$\simeq \sqrt{k}^{\frac{n}{k}}$	$\simeq \sqrt{4/3}^{\frac{n}{2}}$
Time*	$2^{O(n)}$	$2^{O(k)} \times \mathrm{Poly}(n)$	$\mathrm{Poly}(n)$

*Number of arithmetic operations.

- HKZ = Hermite-Korkine-Zolotareff (19th c.).
- LLL = Lenstra-Lenstra-Lovász (1982).
- BKZ = Block Korkine-Zolotareff (Schnorr'87, Hanrot-Pujol-S.'11)

Several types of lattice reduction

	HKZ	BKZ_k	LLL
Hermite factor	$\sqrt{n}$	$\simeq \sqrt{k}^{\frac{n}{k}}$	$\simeq \sqrt{4/3}^{\frac{n}{2}}$
Time*	$2^{O(n)}$	$2^{O(k)} \times \mathrm{Poly}(n)$	$\mathrm{Poly}(n)$

*Number of arithmetic operations.

- HKZ = Hermite-Korkine-Zolotareff (19th c.).
- LLL = Lenstra-Lenstra-Lovász (1982).
- BKZ = Block Korkine-Zolotareff (Schnorr'87, Hanrot-Pujol-S.'11)

Several types of lattice reduction

	HKZ	BKZ_k	LLL
Hermite factor	$\sqrt{n}$	$\simeq \sqrt{k}^{\frac{n}{k}}$	$\simeq \sqrt{4/3}^{\frac{n}{2}}$
Time*	$2^{O(n)}$	$2^{O(k)} \times \mathrm{Poly}(n)$	$\mathrm{Poly}(n)$

*Number of arithmetic operations.

- HKZ = Hermite-Korkine-Zolotareff (19th c.).
- LLL = Lenstra-Lenstra-Lovász (1982).
- BKZ = Block Korkine-Zolotareff
  (Schnorr'87, Hanrot-Pujol-S.'11)

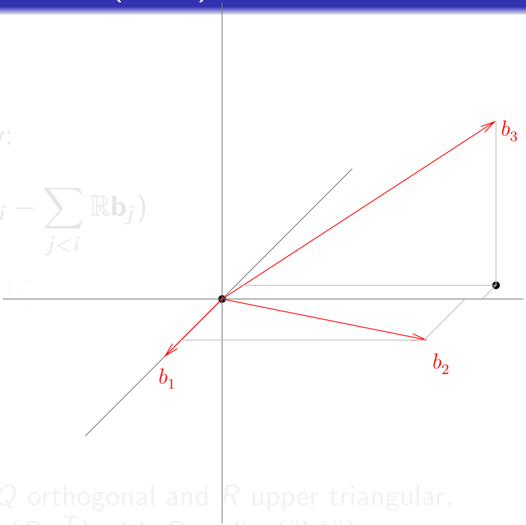
Gram-Schmidt orthogonalization (GSO)

$(\mathbf{b}_i)_i$ linearly independent.

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i, \quad \mathbf{b}_i^* = \mathrm{argmin}_{\|\cdot\|}(\mathbf{b}_i - \sum_{j<i} \mathbb{R}\mathbf{b}_j) = \mathbf{b}_i - \sum_{j<i} \mu_{i,j} \mathbf{b}_j^*$$

$$\forall i > j, \; \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}.$$



Equivalently: $B = QR$ with Q orthogonal and R upper triangular.
$B = (B^* D^{-1}) \cdot (D \mu^T)$ with $D = \mathrm{diag}(\|\mathbf{b}_i^*\|)$.

Gram-Schmidt orthogonalization (GSO)

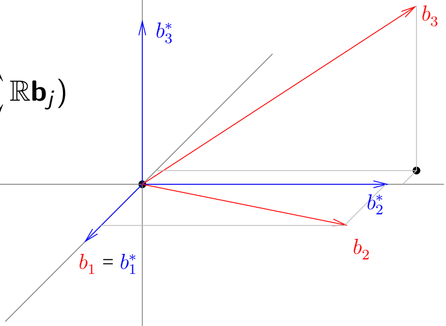
$(\mathbf{b}_i)_i$ linearly independent.

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i, \quad \mathbf{b}_i^* = \operatorname{argmin}_{\|\cdot\|}\Big(\mathbf{b}_i - \sum_{j<i} \mathbb{R}\mathbf{b}_j\Big)$$

$$= \mathbf{b}_i - \sum_{j<i} \mu_{i,j}\mathbf{b}_j^*$$

$$\forall i > j, \ \mu_{i,j} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}.$$



Equivalently: $B = QR$ with Q orthogonal and R upper triangular.

$B = (B^* D^{-1}) \cdot (D\mu^T)$ with $D = \operatorname{diag}(\|\mathbf{b}_i^*\|)$.

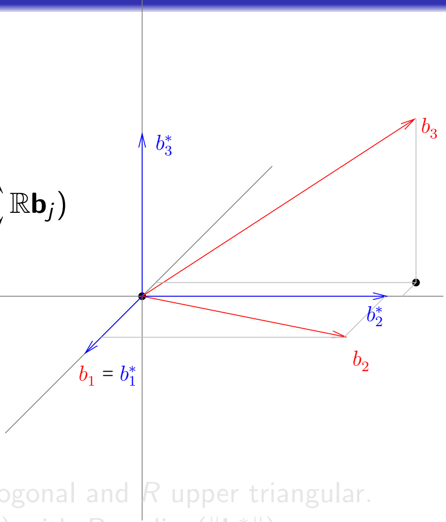
Gram-Schmidt orthogonalization (GSO)

$(\mathbf{b}_i)_i$ linearly independent.

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i, \quad \mathbf{b}_i^* = \operatorname{argmin}_{\|\cdot\|}(\mathbf{b}_i - \sum_{j<i} \mathbb{R}\mathbf{b}_j) = \mathbf{b}_i - \sum_{j<i} \mu_{i,j}\mathbf{b}_j^*$$

$$\forall i > j, \; \mu_{i,j} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}.$$



Equivalently: $B = QR$ with Q orthogonal and R upper triangular.
$B = (B^* D^{-1}) \cdot (D\mu^T)$ with $D = \operatorname{diag}(\|\mathbf{b}_i^*\|)$.

Gram-Schmidt orthogonalization (GSO)

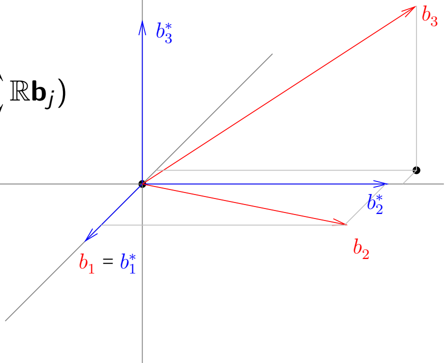
$(\mathbf{b}_i)_i$ linearly independent.

The GSO $(\mathbf{b}_i^*)_i$ is defined by:

$$\forall i, \quad \mathbf{b}_i^* = \operatorname{argmin}_{\|\cdot\|}(\mathbf{b}_i - \sum_{j<i} \mathbb{R}\mathbf{b}_j)$$

$$= \mathbf{b}_i - \sum_{j<i} \mu_{i,j}\mathbf{b}_j^*$$

$$\forall i > j, \ \mu_{i,j} = \frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{\|\mathbf{b}_j^*\|^2}.$$



Equivalently: $B = QR$ with Q orthogonal and R upper triangular.
$B = (B^* D^{-1}) \cdot (D\mu^T)$ with $D = \operatorname{diag}(\|\mathbf{b}_i^*\|)$.

The Lenstra-Lenstra-Lovász reduction (1982)

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \; |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i : \; \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

The Lenstra-Lenstra-Lovász reduction (1982)

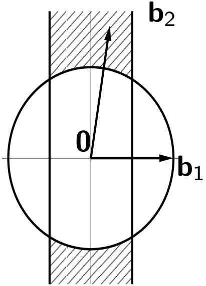
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \; |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i : \; \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

LLL-reduced bases have good quality:
The $r_{i,i}$'s can't drop too fast: $r_{i+1,i+1}^2 \geq (\delta - \frac{1}{4}) r_{i,i}^2$.

$$\begin{aligned} \|\mathbf{b}_1\| &\leq 2^{\mathcal{O}(n)} \cdot \lambda(L) \\ \prod \|\mathbf{b}_i\| &\leq 2^{\mathcal{O}(n^2)} \cdot |\det L|. \end{aligned}$$

Also allows one to solve BDD, CVP, SIVP, etc
with approximation factor $\gamma = 2^{\mathcal{O}(n)}$.



The Lenstra-Lenstra-Lovász reduction (1982)

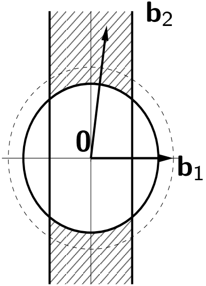
Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i : \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

LLL-reduced bases have good quality:
The $r_{i,i}$'s can't drop too fast: $r_{i+1,i+1}^2 \geq (\delta - \frac{1}{4}) r_{i,i}^2$.

$$\|\mathbf{b}_1\| \leq 2^{\mathcal{O}(n)} \cdot \lambda(L)$$
$$\prod \|\mathbf{b}_i\| \leq 2^{\mathcal{O}(n^2)} \cdot |\det L|.$$

Also allows one to solve BDD, CVP, SIVP, etc with approximation factor $\gamma = 2^{\mathcal{O}(n)}$.



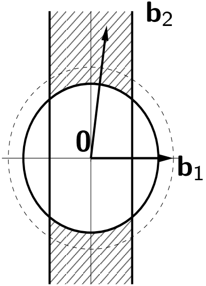
The Lenstra-Lenstra-Lovász reduction (1982)

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i : \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

LLL-reduced bases have good quality:
The $r_{i,i}$'s can't drop too fast: $r_{i+1,i+1}^2 \geq (\delta - \frac{1}{4}) r_{i,i}^2$.

$$\begin{aligned} \|\mathbf{b}_1\| &\leq 2^{\mathcal{O}(n)} \cdot \lambda(L) \\ \prod \|\mathbf{b}_i\| &\leq 2^{\mathcal{O}(n^2)} \cdot |\det L|. \end{aligned}$$



Also allows one to solve BDD, CVP, SIVP, etc with approximation factor $\gamma = 2^{\mathcal{O}(n)}$.

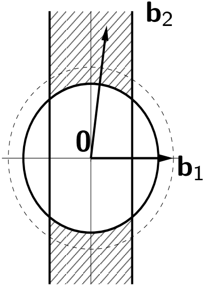
The Lenstra-Lenstra-Lovász reduction (1982)

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \le n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \ |r_{i,j}| \le r_{i,i}/2$ [size-reduction]
- $\forall i : \ \delta \cdot r_{i,i}^2 \le r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

LLL-reduced bases have good quality:
The $r_{i,i}$'s can't drop too fast: $r_{i+1,i+1}^2 \ge (\delta - \frac{1}{4}) r_{i,i}^2$.

$$\begin{aligned} \|\mathbf{b}_1\| &\le 2^{\mathcal{O}(n)} \cdot \lambda(L) \\ \prod \|\mathbf{b}_i\| &\le 2^{\mathcal{O}(n^2)} \cdot |\det L|. \end{aligned}$$



Also allows one to solve BDD, CVP, SIVP, etc with approximation factor $\gamma = 2^{\mathcal{O}(n)}$.

1. Reminders on Euclidean lattices.
2. **Using floating-point arithmetic within lattice algorithms.**
3. The fplll library.

The classical/rational LLL algorithm

Input: $(\mathbf{b}_i)_{i \leq n}$ linearly independent.

1. $j := 2$. While $j \leq n$, do:
2. Perform size-reduction for column j:
3. Compute them exactly.
4. For i from $j-1$ downto 1 do
5. $\mathbf{b}_j := \mathbf{b}_j - \lfloor r_{ij}/r_{ii} \rceil \mathbf{b}_i$.
6. Update the r_{ij}'s.
7. Test Lovasz's condition:
8. If $\delta \cdot r_{j-1,j-1}^2 \leq r_{jj}^2 + r_{j-1,j}^2$, then $j := j+1$.
9. Else swap $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$, $j := \max(j-1, 2)$.

- Assume $B \in \mathbb{Z}^{n \times n}$ with $\max \|\mathbf{b}_i\| \leq 2^\beta$.
- Number of loop iterations: $\mathcal{O}(n^2\beta/\log(1/\delta))$.
- Total bit-cost: $\mathcal{O}(n^5\beta^2(n+\beta))$ [Kaltofen'83].

The classical/rational LLL algorithm

Input: $(\mathbf{b}_i)_{i \leq n}$ linearly independent.

1. $j := 2$. While $j \leq n$, do:
2. Perform size-reduction for column j:
3. Compute them exactly.
4. For i from $j-1$ downto 1 do
5. $\mathbf{b}_j := \mathbf{b}_j - \lfloor r_{ij}/r_{ii} \rceil \mathbf{b}_i$.
6. Update the r_{ij}'s.
7. Test Lovasz's condition:
8. If $\delta \cdot r_{j-1,j-1}^2 \leq r_{jj}^2 + r_{j-1,j}^2$, then $j := j+1$.
9. Else swap $\mathbf{b}_{j-1}$ and $\mathbf{b}_j$, $j := \max(j-1, 2)$.

- Assume $B \in \mathbb{Z}^{n \times n}$ with $\max \|\mathbf{b}_i\| \leq 2^\beta$.
- Number of loop iterations: $\mathcal{O}(n^2\beta/\log(1/\delta))$.
- Total bit-cost: $\mathcal{O}(n^5\beta^2(n+\beta))$ [Kaltofen'83].

Floating-point LLL

What's wrong with the text-book LLL?

$\Rightarrow$ The rationals involved in the QR computations may be huge: the numerators and denominators may have up to $O(n\beta)$ bits.

Floating-point LLL

What's wrong with the text-book LLL?

$\Rightarrow$ The rationals involved in the QR computations may be huge: the numerators and denominators may have up to $O(n\beta)$ bits.

Floating-point LLL, a hybrid algebraic/numeric approach:

- Perform the QR computations with (low-precision) fp arithmetic, while preserving the general structure of LLL.
- If size-reduction is non-trivial, repeat it (iterative refinement).
- Fp arithmetic concerns QR only: The basis computations are still performed exactly (with integer arithmetic).

Quick history of fp-LLL

- 1982, Odlyzko: coded an fp-LLL, to break knapsack cryptosystems.
- 1988, Schnorr: first provable fp-LLL.
- 1991, Schnorr-Euchner: heuristics for practical fp-LLL.
- Mid 90's: Implemented in NTL by Shoup and in Magma by Steel.
- 2005, Nguyen-S.: L^2, a (much) more efficient provable fp-LLL.
- 2009, Morel-S.-Villard: H-LLL, requiring lower precision.
- 2011, Novocin-S.-Villard: $\widetilde{\mathrm{L}}^1$, with quasi-linear time complexity.

Quick history of fp-LLL

- 1982, Odlyzko: coded an fp-LLL, to break knapsack cryptosystems.
- 1988, Schnorr: first provable fp-LLL.
- 1991, Schnorr-Euchner: heuristics for practical fp-LLL.
- Mid 90's: Implemented in NTL by Shoup and in Magma by Steel.
- 2005, Nguyen-S.: L^2, a (much) more efficient provable fp-LLL.
- 2009, Morel-S.-Villard: H-LLL, requiring lower precision.
- 2011, Novocin-S.-Villard: $\widetilde{L}^1$, with quasi-linear time complexity.

	Kaltofen'82	Schnorr'88	L^2/H-LLL	$\widetilde{L}^1$
complexity	$n^5\beta^2(n+\beta)$	$n^4\beta(n+\beta)^2$	$n^5\beta(n+\beta)$	$n^{5+\varepsilon}\beta^{1+\varepsilon}$
precision	$n\beta$	$n+\beta$	$1.6n/0.8n$	

Why does it work?

Using fp arithmetic does not necessarily imply that the output is incorrect, or that the algorithm is heuristic!

- We keep the input lattice, as bases are manipulated exactly.
- The basis operations are given by the approximate fp QR.
- We can prove that we make progress by using:
  - The axioms of fp arithmetic for $(+, \times, /, \sqrt{})$.
  - Rigorous backward stability of Householder's QR algorithm.
  - Rigorous sensitivity analyses of R under small perturbations.

But still, fp-LLL does not quite compute LLL-reduced bases...

Why does it work?

Using fp arithmetic does not necessarily imply that the output is incorrect, or that the algorithm is heuristic!

- We keep the input lattice, as bases are manipulated exactly.
- The basis operations are given by the approximate fp QR.
- We can prove that we make progress by using:
  - The axioms of fp arithmetic for $(+, \times, /, \sqrt{})$.
  - Rigorous backward stability of Householder's QR algorithm.
  - Rigorous sensitivity analyses of R under small perturbations.

But still, fp-LLL does not quite compute LLL-reduced bases...

Why does it work?

Using fp arithmetic does not necessarily imply that the output is incorrect, or that the algorithm is heuristic!

- We keep the input lattice, as bases are manipulated exactly.
- The basis operations are given by the approximate fp QR.
- We can prove that we make progress by using:
  - The axioms of fp arithmetic for $(+, \times, /, \sqrt{})$.
  - Rigorous backward stability of Householder's QR algorithm.
  - Rigorous sensitivity analyses of R under small perturbations.

But still, fp-LLL does not quite compute LLL-reduced bases...

Why does it work?

Using fp arithmetic does not necessarily imply that the output is incorrect, or that the algorithm is heuristic!

- We keep the input lattice, as bases are manipulated exactly.
- The basis operations are given by the approximate fp QR.
- We can prove that we make progress by using:
  - The axioms of fp arithmetic for $(+, \times, /, \sqrt{})$.
  - Rigorous backward stability of Householder's QR algorithm.
  - Rigorous sensitivity analyses of R under small perturbations.

But still, fp-LLL does not quite compute LLL-reduced bases...

What's wrong with the LLL-reduction?

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j: \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i: \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

What's wrong with the LLL-reduction?

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j : \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i : \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

We can't decide reducedness by looking at the (53) top-most bits:

What's wrong with the LLL-reduction?

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j: \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i: \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

We can't decide reducedness by looking at the (53) top-most bits:

$$\begin{bmatrix} 1 & 2^{60} + 2^5 \\ -1 & 2^{60} \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2^{60} \\ -1 & 2^{60} \end{bmatrix}$$

Not reduced Reduced

$$\begin{bmatrix} 1 & 2^{53} + 2^{-1} + 2^{-25} \\ 2^{-10} & -2^{63} \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2^{53} + 1 \\ 2^{-10} & -2^{63} \end{bmatrix}$$

Reduced Not reduced

What's wrong with the LLL-reduction?

Let $\delta \in (1/4, 1)$. A basis $B = (\mathbf{b}_i)_{i \leq n} \in \mathbb{R}^{n \times n}$ with QR-factorisation $B = QR$ is said LLL-reduced if:

- $\forall i, j: \ |r_{i,j}| \leq r_{i,i}/2$ [size-reduction]
- $\forall i: \ \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ [Lovász' condition].

We can't decide reducedness by looking at the (53) top-most bits:

$$\begin{bmatrix} 1 & 2^{60} + 2^5 \\ -1 & 2^{60} \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2^{60} \\ -1 & 2^{60} \end{bmatrix}$$

Not reduced ⟹ Reduced

$$\begin{bmatrix} 1 & 2^{53} + 2^{-1} + 2^{-25} \\ 2^{-10} & -2^{63} \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 2^{53} + 1 \\ 2^{-10} & -2^{63} \end{bmatrix}$$

Reduced ⟹ Not reduced

Sensitivity of the R-factor

- Take $B \in \mathbb{R}^{n \times n}$ non-singular, with $B = QR$.
- Apply a columnwise perturbation ΔB, i.e., $\max_i \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq \varepsilon$.
- That's the perturbation provided by the backward stability analysis of Householder's algorithm, for $\varepsilon \approx 2^{-p}$.
- If ε is very small, then $B + \Delta B$ is non-singular and:

$$B + \Delta B = (Q + \Delta Q)(R + \Delta R).$$

- How large can ΔR be?

Sensitivity of the R-factor

- Take $B \in \mathbb{R}^{n \times n}$ non-singular, with $B = QR$.
- Apply a columnwise perturbation ΔB, i.e., $\max_i \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq \varepsilon$.
- That's the perturbation provided by the backward stability analysis of Householder's algorithm, for $\varepsilon \approx 2^{-p}$.
- If ε is very small, then $B + \Delta B$ is non-singular and:

$$B + \Delta B = (Q + \Delta Q)(R + \Delta R).$$

- How large can ΔR be?

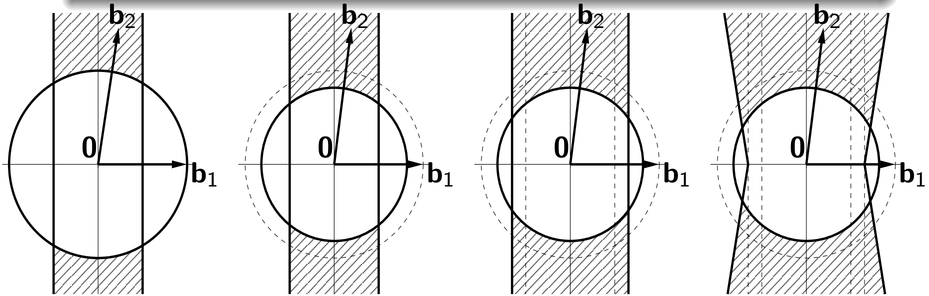
Let $\mathrm{cond}(R) = \||R||R^{-1}|\|$. If $\mathrm{cond}(R) \cdot \varepsilon \lesssim 1$, then:
$B + \Delta B$ is non-singular and $\max \frac{\|\Delta \mathbf{r}_i\|}{\|\mathbf{r}_i\|} \lesssim \mathrm{cond}(R) \cdot \varepsilon$.

Furthermore, if B is LLL-reduced, then $\mathrm{cond}(R) = 2^{\mathcal{O}(n)}$.

Fixing the LLL-reduction

Let $\Xi = (\delta, \eta, \theta)$ with $\eta \in (1/2, 1)$, $\theta > 0$ and $\delta \in (\eta^2, 1)$.
A basis $B \in \mathbb{R}^{n \times n}$ with R-factor R is said Ξ-reduced if:

- $\forall i, j : \; |r_{i,j}| \leq \eta \cdot r_{i,i} + \theta \cdot r_{j,j}$ [Modified size-reduction]
- $\forall i : \; \delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$.



$(1, 1/2, 0)$ $(\delta, 1/2, 0)$ $(\delta, \eta, 0)$ (δ, η, θ)

1. Reminders on Euclidean lattices.
2. Using floating-point arithmetic within lattice algorithms.
3. **The fplll library.**

What is fplll?

`http://perso.ens-lyon.fr/xavier.pujol/fplll/`

- A C++ library, under Lesser GPL v2.1.
- Created in 2005 (current version: 3.1).
- Former developers: Cadé, S.. Current developer: Pujol.
- Fairly compact: $\approx 10,000$ lines.
- Used by SAGE, MAGMA, Pari GP & Mathemagix.
- Main competitor: Shoup's NTL.

What is fplll?

`http://perso.ens-lyon.fr/xavier.pujol/fplll/`

- A C++ library, under Lesser GPL v2.1.
- Created in 2005 (current version: 3.1).
- Former developers: Cadé, S.. Current developer: Pujol.
- Fairly compact: $\approx 10,000$ lines.
- Used by SAGE, MAGMA, Pari GP & Mathemagix.
- Main competitor: Shoup's NTL.

What is fplll?

`http://perso.ens-lyon.fr/xavier.pujol/fplll/`

- A C++ library, under Lesser GPL v2.1.
- Created in 2005 (current version: 3.1).
- Former developers: Cadé, S.. Current developer: Pujol.
- Fairly compact: $\approx 10,000$ lines.
- Used by SAGE, MAGMA, Pari GP & Mathemagix.
- Main competitor: Shoup's NTL.

What is fplll?

`http://perso.ens-lyon.fr/xavier.pujol/fplll/`

- A C++ library, under Lesser GPL v2.1.
- Created in 2005 (current version: 3.1).
- Former developers: Cadé, S.. Current developer: Pujol.
- Fairly compact: $\approx 10,000$ lines.
- Used by SAGE, MAGMA, Pari GP & Mathemagix.
- Main competitor: Shoup's NTL.

Goal: show that our theoretical algorithms are relevant in practice.

What does it do?

- Contains efficient and guaranteed implementations of lattice algorithms, (most) often relying on fp arithmetic:
  1. LLL reduction [Nguyen-S.'05].
  2. HKZ reduction, SVP & CVP solvers [Pujol-S.'08].
  3. And soon, BKZ reduction.
- Contains heuristic variants as well.
- Contains an automatic wrapper that:
  1. Tries the fastest variants first.
  2. Detects when things go wrong.
  3. Eventually switches to more rigorous variants.

What does it do?

- Contains efficient and guaranteed implementations of lattice algorithms, (most) often relying on fp arithmetic:
  1. LLL reduction [Nguyen-S.'05].
  2. HKZ reduction, SVP & CVP solvers [Pujol-S.'08].
  3. And soon, BKZ reduction.
- Contains heuristic variants as well.
- Contains an automatic wrapper that:
  1. Tries the fastest variants first.
  2. Detects when things go wrong.
  3. Eventually switches to more rigorous variants.

What does it do?

- Contains efficient and guaranteed implementations of lattice algorithms, (most) often relying on fp arithmetic:
    1. LLL reduction [Nguyen-S.'05].
    2. HKZ reduction, SVP & CVP solvers [Pujol-S.'08].
    3. And soon, BKZ reduction.
- Contains heuristic variants as well.
- Contains an automatic wrapper that:
    1. Tries the fastest variants first.
    2. Detects when things go wrong.
    3. Eventually switches to more rigorous variants.

What does it use?

Integer arithmetic:

- `long ints` arithmetic is input basis entries are small.
- GNU MP's `mpz`'s.

Floating-point arithmetic:

- `doubles`,
- `DPEs`: exponent stored externally on an `int`,
- External exponent shared for a whole vector,
- MPFR.

GSO/QR numerical algorithm:

- Cholesky's algorithm, starting from approximate/exact $B^T B$.
- Sub-optimal choice for numerical stability...
- but relatively low number of arithmetic operations.

What does it use?

Integer arithmetic:

- `long ints` arithmetic is input basis entries are small.
- GNU MP's `mpz`'s.

Floating-point arithmetic:

- `doubles`,
- `DPEs`: exponent stored externally on an `int`,
- External exponent shared for a whole vector,
- MPFR.

GSO/QR numerical algorithm:

- Cholesky's algorithm, starting from approximate/exact $B^T B$.
- Sub-optimal choice for numerical stability...
- but relatively low number of arithmetic operations.

What does it use?

Integer arithmetic:

- `long ints` arithmetic is input basis entries are small.
- GNU MP's `mpz`'s.

Floating-point arithmetic:

- `doubles`,
- `DPEs`: exponent stored externally on an `int`,
- External exponent shared for a whole vector,
- MPFR.

GSO/QR numerical algorithm:

- Cholesky's algorithm, starting from approximate/exact $B^T B$.
- Sub-optimal choice for numerical stability. . .
- but relatively low number of arithmetic operations.

Is the rational LLL really that bad?

After all, the complexity bounds do not differ that much:

$$n^5 \beta^2 (n+\beta) \quad \text{versus} \quad n^5 \beta (n+\beta).$$

Is the rational LLL really that bad?

After all, the complexity bounds do not differ that much:

$$n^5\beta^2(n+\beta) \quad \text{versus} \quad n^5\beta(n+\beta).$$

Using MAGMA V2.16:

```
> n:=25; beta:=2000;
> B:=RMatrixSpace(Integers(),n,n)!0;
> for i:=1 to n do
>    B[i][i]:=1;
>    B[i][1]:=RandomBits(beta);
> end for;
> time _:=LLL(B:Method:=''Integral'');
Time: 11.700
> time _:=LLL(B);
Time: 0.240
```

Correctness and termination

After all, we can check that $\frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}$ is small. But:

- The execution may loop forever.
- It may be hard to detect for the user.
- Correctness and termination tend to be intertwinned.
- We found a basis with $n = 55$ and $\beta \approx 100$ that makes NTL's LLL_FP loop forever.

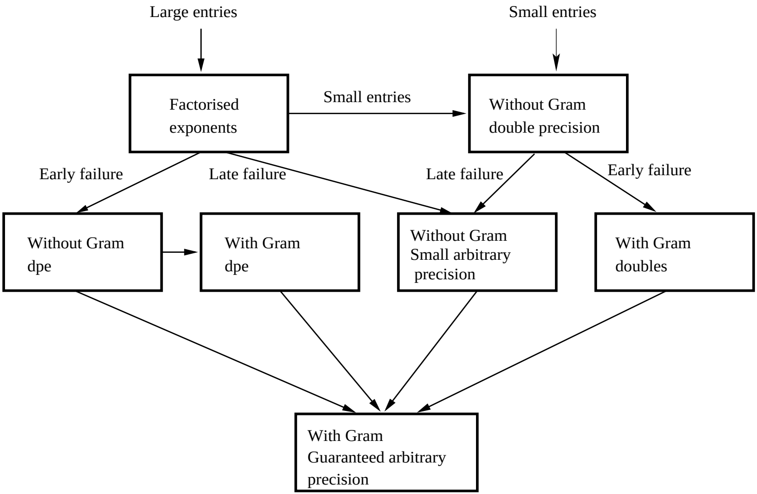
Correctness and termination

After all, we can check that $\frac{\|\mathbf{b}_1\|}{(\det L)^{1/n}}$ is small. But:

- The execution may loop forever.
- It may be hard to detect for the user.
- Correctness and termination tend to be intertwinned.
- We found a 55-dimensional lattice with $\beta \approx 100$ that makes NTL's LLL_FP loop forever.

```
[...]
unexpected behaviour -> exit
=== LLL method end : Size-reduction failed. (kappa=54) ===
=== LLL method : proved<mpz_t, double> ===
Setting precision at 53 bits.
Entering fpLLL:
[...]
====== LLL method end : success ======
```

A hierarchy of variants (slightly outdated)



Current limitations

- The bottleneck used to stem from β.
- Large dimensions ($\gtrsim 150$) were seldom encountered.

- Now it's quite fast up to $n \approx 165$: that's when double precision starts not being sufficient for "generic" bases.
- Then it switches to MPFR, which makes it extremely slow.

- We have ways to push this limit: $n \approx 330$ using H-LLL, maybe $n \approx 1,000$ using new developments.
- Then the complexity with respect to n starts to kick in.

Current limitations

- The bottleneck used to stem from β.
- Large dimensions ($\gtrsim 150$) were seldom encountered.

- Now it's quite fast up to $n \approx 165$: that's when double precision starts not being sufficient for "generic" bases.
- Then it switches to MPFR, which makes it extremely slow.

- We have ways to push this limit: $n \approx 330$ using H-LLL, maybe $n \approx 1,000$ using new developments.
- Then the complexity with respect to n starts to kick in.

Current limitations

- The bottleneck used to stem from β.
- Large dimensions ($\gtrsim$ 150) were seldom encountered.

- Now it's quite fast up to $n \approx 165$: that's when double precision starts not being sufficient for "generic" bases.
- Then it switches to MPFR, which makes it extremely slow.

- We have ways to push this limit: $n \approx 330$ using H-LLL, maybe $n \approx 1,000$ using new developments.
- Then the complexity with respect to n starts to kick in.

Conclusion

- A rigorous use of fp arithmetic for an algebraic computation.
- Why using a hybrid approach?
  Because we can, and it gives the best complexity bounds.
- Rigorous implementation based on a wrapper that automatically chooses fast/rigorous variants.
- fplll is very often the fastest, and the only one providing correctness and termination guarantees.

Projects

Theoretical projects:

- Combine the algorithmic improvements wrt β with those wrt n [Schönhage'84, Koy-Schnorr'01].
- Beat the $\mathcal{O}(n)$ fp precision barrier.
- Get faster algorithms, possibly with bit-complexity

$$\mathcal{O}(n^{\omega+\varepsilon}\beta^{1+\varepsilon}), \text{ with } \omega = 2.376\ldots$$

And keep up with the algorithmic improvements!!!

- H-LLL [Morel-S-Villard'09] is still not implemented.
- BKZ is just being implemented.
- [Novocin-S-Villard'11] needs cleaning before implementation.

Projects

Theoretical projects:

- Combine the algorithmic improvements wrt β with those wrt n [Schönhage'84, Koy-Schnorr'01].
- Beat the $\mathcal{O}(n)$ fp precision barrier.
- Get faster algorithms, possibly with bit-complexity

$$\mathcal{O}(n^{\omega+\varepsilon}\beta^{1+\varepsilon}), \text{ with } \omega = 2.376\ldots$$

And keep up with the algorithmic improvements!!!

- H-LLL [Morel-S-Villard'09] is still not implemented.
- BKZ is just being implemented.
- [Novocin-S-Villard'11] needs cleaning before implementation.