M2 - RÉSEAUX EN CRYPTOGRAPHIE

ALEXANDRE WALLET - QUYEN NGUYEN

Important : `sage` travaille naturellement avec des vecteurs lignes, et on représentera donc un réseau par une matrice "ligne" B. Ne pas hésiter à regarder `https://doc.sagemath.org/html/en/tutorial/tour_linalg.html` et de manière générale, la documentation en ligne de `sage`.

Exercice 1 (Echauffement)**.** Implémenter l'algorithme de Gauss-Lagrange.

Algorithm 1: Algorithme de Gauss-Lagrange

---

input : Une base ordonnée $(\mathbf{u}, \mathbf{v})$ avec $\|\mathbf{u}\| \leq \|\mathbf{v}\|$
output: Une base Gauss-Lagrange réduite du réseau (voir aussi TD1)
repeat
  $x = \lfloor \langle \mathbf{u}, \mathbf{v} \rangle / \|\mathbf{u}\|^2 \rceil$
  $\mathbf{t} = \mathbf{v} - x\mathbf{u}$
  $\mathbf{v} = \mathbf{u}$
  $\mathbf{u} = \mathbf{r}$
until $\|\mathbf{u}\| \geq \|\mathbf{v}\|$;
return $(\mathbf{v}, \mathbf{u})$

Exercice 2. Le but est d'arriver à une implémentation *simple* de LLL, tel que décrit en cours. Pour commencer on s'autorisera à "tout recalculer" même si c'est sous-optimal. On rappelle qu'en "ligne", le procédé de Gram-Schmidt s'écrit $\mathbf{B} = \mathbf{U}\widetilde{\mathbf{B}}$, avec $\mathbf{U}$ triangulaire inférieure, et peut se calculer par `B.gram_schmidt()`.

(1) Implémenter une fonction `SizeReduction` comme décrit en cours. Tester sur des exemples que votre fonction est correcte.
(2) Implémenter l'algorithme LLL à l'aide de `SizeReduction`. Tester votre algorithme en vérifiant que la sortie est bien réduite au sens du cours.
(3) Comparer votre algorithme à la méthode `.LLL()` de `sage` (faire varier la taille des entrées et le rang des réseaux).
(4) Etudier le nombre d'échanges sur plusieurs exemples quand n augmente, et le comportement de l'algorithme lors d'un échange (il sera judicieux d'afficher les indices des vecteurs échangés).
(5) (Pour aller plus loin) Programmer une manière de visualier l'évolution des normes des vecteurs de Gram-Schmidt pendant l'exécution de votre LLL.
(6) (Toujours plus loin) Tester LLL et BKZ sur des grandes instances. "Grandes" peut signifier grande dimension, mais aussi grands nombres. [1]

---

1. Par exemple, `sage` contient un générateur de "réseaux cryptographiques [2]", qu'on peut récupérer en tapant `from sage.crypto import gen_lattice`, puis en appelant la fonction `gen_lattice()`. Ces réseaux contiennent par construction des vecteurs extrêmement courts.

Exercice 3. On utilise ici LLL pour trouver des relations entre réels $x_1, \dots, x_d$. On cherche $n_1, \dots, n_d$ des entiers tels que $|\sum_{i=1}^d n_i x_i|$ est petite avec LLL, selon une approche de Kannan-Lenstra-Lovàsz. On considère le réseau engendré par les lignes $\mathbf{b}_i$ de la matrice suivante

$$\mathbf{B} = \begin{bmatrix} \lfloor Cx_1 \rceil & 1 & 0 & \dots & 0 \\ \lfloor Cx_2 \rceil & 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ \lfloor Cx_d \rceil & 0 & \dots & 0 & 1 \end{bmatrix} \in \mathcal{M}_{d,d+1}(\mathbb{Z})$$

Le premier vecteur de la base renvoyée par LLL est un vecteur court de la forme

$$n_1\mathbf{b}_1 + n_2\mathbf{b}_2 + \dots + n_d\mathbf{b}_d = (\sum_{i=1}^{d} n_i \lfloor Cx_i \rceil, n_1, \dots, n_d).$$

Lorsque C est assez grand, cela implique que $|\sum n_i x_i|$ est petite, et les n_i ont une bonne chance de décrire une relation entre les x_i. Utiliser cette méthode pout trouver le polynôme minimal de $\sqrt{2} + \sqrt{3} + \sqrt[3]{3}$. On travaillera avec des réels `RealField(400)` et $C = 10^{100}$. Pour vérifier le résultat, on pourra comparer avec la méthode `.minpoly()` de `sage`.

Exercice 4. L'implémentation de LLL dans l'exercice 2 n'est clairement pas optimisée : il n'y pas besoin de recalculer la totalité des Gram-Schmidts en général après un échange, ni même la totalité des μ_{ij}. D'autre part, il n'y a pas non plus besoin de size-réduire toute la base.

(1) Modifier la fonction `SizeReduction` (et éventuellement votre LLL) pour ne réduire que les μ_{ij} nécessaires, lorsqu'il le faut. Comparer le temps d'exécution.

(2) On peut décortiquer la phase de vérification de la condition de Lovàsz en deux fonctions : un `Swap` qui échange deux vecteurs de la base $\mathbf{B}$, et une fonction `Update` qui met à jour uniquement les Gram-Schmidts et les μ_{ij} nécessaires. Implémenter ces deux fonctions au mieux, et comparer le temps d'exécution de ce nouvel algorithme avec l'ancien.

Exercice 5. Des suggestions pour aller encore plus loin, pour les intéressés :

— Implémenter un algorithme pour mettre une matrice sous forme de Hermite.

— Implémenter des algorithmes pour résoudres les problèmes "faciles" vus en cours (en `sage`, on peut mettre $\mathbf{B}$ en forme de Hermite avec $\mathbf{B}$`.hermite_form()`).

— S'esssayer à implémenter des algorithmes d'énumération et de crible.