Réseaux euclidiens

Ce cours est largement inspiré de divers textes de Phong Q. NGuyen, notamment[10, 11]. Le livre (librement accessible en ligne de) Steven Galbraith [4] est une bonne référence aussi, tout comme le livre d'Antoine Joux [6] (qu'on peut trouver aussi en ligne... mais pas librement !).

On note les vecteurs en **gras** et on les écrit en ligne (souvent, en France, ils sont écrits en colonne ; ce document utilise plutôt la convention anglo-saxonne). Les coordonnées individuelles ne sont pas en gras (ce sont juste des nombres). Donc on a $\mathbf{x} = (x_1, \ldots, x_n)$. OK ?

0.1 Introduction

La *norme euclidienne* d'un vecteur (sa « longueur ») est notée :

$$\|\mathbf{x}\| = \sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}$$

Considérons des vecteurs $\mathbf{b}_1, \ldots, \mathbf{b}_m$ de $\mathbb{R}^n$. À ce stade, tout le monde sait ce qu'est un espace vectoriel engendré par $(\mathbf{b}_1, \ldots, \mathbf{b}_m)$: c'est l'ensemble des combinaisons linéaires des $\mathbf{b}_i$, autrement dit c'est l'ensemble

$$\langle \mathbf{b}_1, \ldots, \mathbf{b}_n \rangle = \left\{ \sum_{i=1}^{m} \lambda_i \mathbf{b}_i : (\lambda_1, \ldots, \lambda_m) \in \mathbb{R}^m \right\}$$

On sait que si deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ appartiennent à un espace vectoriel, alors leur somme $\mathbf{x} + \mathbf{y}$ y appartient aussi, de même que $\lambda\mathbf{x}$, pour tout $\lambda \in \mathbb{R}$. Lorsque les $\mathbf{b}_i$ sont linéairement indépendants, on dit qu'ils forment une *base* de l'espace vectoriel en question (et on a $m = n$).

Un *réseau euclidien* (« Euclidean lattice »), c'est presque la même chose, mais avec des entiers. Il suffit de remplacer $\mathbb{R}$ par $\mathbb{Z}$ dans le paragraphe précédent ! Mais bien sûr, ça change tout.

Définition 1 : Réseau euclidien

> Si $\mathbf{b}_1, \ldots, \mathbf{b}_d$ sont des vecteurs de $\mathbb{Z}^n$ alors ils **engendrent** le **réseau euclidien** $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ —qu'on va abréger $\mathcal{L}$— qui est formé de toutes les combinaisons linéaires à coefficients entiers des $\mathbf{b}_i$:
>
> $$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \left\{ \sum_{i=1}^{d} \mu_i \mathbf{b}_i : \mu_1, \ldots, \mu_d \in \mathbb{Z} \right\}.$$
>
> Si en plus les $\mathbf{b}_i$ sont linéairement indépendants (ce qui sera généralement le cas), on dit qu'ils forment une **base** de $\mathcal{L}$. Dans ce cas, d est la **dimension** du réseau.
>
> Dans le cas particulier important où $d = n$ (la dimension d du réseau est égale à la dimension n de l'espace ambiant $\mathbb{R}^n$), on dit que le réseau est **de rang plein**.

Par exemple, la figure 1 montre le réseau de dimension 2 engendré par $\mathbf{b_1} = (1, 5)$ et $\mathbf{b_2} = (2, 2)$. La différence avec les espaces vectoriels saute aux yeux : un réseau est un ensemble *discret* (par opposition à « continu », c.a.d. qu'il y a « du vide » entre les points du réseau). On voit que les points du réseau sont disposés régulièrement dans l'espace. La figure 2 montre un réseau en trois dimensions.

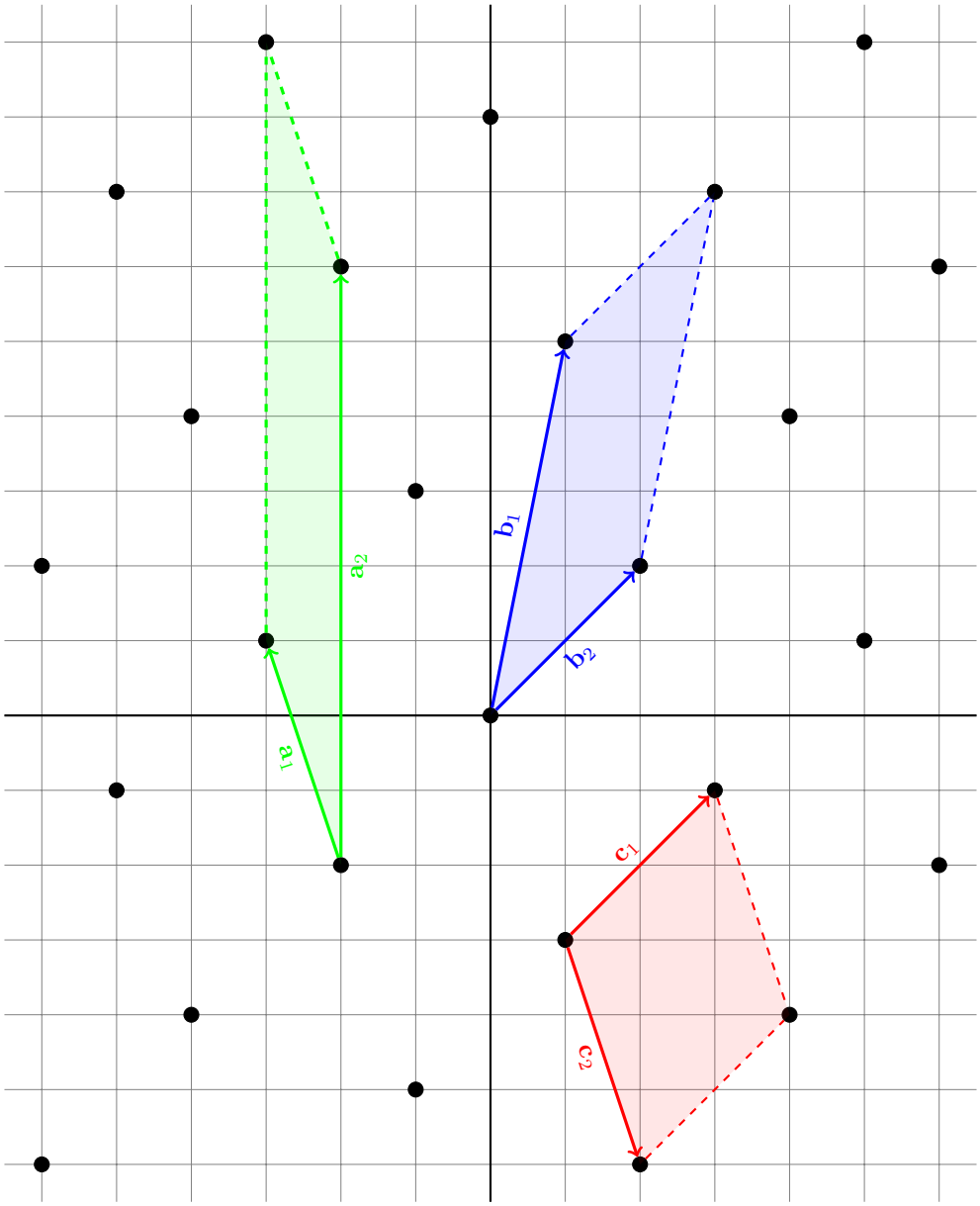

FIGURE 1 – Un réseau euclidien en deux dimensions (ce sont les points noirs), avec plusieurs bases possibles et les parallélepipèdes fondamentaux qui correspondent.

0.1.1 Bases d'un réseau

Comme une base est formée d'une collection de vecteurs, il est commode de les représenter par des matrices. Ainsi, la base $(\mathbf{b}_1, \mathbf{b}_2)$ ci-dessus est représentée par (les lignes de) la matrice :

$$B = \begin{pmatrix} 1 & 5 \\ 2 & 2 \end{pmatrix}$$

Tous les points du réseau s'écrivent donc $\mathbf{x}B$, où $\mathbf{x}$ est un vecteur à coordonnées entières.

Le même réseau peut être engendré par plusieurs bases. En effet, celui de la figure 1 aurait aussi été engendré par

$$A = \begin{pmatrix} -1 & 3 \\ 0 & 8 \end{pmatrix} \quad \text{et} \quad C = \begin{pmatrix} 2 & 2 \\ 1 & -3 \end{pmatrix}$$

On passe d'une matrice de base à une autre en multipliant (à gauche) par une matrice de « changement de base » à coefficients entiers et dont l'inverse est aussi à coefficients entiers (une telle matrice est dite *unimodulaire* ; il faut et il suffit que son determinant soit ± 1).

On démontre que si $A = UB$, où U est une matrice carrée à coefficients entiers et de déterminant ± 1, alors les lignes de A engendrent le même réseau que les lignes de B. Notons $\mathcal{L}(A)$ le réseau engendré par les lignes de A ;

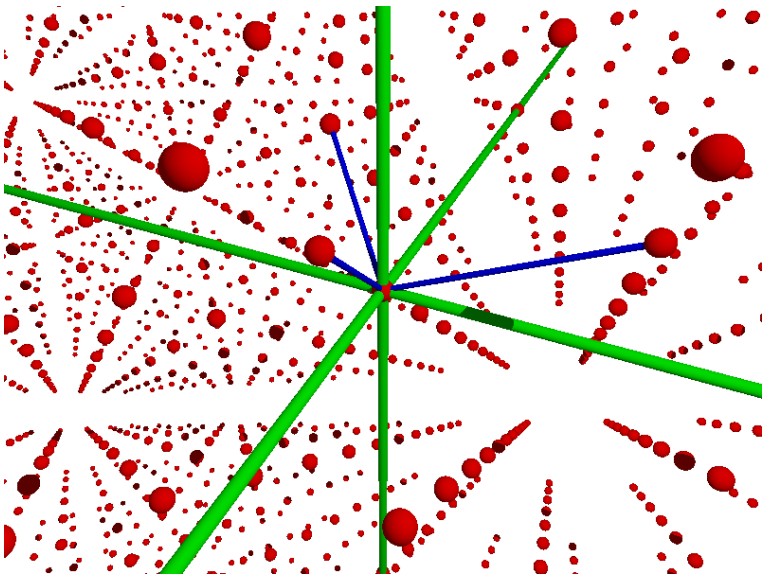

FIGURE 2 – Un réseau euclidien en trois dimensions. Les axes sont en vert et les vecteurs de base en bleu.

un point de ce réseau s'écrit $\mathbf{x}A = (\mathbf{x}U)B$, donc c'est en particulier une combinaison linéaire entière des lignes de B. On a donc $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. Ensuite, un point de $\mathcal{L}(B)$ s'écrit $\mathbf{x}B = (\mathbf{x}U^{-1})A$, donc c'est une combinaison linéaire entière des lignes de A (en effet U^{-1} est à coefficients entiers). Ceci prouve que $\mathcal{L}(B) \subseteq \mathcal{L}(A)$. On a donc égalité entre les deux.

Maintenant, montrons que si $\mathcal{L}(A) = \mathcal{L}(B)$, alors il existe une matrice unimodulaire U telle que $A = UB$. Par définition, les lignes de A appartiennent toutes à $\mathcal{L}(B)$, donc il existe une matrice U à coefficients entiers telle que $A = UB$. De manière symétrique, il existe V à coefficients entiers telle que $B = VA$. On a donc $A = UVA$, autrement dit $(UV - I)A = 0$. Les lignes de A sont linéairement indépendantes sur $\mathbb{R}$, donc le noyau à gauche de A est le sous-espace vectoriel trivial qui ne contient que $\{0\}$. Par conséquent, on a $UV = I$. Ceci prouve que U est inversible sur $\mathbb{Z}$, car V, son inverse, est à coefficients entiers.

Étant donné une base B d'un réseau, il est facile de déterminer si un vecteur $\mathbf{y}$ appartient au réseau ou pas : il suffit de tester si le système linéaire $\mathbf{x}B = \mathbf{y}$ possède une solution $\mathbf{x}$ à coefficients entiers.

0.1.2 Volume

Si on fixe une base $B = (\mathbf{b_1}, \ldots, \mathbf{b}_d)$, on peut former le *parallélepipède fondamental* qui correspond (ils sont représentés sur la figure 1) c'est-à-dire l'ensemble

$$\left\{ \sum_{i=1}^{d} x_i \mathbf{b}_i : x_1, \ldots, x_d \in [0; 1[\right\}.$$

Un tel parallélépipède réalise un pavage de $\mathbb{R}^n$. Des bases différentes en donnent de formes différentes, mais il se trouve qu'ils ont tous le même *volume*[1].

Définition 2 : Volume

Soit une matrice B. Le **volume** du parallélépipède fondamental engendré par les lignes de B est $\mathrm{Vol}(B) = \sqrt{\det BB^t}$.

Le coefficient d'indice (i, j) de la « matrice de Gram » BB^t est le produit scalaire $\mathbf{b}_i \cdot \mathbf{b}_j$. En conséquence, si la base est orthogonale, tous ces produits scalaires sont nuls saufs ceux sur la diagonale, et le déterminant forme leur produit. Le volume est alors le produit des longueur des $\mathbf{b}_i$, donc c'est bien le volume (au sens usuel) du parallélepipède qu'ils forment. Et en fait, c'est bien le cas même s'ils ne sont pas orthogonaux.

Dans le cas particulier (important) où B est une matrice carrée, alors $\mathrm{Vol}(B) = |\det B|$. En effet, dans ce cas $\det BB^t = (\det B)(\det B^t) = \det B^2$, donc $\mathrm{Vol}(B) = \sqrt{(\det B)^2} = |\det B|$.

Si deux bases engendrent le même réseau, alors leur parallélépipèdes fondamentaux ont le même volume.

1. En deux dimensions, le « volume » est simplement la surface (en mètres-carrés). En trois dimension, c'est le volume normal (en mètres-cubes). En $d > 3$ dimension... c'est « l'hypervolume » en m^d.

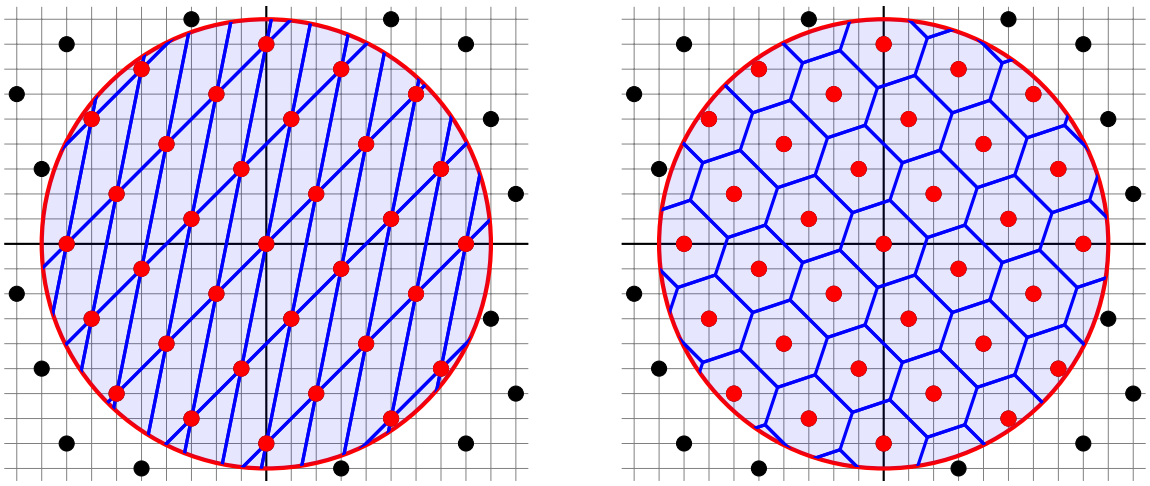

FIGURE 3 – Illustration de l'heuristique Gaussienne. À gauche : le parallélepipède fondamental associé à la base B. À droite : le diagramme de Voronoï du réseau. Dans les deux cas, le nombre de points du réseau dans le cercle est à peu près le nombre de copies de la forme répétée contenues dans le cercle, donc à peu près le volume du cercle divisé par le volume du réseau.

Il existe une matrice unimodulaire U telle que $A = UB$, et

$$\det AA^t = \det UBB^tU^t = (\det U)(\det BB^t)(\det U^t) = \det BB^t,$$

car $\det U = \det U^t = \pm 1$. Donc les deux volumes sont les mêmes.

Par conséquent, le volume ne dépend pas du choix de la base, mais c'est une caractéristique géométrique intrinsèque du réseau lui-même. On parle donc du *volume* du réseau $\mathcal{L}$ et on le note $\mathrm{Vol}(\mathcal{L})$, pour parler du volume du parallélépipède fondamental associé à une base du réseau.

En fait, c'est même un peu plus fort que ça : une partie D de $\mathbb{R}^n$ (mesurable), telle que les ensembles $\mathbf{b} + D$ (pour $\mathbf{b} \in \mathcal{L}$) recouvrent $\mathbb{R}^n$ et sont d'intérieurs disjoints, s'appelle un domaine fondamental du réseau $\mathcal{L}$. Tous les domaines fondamentaux de $\mathcal{L}$ ont le même volume. Par exemple, les régions du diagramme de Voronoï associées aux points du réseau font aussi l'affaire. Cf. figure 3.

Calculer le volume est généralement facile. Le volume du réseau de la figure 1 est donc 8. En général, on a souvent affaire à des matrices B qui sont triangulaires, donc le déterminant est très facile à calculer : c'est le produit des entrées sur la diagonale.

Cette notion de volume a plusieurs intérêts. D'abord, plus le volume est faible, plus les points du réseau sont serrés les uns aux autres : la « densité » du réseau est grosso-modo l'inverse du volume. Chaque point du réseau « occupe » un volume $\mathrm{Vol}(\mathcal{L})$.

Ceci permet de faire le raisonnement suivant, qui se nomme *l'heuristique gaussienne*. Prenons une boule[2] de rayon r ; combien de point du réseau contient-elle ? On peut estimer ceci en disant :

$$\#\text{ points dans la boule} \approx \frac{\text{Volume de la boule}}{\text{Volume du réseau}}.$$

Ceci est une approximation, car à la frontière de la boule il y a des parallélépipèdes fondamentaux tronqués (cf. figure 3).

0.1.3 Défaut d'orthogonalité d'une base

Toutes les bases d'un réseau ne sont pas aussi intéressantes les unes que les autres. Certaines ont des vecteurs plus longs que d'autres ; dans certaines, les vecteurs sont plus orthogonaux que dans d'autres (rappel : deux vecteurs sont orthogonaux s'ils sont « perpendiculaires », c.a.d. si leur produit scalaire est nul).

En général, un réseau n'admet pas de base orthogonale (c.a.d. dont les vecteurs sont deux-à-deux orthogonaux). C'est par exemple le cas de celui de la figure 1. À défaut, une base « réduite », qui sont formée par des vecteurs relativement courts et quasi-orthogonaux, est intéressante (on verra après pourquoi).

2. c.a.d. l'ensemble des points qui sont à distance $\leq r$ du centre, quel que soit le nombre de dimensions...

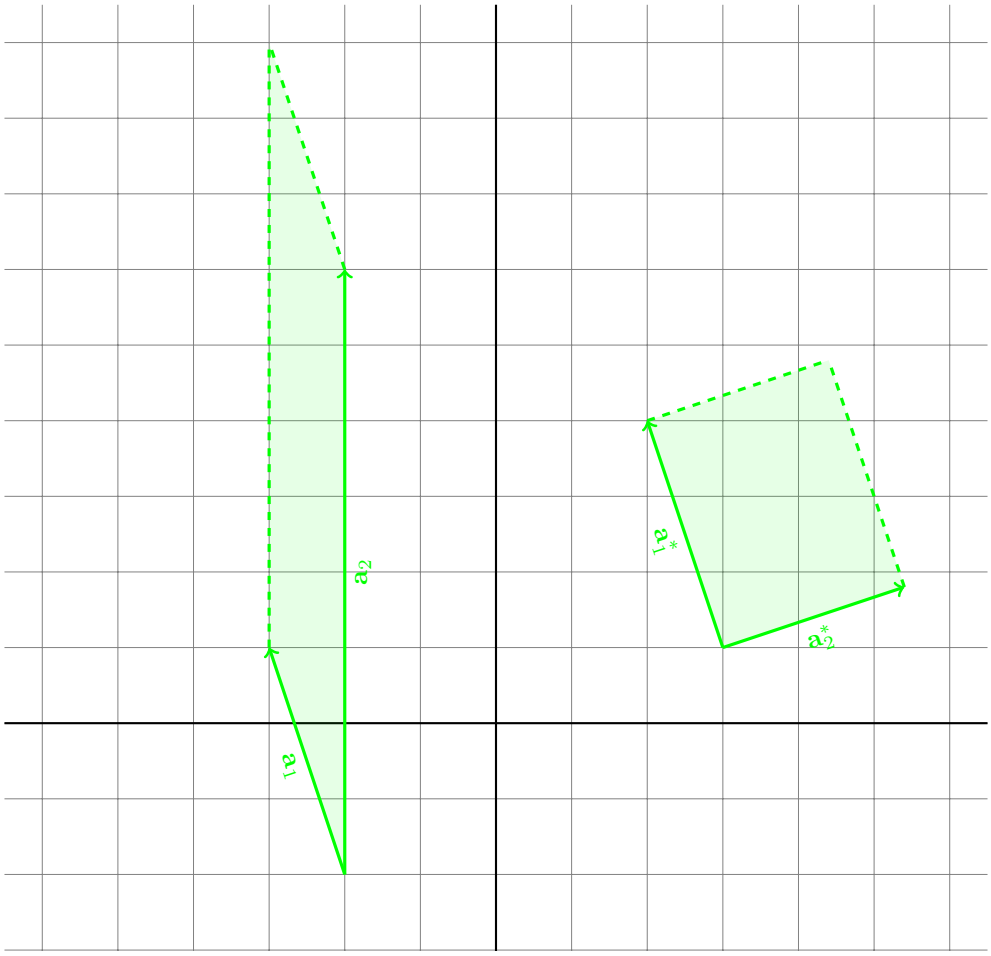

FIGURE 4 – Illustration du défaut d'orthogonalité et de l'orthogonalisation d'une base. Les deux zones vertes ont le même volume (8) et $\mathbf{a}_2^*$ est beaucoup plus court que $\mathbf{a}_2$.

Considérons une base $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$ d'un réseau $\mathcal{L}$. Si les vecteurs de la base étaient orthogonaux, alors le volume du parallélépipède fondamental qui leur est associé serait le produit de la longueur des côtés :

$$\mathrm{Vol}(\mathcal{L}) = \|\mathbf{b}_1\| \times \cdots \times \|\mathbf{b}_d\|.$$

Mais en général, quand une base n'est pas orthogonale, le produit de la longueur de ses vecteurs est *plus grand* que le volume du réseau. Pour le réseau de la figure 1, dont le volume est 8, on trouve :

$$\begin{array}{rcrcl} \|\mathbf{a}_1\| \times \|\mathbf{a}_2\| & = & 8\sqrt{10} & \approx & 3.16\,\mathrm{Vol}(\mathcal{L}) \\ \|\mathbf{b}_1\| \times \|\mathbf{b}_2\| & = & 4\sqrt{13} & \approx & 1.8\,\mathrm{Vol}(\mathcal{L}) \\ \|\mathbf{c}_1\| \times \|\mathbf{c}_2\| & = & 4\sqrt{(5)} & \approx & 1.2\,\mathrm{Vol}(\mathcal{L}) \end{array}$$

On voit que moins les vecteurs de la base ont l'air orthogonaux, plus l'écart entre le produit de leur longueur et le volume est important. La figure 4 l'illustre d'une autre manière : étant donné une base B, on peut « l'orthogonaliser » (par exemple avec le procédé de Gram-Schmidt, ou bien en calculant une factorisation QR) et obtenir une base B^* qui engendre le même espace vectoriel. On voit que les vecteurs de B^* sont plus courts.

Ceci permet de définir le *défaut d'orthogonalité* d'une base $B = (\mathbf{b}_1, \ldots, \mathbf{b}_d)$: c'est le nombre $\|\mathbf{b}_1\| \times \cdots \times \|\mathbf{b}_d\| / \mathrm{Vol}(\mathcal{L})$. Si B est une base orthogonale, alors ce nombre vaut 1. Plus il est grand, moins la base est orthogonale.

Le mathématicien Charles Hermite (1822–1901), pionnier de l'étude des réseaux euclidiens, a démontré que tout réseau (de rang plein) admet des bases dont le défaut d'orthogonalité est plus petit que $(4/3)^{n(n-1)/4}$ (ce qui est fort, c'est que ça ne dépend que de la dimension, pas de la taille des coefficients dans n'importe quelle matrice de base). Mais en général, on ne sait pas comment les calculer en temps polynomial.

0.1.4 Plus courts vecteurs

Dans un réseau euclidien, on peut parler du *plus court vecteur* (non-nul), alors que ça n'aurait pas de sens dans un espace vectoriel. Dans la figure 1, on voit bien que c'est $\mathbf{b_2} = (2, 2)$.

La longueur du plus court vecteur d'un réseau $\mathcal{L}$ en est une caractéristique très importante. On l'appelle le *premier minimum* et on la note $\lambda_1(\mathcal{L})$.

On verra plus tard que ce n'est pas facile de trouver explicitement un plus court vecteur dans un réseau quelconque. Cependant, on peut dire des choses sur sa taille. Une première approche consiste à utiliser l'heuristique Gaussienne : on peut estimer le nombre de points du réseau dans une boule de rayon r ; on fait baisser r jusqu'à ce que ce nombre vaille 1 ; alors r est une estimation de la taille du plus court vecteur du réseau. Dit autrement : on s'attend à ce qu'une boule de rayon $\lambda_1(\mathcal{L})$ ait sensiblement le même volume que le réseau.

Notons ν_n le volume de la boule unité (c.a.d. de rayon 1) en n dimensions. Le volume d'une boule de rayon r est $r^n\nu_n$. Le raisonnement ci-dessus dit qu'on devrait avoir $\mathrm{Vol}(\mathcal{L}) \approx \nu_n \lambda_1(\mathcal{L})^n$, autrement dit

$$\lambda_1(\mathcal{L}) \approx \left(\frac{\mathrm{Vol}(\mathcal{L})}{\nu_n}\right)^{\frac{1}{n}}$$

Pour aller plus loin, il faudrait connaître ν_n. Nous savons depuis le collège que $\nu_2 = \pi$ et $\nu_3 = \frac{4}{3}\pi$, mais au-delà on admet que

$$\nu_n \sim \frac{1}{\sqrt{\pi n}} \left(\frac{2e\pi}{n}\right)^{n/2}.$$

En combinant ceci avec le raisonnement précédent, on trouve :

$$\lambda_1(\mathcal{L}) \approx \sqrt{\frac{n}{2e\pi}} \left(\sqrt{\pi n}\,\mathrm{Vol}(\mathcal{L})\right)^{\frac{1}{n}}.$$

Et si on fait sauter tous les petits facteurs embêtant, on trouve $\lambda_1(\mathcal{L}) \approx \sqrt{n}\,\mathrm{Vol}(\mathcal{L})^{\frac{1}{n}}$. Cette estimation est très pratique, mais il faut garder à l'esprit qu'il peut y avoir des cas particuliers où elle est complètement fausse. C'est en particulier le cas si la base du réseau qu'on considère contient *déjà* des vecteurs sensiblement plus courts que ce que l'heuristique gaussienne « prédit ».

Heureusement, on peut rendre ceci un peu plus carré, grâce à un théorème de Hermann Minkowski (1864–1909).

Théorème 1 (conséquence du théorème du corps convexe de Minkowski)**.** *Un réseau $\mathcal{L}$ de dimension d dans $\mathbb{R}^n$ contient un vecteur non-nul $\mathbf{x}$ tel que*

$$\|\mathbf{x}\| \leq 2\left(\frac{\mathrm{Vol}(\mathcal{L})}{\nu_d}\right)^{\frac{1}{d}}$$

La preuve de ce théorème est non-constructive et ne dit pas *comment* on pourrait trouver le vecteur court dont l'existence est garantie. Si on déplie la définition de ν_d, on trouve

$$\lambda_1(\mathcal{L}) \leq \sqrt{\frac{2d}{e\pi}} \left(\sqrt{\pi d}\,\mathrm{Vol}(\mathcal{L})\right)^{\frac{1}{d}} \leq \frac{\sqrt{d}}{2} \left(\sqrt{\pi d}\,\mathrm{Vol}(\mathcal{L})\right)^{\frac{1}{d}}$$

Il faut noter que le théorème de Minkowski donne une borne supérieure sur la taille du plus court vecteur d'un réseau. Le résultat (garanti) qu'on obtient est seulement deux fois pire que l'estimation au doigt mouillé qu'on avait tiré de l'heuristique gaussienne. Mais par contre, il n'y a pas de borne inférieure correspondante, et seule l'heuristique gaussienne permet d'estimer (avec précaution) la taille du plus court vecteur.

En gros, un réseau « aléatoire » n'a pas de raison de contenir des vecteurs sensiblement plus courts que ce que ces deux résultats suggèrent.

0.1.5 La constante de Hermite

Le théorème de Minkowski a de nombreuses conséquences importantes. Il implique en particulier que $\lambda_1(\mathcal{L})/\,\mathrm{Vol}(\mathcal{L})^{1/d}$ est majoré par une constante qui dépend uniquement de la dimension, et pas de la base du réseau.

On appelle *constante de Hermite* le supremum de $\lambda_1(\mathcal{L})^2/\,\mathrm{Vol}(\mathcal{L})^{2/d}$, pris sur tous les réseaux de dimension d. Les valeurs de ces constantes sont mystérieuses et mal connues. On sait que $\gamma_8 = 2$ et $\gamma_{24} = 4$, mais aucune

valeur plus grande n'est connue avec certitude. On ne sait même pas si la séquence est croissante ! On sait cependant que $\gamma_d \leq 1 + d/4$.

L'intérêt, c'est que n'importe quel réseau de dimension d contient un vecteur de longueur inférieure à $\lambda_1(\mathcal{L}) \leq \sqrt{\gamma_d}\,\mathrm{Vol}(\mathcal{L})^{1/d}$.

Ceci implique en particulier le résultat suivant, qui est généralement plus pratique à utiliser que le théorème 1 :

Théorème 2 (Borne de Minkowski simplifiée)**.** *Si $\mathcal{L}$ est un réseau de dimension d, alors $\lambda_1(\mathcal{L}) \leq \sqrt{d}\,\mathrm{Vol}(\mathcal{L})^{1/d}$.*

0.2 Problèmes algorithmiques dans les réseaux

Il y a de nombreux problèmes algorithmiques difficiles qui tournent autour des réseaux, mais les deux principaux sont les plus simples :

Définition 3 : Shortest Vector Problem (SVP)

> Étant donné une base d'un réseau $\mathcal{L}$, calculer un plus court vecteur de $\mathcal{L}$, c.a.d. un vecteur $\mathbf{x}$ tel que $\|\mathbf{x}\| = \lambda_1(\mathcal{L})$.

Définition 4 : Closest Vector Problem (CVP)

> Étant donné une base d'un réseau $\mathcal{L}$ et un vecteur $\mathbf{x}$, calculer un vecteur de $\mathcal{L}$ le plus proche possible de $\mathbf{x}$, c.a.d. un vecteur $\mathbf{y}$ tel que $\|\mathbf{x} - \mathbf{y}\|$ soit minimal.

Il faut noter qu'il s'agit de deux problèmes d'optimisation, et que même si on nous donnait la solution, il n'est pas évident de se convaincre qu'elle est optimale.

En plus, à première vue il n'est même pas évident que les problèmes sont simplement *décidables* : en effet, fournir une procédure qui s'arrête en temps fini en produisant le vecteur le plus court du réseau n'est pas si évident que ça, car le réseau contient une infinité de vecteurs.

Les deux problèmes SVP et CVP sont NP-durs (le paramètre principal est la dimension du réseau). Pour SVP, cela a été démontré pour la norme infinie en 1981 par Peter van Emde Boas [13]. Pour la norme euclidienne, cela a été conjecturé dès 1981, mais démontré seulement en 1998 par Miklós Ajtai (et encore, avec des réductions randomisées) [1]. Démontrer que CVP est NP-dur est assez facile. La première preuve (un peu compliquée) date de 1981 (Van Emde Boas encore). Une preuve fortement simplifiée a été donnée par Micciancio en 2001 [9], via une réduction à SUBSET SUM.

Dans la pratique, les deux problèmes sont *faciles* à résoudre en petite dimension ($n \leq 50$ disons), mais *complètement impossible* à résoudre en grande dimension ($n \geq 200$ disons). Pour SVP, il y a deux familles d'algorithmes qui sont utilisables : les algorithmes *d'énumération* qui fonctionnent en temps $2^{\mathcal{O}(n \log n)}$ et en espace faible d'une part, et les algorithmes de *crible* qui fonctionnent en temps et en espace $2^{\mathcal{O}(n)}$ — en pratique l'espace devient vite un problème.

On peut considérer aussi des versions approchées des deux problèmes en question.

Définition 5 : Approximate Shortest Vector Problem ($\mathbf{SVP}_\gamma$)

> Étant donné une base d'un réseau $\mathcal{L}$ et un paramètre $\gamma \geq 1$, calculer un vecteur $\mathbf{x}$ tel que $\|\mathbf{x}\| \leq \gamma \cdot \lambda_1(\mathcal{L})$.

Définition 6 : Approximate Closest Vector Problem ($\mathbf{CVP}_\gamma$)

> Étant donné une base d'un réseau $\mathcal{L}$, un vecteur $\mathbf{x}$ et un paramètre $\gamma \geq 1$ calculer un vecteur $\mathbf{y} \in \mathcal{L}$ tel que pour tout $\mathbf{z} \in \mathcal{L}$ on a : $\|\mathbf{x} - \mathbf{y}\| \leq \gamma \cdot \|\mathbf{x} - \mathbf{z}\|$ ($\mathbf{y}$ est au pire γ fois trop loin de $\mathbf{x}$).

Plus le facteur d'approximation est grand, plus les problèmes approchés sont faciles. Si γ est une constante (c.a.d. indépendant de la dimension du réseau), les problèmes approchés restent NP-durs. Plus précisément, Dinur, Kindler Raz et Safra ont montré en 2003 [3] que CVP est NP-dur tant que $\gamma \leq n^{c/\log\log n}$ pour n'importe quelle constante c.

Au contraire, on va voir qu'il existe des algorithmes polynomiaux pour résoudre les problèmes approchés, qui n'atteignent que des facteurs d'approximations exponentiellement grands. Et il y a des cas intermédiaires, comme $\gamma = \sqrt{n/\log n}$, qui n'est probablement pas NP-dur, mais pour lequel on ne connaît pas d'algorithme d'approximation efficace.

Enfin, il y a un dernier problème algorithmique intéressant, dérivé du théorème 1.

Définition 7 : Hermite Shortest Vector Problem ($\mathbf{HSVP}_\gamma$)

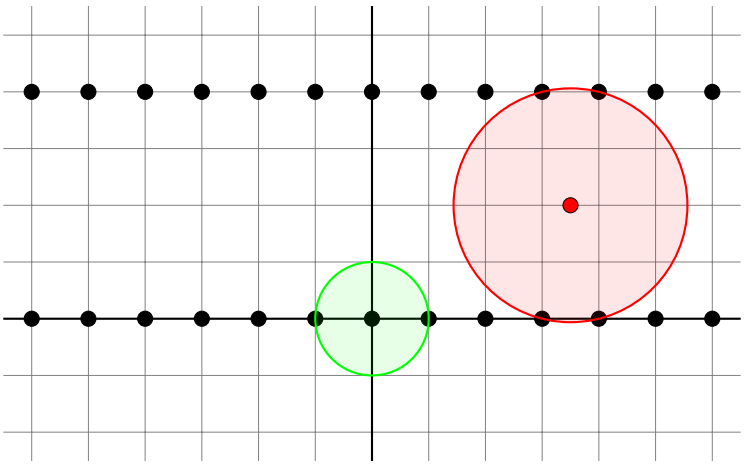

FIGURE 5 – SVP vs CVP

> Étant donné une base d'un réseau $\mathcal{L}$ de dimension d et un paramètre $\gamma \geq 1$, calculer un vecteur $\mathbf{x}$ tel que $\|\mathbf{x}\| \leq \gamma \times \text{Vol}(\mathcal{L})^{1/d}$.

L'avantage de ce dernier problème est qu'il est facile de vérifier les solutions... et qu'il est dur quand même. En plus, il a été démontré (par László Lovasz en 1986) que si on sait approximer HSVP avec facteur γ, on peut approximer SVP avec facteur γ^2 [8].

On peut remarquer que CVP est en quelque sorte plus général que SVP : dans SVP, on cherche le vecteur le plus proche du point de coordonnées $(0, \ldots, 0)$, tandis que dans CVP on chercher à partir d'un point arbitraire de l'espace. Il faut noter que les deux problèmes peuvent avoir des réponses assez différentes : la distance d'un point quelconque de l'espace au reste du réseau peut être sensiblement plus grande que le premier minimum (cf. figure 5)

0.2.1 (*) Décidabilité de SVP

Cette section démontre qu'il existe un algorithme (inefficace) qui résout SVP. Autrement dit, étant donné une matrice U, de taille $d \times n$, à coefficients entiers il s'agit de trouver le plus court vecteur non-nul qui s'écrive $\mathbf{y} = \mathbf{x}U$ (où $\mathbf{x}$ est un vecteur à coefficients entiers, et donc $\mathbf{y}$ aussi).

Ce n'est pas forcément évident *a priori* car on ne peut pas tester tous les $\mathbf{x}$ possibles, vu qu'il y en a une infinité. Notons $\mathbf{x} = (x_1, \ldots, x_n)$ le plus court vecteur du réseau. On a $\|\mathbf{x}\|^2 = {x_1}^2 + \cdots + {x_n}^2$. Par conséquent, comme tous ces carrés sont positifs, on a $|x_i| \leq \|\mathbf{x}\|$ pour $1 \leq i \leq n$. Maintenant, n'importe quel autre vecteur $\mathbf{y}$ du réseau (par exemple une ligne de U) est plus long que $\mathbf{x}$, et ceci donne par conséquent une borne sur les valeurs absolues des coordonnées du plus court vecteur.

On peut alors faire une recherche exhaustive dans le parallélépipède qui contient tous les $\mathbf{x}$ correspondant. Autrement dit : soit $\mathbf{y}$ un vecteur (non-nul) du réseau. Initialiser $\mathbf{x} \leftarrow \mathbf{y}$ et $B \leftarrow \|\mathbf{y}\|$. Pour tous les n-uplets d'entiers $\mathbf{z} = (z_1, \ldots, z_n)$ avec $-B \leq z_i \leq B$, faire : tester si $\mathbf{z}$ est un point du réseau ; le cas échéant, calculer sa norme ; si $|\mathbf{z}| < \|\mathbf{x}\|$, alors faire $\mathbf{x} \leftarrow \mathbf{z}$.

Lorsque cette procédure termine, $\mathbf{x}$ est le plus court vecteur. Elle est néanmoins très inefficace...

0.3 Algorithme LLL et améliorations

En 1982, Hendrik Lenstra [3] (né en 1949), Arjen Lenstra [4] (né en 1956) et László Lovász (né en 1948) ont inventé le célèbre algorithme qui porte leurs initiales.

L'algorithme LLL prend en entrée une base quelconque d'un réseau ainsi qu'un paramètre $\frac{1}{4} \leq \delta < 1$. Il s'exécute en temps polynomial (en le nombre de bits nécessaire pour écrire la base) et produit en sortie une base « *LLL-réduite avec un facteur δ* ». On ne décrira pas ici ni cette notion, ni l'algorithme LLL lui-même (les lecteurs curieux pourront consulter la section 0.4).

3. Il a aussi mis au point l'algorithme de factorisation basé sur les courbes elliptiques, qui est le meilleur pour trouver de petits facteurs premiers...
4. Il a aussi co-inventé le meilleur algorithme de factorisation des grands entiers, le crible algébrique...

Par exemple, avec la base suivante, dont tous les vecteurs sont assez grands :

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 2121390710 \\ 0 & 1 & 0 & 0 & 365500767 \\ 0 & 0 & 1 & 0 & 2647889201 \\ 0 & 0 & 0 & 1 & 1407355715 \\ 0 & 0 & 0 & 0 & 2585271343 \end{pmatrix},$$

l'exécution de l'algorithme LLL produit la base « LLL-réduite » :

$$LLL(B) = \begin{pmatrix} -7 & 28 & -35 & 3 & -36 \\ -4 & -51 & 24 & -2 & -22 \\ 50 & -8 & -29 & -72 & 2 \\ -36 & 33 & 41 & -59 & -48 \\ -70 & -41 & -65 & 7 & 11 \end{pmatrix}$$

dont les vecteurs sont beaucoup plus courts. Les deux matrices engendrent le même réseau, mais la deuxième en est une description beaucoup plus « commode ».

La complexité dans le pire des cas des meilleures implantations de LLL est de $\mathcal{O}\left(d^4 n \log B.(d + \log B)\right)$, où B désigne le nombre de bits des entrées de la matrice représentant la base du réseau fournie en entrée. Autant le dire tout de suite : l'algorithme est polynomial, mais ça peut coûter cher quand même dans le pire des cas. Souvent, en pratique, la complexité empirique est meilleure que ce que cette borne suggère.

Sur le plan pratique, l'algorithme LLL est disponible dans plusieurs logiciels open-source. Le plus facile d'utilisation est certainement `SageMath` (avec une syntaxe qui est quasiment du `python`). Il existe aussi une librairie `C` nommée `fplll` avec un utilitaire en ligne de commande, et surtout un paquet `python fpylll` qui permet de l'utiliser dans `python`. En fait, `fpylll` est inclus dans `SageMath`. Outre l'algorithme LLL, il contient aussi des algorithmes pour résoudre exactement CVP et SVP (en petite dimension bien sûr).

L'intérêt de l'algorithme LLL est donné tout entier dans le théorème suivant :

Théorème 3. *Soit $\alpha = 1/(\delta - \frac{1}{4})$ et soit $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ une base LLL-reduite avec un facteur δ d'un réseau euclidien $\mathcal{L}$. Alors :*

1. $\|\mathbf{b}_1\| \leq \alpha^{(d-1)/4} \left(\mathrm{Vol}(\mathcal{L})\right)^{1/d}$.
2. $\|\mathbf{b}_1\| \leq \alpha^{(d-1)/2} \lambda_1(\mathcal{L})$.
3. $\|\mathbf{b}_1\| \times \cdots \times \|\mathbf{b}_d\| \leq \alpha^{d(d-1)/4} \mathrm{Vol}(\mathcal{L})$.

Voici quelques commentaires.

Historiquement, l'algorithme LLL a été présenté avec $\delta = \frac{3}{4}$, ce qui donne $\alpha = 2$. On ne sait pas si l'algorithme est toujours polynomial avec $\delta = 1$ (qui donnerait $\alpha = \frac{4}{3}$). En pratique, $\delta = 0.99$ fonctionne assez bien, et c'est généralement le réglage par défaut dans les implantations efficaces de LLL.

Le premier vecteur d'une base LLL-reduite est relativement court. Il est au pire $\left(\alpha^{1/4}\right)^d$ fois plus long que la « borne de Minkowski » donnée par le théorème 1. Ceci implique que l'algorithme LLL permet de résoudre HSVP avec « facteur de Hermite » $\approx (4/3)^{d/4} \approx 1.0745^d$ (il est exponentiel en la dimension).

Mais le plus intéressant est le deuxième point : le premier vecteur d'une base LLL-réduite est $\sqrt{\alpha}^d$ fois plus long que le plus court vecteur du réseau. Ceci implique que l'algorithme LLL permet de résoudre SVP avec facteur d'approximation $\approx \sqrt{4/3}^d \approx 1.1547^d$ (il est encore exponentiel en la dimension). Par conséquent, l'algorithme LLL nous donne accès à un moyen simple de calculer des vecteurs « relativement courts » d'un réseau en temps polynomial : il suffit de prendre le premier vecteur d'une base LLL-réduite. Au passage, si un réseau contient *un* vecteur très court, et que tous les autres sont au moins $2^{d/2}$ fois plus longs, alors le deuxième point du théorème affirme que l'algorithme LLL va trouver ce vecteur très court en temps polynomial (car c'est le seul qui peut satisfaire le 2ème point du théorème).

Le troisième point du théorème montre que le défaut d'orthogonalité des bases LLL-reduites est majoré par $\alpha^{d^2/4}$. Autrement dit, l'algorithme LLL, à partir d'une base quelconque en entrée, produit une base dont le défaut d'orthogonalité est contrôlé (et donc, dont les vecteurs sont plutôt courts).

Dans la section 0.1.3, il est affirmé que tout réseau possède des bases dont le défaut d'orthogonalité est majoré par $(4/3)^{n(n-1)/4}$. En fait, l'algorithme LLL ne permet pas de les calculer en temps polynomial, car il faudrait pour cela poser $\delta = 1$...

Il faut noter que ces résultats sont valables dans le pire des cas. En pratique, l'algorithme LLL se comporte souvent mieux que prévu, et en particulier il peut renvoyer une base composée de vecteurs sensiblement plus courts que ce que le théorème annonce. En pratique, on observe plutôt un facteur de Hermite de l'ordre de 1.0219^n [5]. Pour des réseaux ayant des structures spéciales, la situation peut être encore plus favorable. Le facteur d'approximation pour SVP est généralement le carré de celui pour HSVP : de toute façon c'est compatible avec la réduction entre les deux.

Enfin, il faut noter qu'en deux dimensions ($d = 2$), LLL renvoie prouvablement les deux vecteurs les plus courts du réseau. Dans ce cas particulier, LLL est équivalent à un algorithme dû à Lagrange en 1775.

0.3.1 (*) Algorithme BKZ

Il existe un autre algorithme qui produit des bases améliorées : il s'agit de la BKZ-réduction (pour Block-Korkine-Zolotarev), dûe à Schnorr et Euchner en 1994 [12].

Il prend en argument un paramètre supplémentaire β (la taille du « bloc »), et son temps d'exécution est fortement exponentiel en β. En pratique, il produit des bases plus réduites. L'algorithme BKZ, s'il termine, atteint prouvablement un « facteur de Hermite » (approximation pour HSVP) de $\sqrt{\gamma_\beta}^{1+(n-1)/(\beta-1)}$. Par exemple, avec $\beta = 24$, cela donne $\approx 1.0305^n$. Encore une foi, la pratique est meilleure que la théorie : le facteur observé empiriquement est plutôt de l'ordre de 1.0125^n.

De même, si BKZ termine, il permet d'approximer SVP avec facteur $\gamma_\beta^{(n-1)/(\beta-1)}$. Avec $\beta = 24$, cela donne $\approx 1.0621^n$. Ces résultats sont décrits dans [5].

De manière asymptotique, l'algorithme BKZ (avec une taille de bloc β) atteint un facteur de Hermite de l'ordre de $\beta^{\frac{1}{2\beta}}$ en temps $\beta^{\frac{\beta}{2e}}$. Ceci a été amélioré en 2020 par Albrecht, Bai, Fouque, Kirchner, Stehlé et Wen avec un autre algorithme qui obtient le même résultat en temps $\approx k^{k/8}$ (cf. [**?**]).

En pratique, jusqu'à quelle dimension peut-on résoudre HSVP avec un facteur d'approximation linéaire ? Avec LLL on doit pouvoir monter jusqu'à la dimension 250. Avec BKZ-24, on doit pouvoir monter jusqu'à la dimension 500.

0.4 (*) Orthogonalisation de Gram-Schmidt

Cette section contient des compléments qui reposent sur le processus d'orthogonalisation de Gram-Schmidt.

0.4.1 Généralités

L'*orthogonalisation de Gram-Schmidt* est une procédure qui, à partir d'une base B, produit une base orthogonale $B^\star$ telle que $\mathbf{b}_1^\star = \mathbf{b}_1$ d'une part, et $(\mathbf{b}_1^\star, \ldots, \mathbf{b}_i^\star)$ engendre le même espace vectoriel que $(\mathbf{b}_1, \ldots, \mathbf{b}_i)$ d'autre part. On dit que les $\mathbf{b}_i^\star$ sont les *orthogonalisés de Gram-schmidt* (GSO) des $\mathbf{b}_i$. On note que B est une matrice $d \times n$ (qui engendre un réseau de dimension d dans $\mathbb{Z}^n$).

À chaque étape, $\mathbf{b}_i^\star$ est obtenu en projetant orthogonalement $\mathbf{b}_i$ sur l'espace engendré par $\mathbf{b}_1^\star, \ldots, \mathbf{b}_{i-1}^\star$. Le vecteur qui va de la projection à b_i est bien orthogonal aux précédents.

$$\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij} \mathbf{b}_j^\star \qquad \text{avec} \quad \mu_{ij} = \frac{\mathbf{b}_i \cdot \mathbf{b}_j^\star}{\|\mathbf{b}_j^\star\|^2}$$

En général, les $\mathbf{b}_i^\star$ ne sont pas à coordonnées entières mais rationnelles. On peut toutes les écrire comme des fractions de dénominateur $\|\mathbf{b}_1^\star\|^2 \times \cdots \times \|\mathbf{b}_n^\star\|^2$.

La matrice $B^\star$ n'est pas une *matrix orthogonale* (c.a.d. telle que $A^{-1} = A^t$) car il faudrait pour cela que les lignes soient de norme un, ce qui n'est pas le cas a priori.

La matrice de passage μ qui satisfait $B = \mu \times B^\star$ est formée des μ_{ij} :

$$\mu = \begin{pmatrix} 1 & & & & \\ \mu_{21} & 1 & & & \\ \ldots & \ldots & \ddots & & \\ \ldots & \ldots & \ldots & 1 & \\ \mu_{n1} & \mu_{n2} & \ldots & \mu_{n(n-1)} & 1 \end{pmatrix}$$

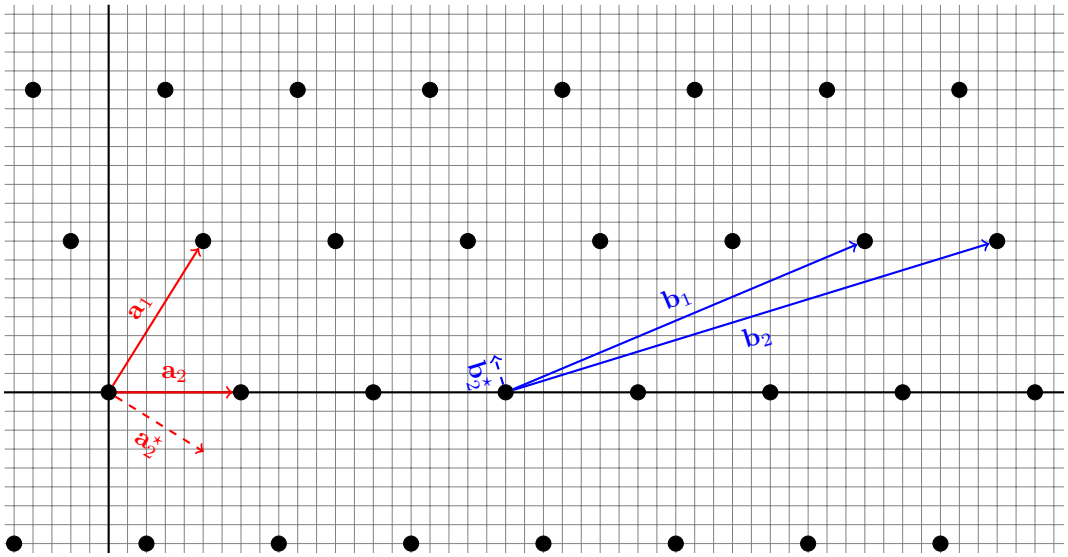

FIGURE 6 – Orthogonalisation d'une bonne base et d'une mauvaise base.

Ceci est similaire à la factorisation QR ($B = QR$ avec Q orthogonale et R triangulaire). Mais ici c'est transposé, et $B^\star$ n'est pas vraiment orthogonale.

Ceci implique que $\det(BB^t) = \det(\mu B^\star B^{\star t}\mu^t) = \det(B^\star B^{\star t}) = \|\mathbf{b}_1^\star\|^2 \times \cdots \times \|\mathbf{b}_n^\star\|^2$. Par conséquent, le volume du réseau est le produit des normes des $\mathbf{b}_i^\star$.

La figure 8 montre ce qui se passe quand on orthogonalise une bonne base et une mauvaise base. Si le premier vecteur de départ ($\mathbf{b}_1$) est très long, alors forcément les $\mathbf{b}_i^\star$ qui vont suivre seront tout petits (car le produit de leur norme est fixé).

Ceci implique au passage que si on permute les $\mathbf{b}_i$, alors cela peut complètement changer les $\mathbf{b}_i^\star$.

Plus généralement, les vecteurs de la base orthogonalisée sont toujours plus courts que ceux de départ. On a en effet :

Lemme 1. *Pour n'importe quelle base $B = (\mathbf{b_1}, \ldots, \mathbf{b_d})$ d'un réseau $\mathcal{L}$ dans $\mathbb{Z}^n$, on a :*

i) $\|\mathbf{b}_i^\star\| < \|\mathbf{b}_i\|$ *pour* $1 \leq i \leq d$.

ii) $\min_i \|\mathbf{b}_i^\star\| \leq \lambda_1(\mathcal{L})$.

Avant de prouver ceci, notons une implications : si on a une base « pas trop mal » (plutôt orthogonales, vecteurs plutôt courts), alors les vecteurs orthogonalisés ne seront peut-être pas être trop courts, et on obtiendra ainsi une borne *inférieure* intéressante sur la taille de $\lambda_1(\mathcal{L})$.

Démonstration

Premièrement :

$$\begin{aligned}\|\mathbf{b}_i^\star\|^2 = \mathbf{b}_i^\star \cdot \mathbf{b}_i^\star &= \left(\mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j^\star\right) \cdot \left(\mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_j^\star\right) \\ &= \|\mathbf{b}_i\|^2 - 2\sum_{j=1}^{i-1} \mu_{ij}\mathbf{b}_i \cdot \mathbf{b}_j^\star + \sum_{j=1}^{i-1} \mu_{ij}^2 \|\mathbf{b}_j^\star\|^2 \\ &= \|\mathbf{b}_i\|^2 - \sum_{j=1}^{i-1} {\mu_{ij}}^2 \|\mathbf{b}_j^\star\|^2 \qquad (1)\end{aligned}$$

Deuxièmement. Considérons un vecteur quelconque $\mathbf{x} \in \mathbb{Z}^n$, non-nul. On a $\mathbf{x}B = \sum_{i=1}^d x_i \mathbf{b}_i$. Notons i le

plus grand indice tel que $x_i \neq 0$. Alors :

$$\begin{aligned}\mathbf{x}B \cdot \mathbf{b}_i^\star &= \left(\sum_{j=1}^{n} x_j \mathbf{b}_j\right) \cdot \mathbf{b}_i^\star \\ &= \sum_{j=1}^{n} x_j \left(\mathbf{b}_j^\star + \sum_{k=1}^{j-1} \mu_{jk}\mathbf{b}_k^\star\right) \cdot \mathbf{b}_i^\star \\ &= x_i \mathbf{b}_i^\star \cdot \mathbf{b}_i^\star \\ &= x_i \|\mathbf{b}_i^\star\|^2.\end{aligned}$$

De plus, l'inégalité de Cauchy-Schwartz dit que : $|\mathbf{x}B \cdot \mathbf{b}_i^\star| \leq \|\mathbf{x}B\| \cdot \|\mathbf{b}_i^\star\|$. En combinant ceci avec ce qu'on avait juste avant, on trouve que : $|x_i| \cdot \|\mathbf{b}_i^\star\| \leq \|\mathbf{x}B\|$. Et comme x_i est un entier non-nul, ceci implique que $\mathbf{b}_i^\star$ est plus court que $\mathbf{x}B$. Autrement dit, n'importe quel vecteur du réseau est plus long que l'un des $\mathbf{b}_i^\star$. Par conséquent, n'importe quel vecteur du réseau (y compris le plus court) est plus long que le plus court des $\mathbf{b}_i^\star$.

0.4.2 Complexité de l'orthogonalisation de Gram-Schmidt

À première vue, calculer $B^\star$ à partir de B nécessite $\mathcal{O}\left(d^2 n\right)$ opérations arithmétiques : il y a $d(d-1)/2$ coefficients μ_{ij} et obtenir chacun d'entre eux nécessite un produit scalaire entre deux vecteurs de taille n. Ceci est néanmoins trompeur : ce sont des opérations sur des rationnels qui peuvent devenir de plus en plus gros.

Soit M une borne supérieure sur la norme des $\mathbf{b}_i$. Les coefficients de $\mathbf{b}_i^\star$ sont des rationnels où le numérateur et le dénominateur peuvent faire $\approx i \log_2 M$ bits, ce qui est majoré par $\mathcal{O}\left(d \log M\right)$. Par conséquent, $B^\star$ occupe donc $\mathcal{O}\left(nd^2 \log M\right)$ bits. Avec les algorithmes classiques, chaque opération sur des nombres de cette taille coûte $\mathcal{O}\left(d^2 \log^2 M\right)$, donc le coût total du calcul de $B^\star$ est $\mathcal{O}\left(nd^4 \log^2 M\right)$ (c'est sensiblement plus qu'annoncé au-dessus).

Justification de ces affirmations. Notons $D_i = \|\mathbf{b}_1^\star\|^2 \times \cdots \times \|\mathbf{b}_i^\star\|^2$, avec $D_0 = 1$.

CLAIM : on prétend que les coefficients de $\mathbf{b}_i^\star$ peuvent tous s'écrire comme des fractions de dénominateur D_{i-1} et que μ_{ij} peut s'écrire comme une fraction de dénominateur D_j.

Démonstration

Preuve par récurrence : pour $i = 1$, c'est trivial. $D_1 = \|\mathbf{b}_1\|^2$ et il n'y a pas de coefficient μ_{01}.

Pour $i > 1$, c'est assez facile à voir pour μ_{ij} : comme les coefficients de $\mathbf{b}_j^\star$ sont de dénominateur D_{j-1}, il s'ensuit que c'est aussi le cas du produit scalaire $\mathbf{b}_i \cdot \mathbf{b}_j^\star$. Et donc μ_{ij} est de dénominateur D_j car on divise par $\|\mathbf{b}_i^\star\|^2$. Ensuite, pour $\mathbf{b}_i^\star$, la somme des $\mu_{ij}\mathbf{b}_j^\star$ peut se mettre à dénominateur D_{i-1} (c'est $\mu_{i(i-1)}$ qui fait intervenir le plus gros dénominateur). □

Notons X une borne supérieure sur les carrés des normes des vecteurs de la base de départ : $\|\mathbf{b}_i^\star\|^2 \leq \|\mathbf{b}_i\|^2 \leq X$. Alors $D_i \leq X^i$. Les coefficients de $\mathbf{b}_i^\star$ peuvent être représentés par des fractions de dénominateur $\leq X^{i-1}$, et comme $\|\mathbf{b}_i^\star\|^2 \leq X$, les dénominateurs sont forcément plus petits que X^i en valeur absolue (sinon la norme serait trop grande). Chacun de ces coefficients occupe donc au maximum $(2i-1)\log_2 X$ bits. Ceci est majoré par $\mathcal{O}\left(d \log_2 X\right)$.

0.4.3 Garanties offertes par LLL sur les GSO

L'algorithme LLL produit une base composée de vecteur « courts ». En fait, pour atteindre ce but, il produit une base pour laquelle la norme des GSO ne baisse pas trop vite.

Une base B est *LLL-réduite avec un facteur* δ si les deux conditions suivantes sont réunies :

1. (*Size reduction*) $|\mu_{ij}| < \frac{1}{2}$ pour tout $1 \leq i < j \leq d$.
2. (*Condition de Lovász*) $\|\mathbf{b}_{i+1}^\star + \mu_{i+1,i}\mathbf{b}_i^\star\|^2 \geq \delta \|\mathbf{b}_i\|^2$ pour tout $1 \leq i < d$.

La condition de Lovász signifie grosso-modo que si on permute $\mathbf{b}_{i+1}^\star$ et $\mathbf{b}_i$ dans la base, ça ne fait pas trop chuter la norme des GSO.

Tout l'intérêt de la définition est le suivant :

$$\begin{aligned}
\|\mathbf{b}_{i+1}^\star + \mu_{i+1,i}\mathbf{b}_i^\star\|^2 &= \|\mathbf{b}_{i+1}^\star\|^2 + {\mu_{i+1,i}}^2\|\mathbf{b}_i^\star\|^2 && \text{(car ils sont orthogonaux)}\\
&\leq \|\mathbf{b}_{i+1}^\star\|^2 + \frac{1}{4}\|\mathbf{b}_i^\star\|^2 && \text{(size-reduction)}\\
\delta\|\mathbf{b}_i^\star\|^2 &\leq \|\mathbf{b}_{i+1}^\star\|^2 + \frac{1}{4}\|\mathbf{b}_i^\star\|^2 && \text{(condition de Lovász)}\\
\left(\delta - \frac{1}{4}\right)\|\mathbf{b}_i^\star\|^2 &\leq \|\mathbf{b}_{i+1}^\star\|^2
\end{aligned}$$

Donc, dans une base LLL-réduite, la norme des GSO diminue de manière contrôlée. En particulier, $\|\mathbf{b}_i^\star\| \geq (\delta - 1/4)^{(i-1)/2}\|\mathbf{b}_1\|$. Et par exemple, on retombe sur une des garanties du théorème 3. En effet, $\lambda_1(L) \geq \min_i \|\mathbf{b}_i^\star\| \geq (\delta - 1/4)^{(d-1)/2}\|\mathbf{b}_1\|$.

0.4.4 Qualité de la réduction de réseau

Quand on s'intéresse à une base réduite (par LLL, BKZ, ou n'importe quel autre algorithme), on observe généralement que les normes $\|b_1^\star\|, \|b_2^\star\|, \ldots$ décroissent exponentiellement vite. C'est très visible si on trace la courbe des $\log \|b_i^\star\|$: ils sont généralement alignés. Autrement dit, on a généralement $\log \|b_i^\star\| \approx d - c \times i$ avec $c > 0$.

La vitesse de cette baisse est un des meilleurs moyens de juger la qualité de la réduction de réseau.

Si un algorithme de réduction de réseaux peut garantir que $c\|b_{i+1}^\star\| \geq \|b_i^\star\|$ pour une certaine constante $c > 1$, alors $\operatorname{Vol}\mathcal{L} = \|b_1^\star\| \times \cdots \times \|b_d^\star\| \geq c^{d(d-1)/2}\|b_1\|$. Par conséquent, cet algorithme offre un « facteur de Hermite » égal à $\|b_1\|/\operatorname{Vol}\mathcal{L}^{\frac{1}{d}} = \sqrt{c}^{d-1}$.

D'autre part, comme on a alors $c^d\|b_d^\star\| \geq \|b_1\|$, donc $\|b_1\| \leq c^d\lambda_1(\mathcal{L})$.

On voit donc le lien direct qu'il peut y avoir entre la capacité d'approximation de (H)SVP offerte par un algorithme de réduction de réseau d'une part, et le rythme de diminution de la taille des vecteurs orthogonalisés.

0.4.5 Algorithmes d'énumération pour SVP/CVP

Dans cette sous-section, on présente un algorithme capable de résoudre SVP, ce qui, a première vue, n'est pas si évident que ça.

Considérons un plus court vecteur non-nul $\mathbf{x}$ dans un réseau de base B. Il s'écrit $\mathbf{x} = x_1\mathbf{b}_1 + \cdots + x_d\mathbf{b}_d$.

Alors :

$$\mathbf{x} = \sum_{i=1}^{d} x_i\mathbf{b}_i = \sum_{i=1}^{d} x_i\left(\mathbf{b}_i^\star + \sum_{j=1}^{i-1}\mu_{ij}\mathbf{b}_j^\star\right) = \sum_{j=1}^{d}\left(x_j + \sum_{i=j+1}^{d}\mu_{ij}x_i\right)\mathbf{b}_j^\star$$

Comme les GSO sont orthogonaux, on a :

$$\|\mathbf{x}\|^2 = \sum_{j=1}^{d}\left(x_j + \sum_{i=j+1}^{d}\mu_{ij}x_i\right)^2\|\mathbf{b}_j^\star\|^2$$

Tous les termes de cette somme sont des carrés, donc ils sont tous positifs. En particulier, ils sont tous plus petit que $\|\mathbf{x}\|^2$. Si on regarde le dernier, on trouve alors que ${x_d}^2\|\mathbf{b}_d^\star\|^2 \leq \|\mathbf{x}\|^2$, ou encore $|x_d| \leq \|\mathbf{x}\|/\|\mathbf{b}_d^\star\|$.

Supposons qu'on connaisse une borne supérieure M sur la longueur de $\mathbf{x}$ (par exemple, la borne de Minkowski, ou bien la taille de n'importe quel vecteur connu pour être dans le réseau). Et voilà déjà x_d contraint de vivre dans un intervalle fini !

Ensuite, pour les autres coefficients, on peut dire que

$$\left|x_i + \sum_{j=i+1}^{d}\mu_{ij}x_j\right| \leq \frac{M}{\|\mathbf{b}_i^\star\|}$$

Donc x_i est enfermé dans un intervalle (centré en $\sum_{j=i+1}^{d}\mu_{ij}x_j$) de taille bornée. Ceci aboutit à une stratégie algorithmique pour résoudre SVP :

— Partir d'une base quelconque.
— Déterminer l'intervalle I_d dans lequel x_d doit se trouver.
— Pour chaque valeur possible de $x_d \in I_d$ (il y en a un nombre fini) :
  — Déterminer l'intervalle I_{d-1} dans lequel x_{d-1} doit se trouver.
  — Pour chaque valeur possible de $x_{d-1} \in I_{d-1}$ (il y en a un nombre fini) :
    — ...
      — Une fois qu'on a x_1, on a une valeur possible de $\mathbf{x}$. Calculer sa norme et le garder si c'est le plus court vecteur non-nul qu'on a rencontré jusque-là.
— Renvoyer le plus court vecteur non-nul rencontré.

Plus la base B de départ est de bonne qualité, plus la borne sur la taille des coefficients de $\mathbf{x}$ en base B est forte et plus cet algorithme va aller vite. Si on utilise une base LLL-réduite avec $\delta = 3/4$, on utilise $\|\mathbf{b}_1\|$ comme borne sur la taille du plus court vecteur et on trouve que la taille de l'intervalle dans lequel vit chaque coefficient est majorée par :

$$|I_l| \leq \frac{\|\mathbf{b}_1\|}{\|\mathbf{b}_k^\star\|} \leq \frac{\|\mathbf{b}_1\|}{(1/2)^{(k-1)/2}\|\mathbf{b}_1\|} \leq 2^{(k-1)/2}$$

Le nombre de valeurs de $\mathbf{x}$ à tester est donc :

$$|I_1| \times \cdots \times |I_d| \leq \frac{1}{\sqrt{2}} 2^{\sum_{i=1}^{d} i/2} \leq 2^{d(d+1)/4}$$

Donc, au total, il y a environ $2^{d^2/4}$ valeurs de $\mathbf{x}$ à tester. Facile ! Cette procédure est assez ancienne. On en trouve une description (en deux dimensions) dans l'édition de 1969 de *The Art of Computer Programming* de Knuth [7].

0.5 Algorithmes polynomiaux pour approcher CVP

Dans cette section, on considère un réseau $\mathcal{L}$, de rang plein, dont on possède une base $B = (\mathbf{b_1}, \ldots, \mathbf{b_n})$, et on cherche à résoudre CVP dedans de manière approchée, pour un vecteur « cible » $\mathbf{y}$.

Une première remarque c'est que la situation est beaucoup plus facile si la base B qu'on nous fournit est orthogonale. Par exemple, si c'est le cas, alors le plus court vecteur du réseau est forcément déjà dedans ! De même, CVP est facile aussi à résoudre dans ce cas-là.

Poursuivant cette idée-là, László Babai (né en 1950) a observé en 1986 [2] que même si on ne dispose pas d'une base orthogonale, mais d'une base de « bonne qualité » (défaut d'orthogonalité faible), alors on peut espérer s'en tirer assez bien en pratique, au moins tant que la dimension n'est pas trop grande. Ceci lui a permis de proposer deux algorithmes simples pour approcher CVP : la méthode de l'arrondi et la méthode de l'hyperplan le plus proche.

0.5.1 Méthode du plongement

Il s'agit d'une technique heuristique pour résoudre CVP en le ramenant à SVP (ce qui permet d'utiliser LLL pour le résoudre approximativement). On forme le réseau $\mathcal{L}'$ de dimension $n+1$ engendré par

$$B' = \begin{pmatrix} \mathbf{b}_1 & 0 \\ \mathbf{b}_2 & 0 \\ \vdots & \\ \mathbf{b}_n & 0 \\ \mathbf{y} & 1 \end{pmatrix}$$

Si $\mathbf{z}$ est le vecteur le plus court de $\mathcal{L}$, alors $(\mathbf{z}, 0)$ appartient à $\mathcal{L}'$ et il a la même longueur. Les deux réseaux ont par ailleurs le même volume (car $\det B' = \det B$) et (à un près) ils ont la même dimension. On pourrait donc s'attendre à ce que leurs plus courts vecteurs soient à peu près de la même taille.

Considérons la solution $\mathbf{x} \in L$ de notre instance de CVP. Le vecteur $(\mathbf{y} - \mathbf{x}, 1)$ appartient à $\mathcal{L}'$, et il est « court » (sa longueur est la distance entre $\mathbf{y}$ et le point du réseau le plus proche). Ainsi, si la distance de $\mathbf{y}$ au réseau $\mathcal{L}$ est bien plus petite que $\lambda_1(\mathcal{L})$, on pourrait espérer que ce vecteur $(\mathbf{y} - \mathbf{x}, 1)$ est un plus court vecteur de $\mathcal{L}'$. Si tel était le cas, alors on pourrait résoudre notre instance de CVP en résolvant SVP dans $\mathcal{L}'$.

Et si $\mathbf{y}$ était *vraiment* très près d'un point du réseau $\mathcal{L}$, alors même un algorithme qui résout SVP approximativement (comme LLL) permettrait de révéler la solution exacte.

Il faut noter que cette méthode est heuristique : il y a des cas où elle échoue (de manière déterministe).

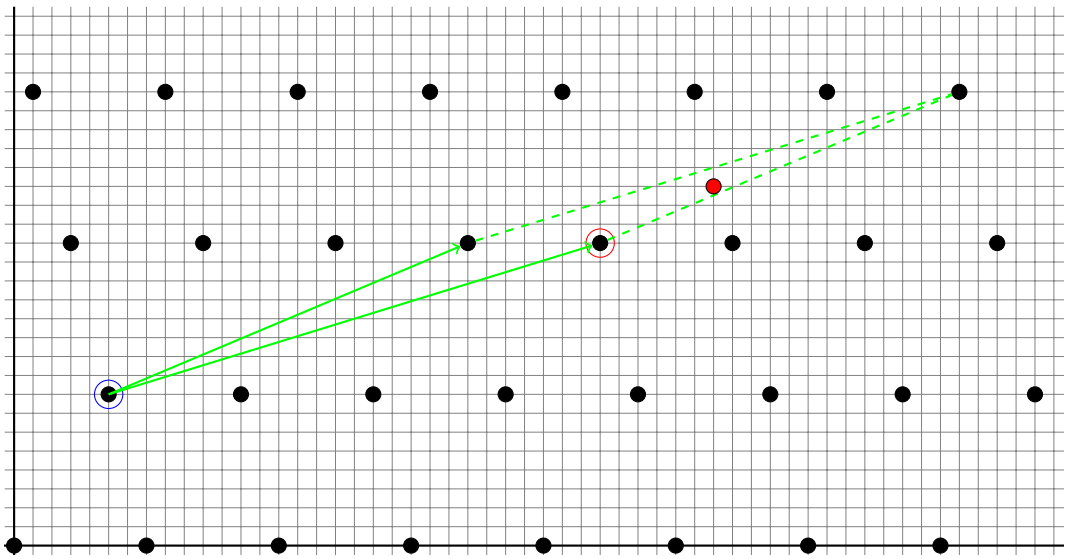

FIGURE 7 – Tentative de résolution de CVP avec la méthode de l'arrondi. La cible est le point rouge. Comme la base est très mauvaise, le sommet du parallélépipède le plus proche de la cible... en est assez éloigné, et ce n'est pas le CVP.

0.5.2 Méthode de l'arrondi

Puisque $\mathcal{B}$ est une base de $\mathbb{R}^n$ (en tant qu'espace vectoriel), on peut écrire

$$\mathbf{y} = \sum_{i=1}^{n} \lambda_i \mathbf{b}_i,$$

où les λ_i sont des *réels* (et pas forcément des entiers). Maintenant, si on *arrondit* les λ_i, alors on obtient un point du réseau !

Si on arrondit vers zéro, alors on obtient un point $\mathbf{b}$ (entouré en bleu sur la figure 7) tel que la cible $\mathbf{y}$ est contenue dans le parallélépipède fondamental ancré en $\mathbf{b}$. En arrondissant au plus près, on obtient un sommet du parallélépipède fondamental (entouré en rouge) qui est plus proche de la cible.

La figure 7 montre que si la base est « mauvaise » (vecteurs longs et peu orthogonaux), ça ne risque pas de bien marcher. En effet, si le parallélépipède est très étiré, son sommet le plus proche de la cible peut en être assez éloigné (plus en tout cas que d'autres points en dehors du parallélépipède).

Notons $\lfloor x \rceil$ l'entier le plus proche de x. On a dit qu'on posait :

$$\mathbf{x} = \sum_{i=1}^{n} \lfloor \lambda_i \rceil \, \mathbf{b}_i$$

Et il est clair que c'est bien un point du réseau $\mathcal{L}$. Le problème consiste à voir s'il est loin ou près de $\mathbf{y}$. Pour cela on écrit :

$$\|\mathbf{y} - \mathbf{x}\| = \left\| \sum_{i=1}^{n} (\lambda_i - \lfloor \lambda_i \rceil) \, \mathbf{b}_i \right\| \leq \sum_{i=1}^{n} \|\lambda_i - \lfloor \lambda_i \rceil \, \mathbf{b}_i\| \leq \frac{1}{2} \sum_{i=1}^{n} \|\mathbf{b}_i\|$$

(Le passage à la première inégalité utilise simplement l'inégalité triangulaire ; pour passer à la second on utilise le fait que l'entier le plus proche est forcément à distance $\leq 1/2$). On voit que plus les vecteurs de la base $\mathcal{B}$ sont courts, plus cette technique va fournir un vecteur proche. Ce qui n'est pas facile, c'est de se convaincre que le facteur d'approximation est garanti. En fait, il est possible (mais difficile) de démontrer que :

Théorème 4 (Babai, 1986)**.** *La méthode de l'arrondi utilisée avec des bases LLL-réduites avec un facteur* $\delta = 3/4$ *approxime CVP avec un facteur* $1 + 2d(9/2)^{d/2}$.

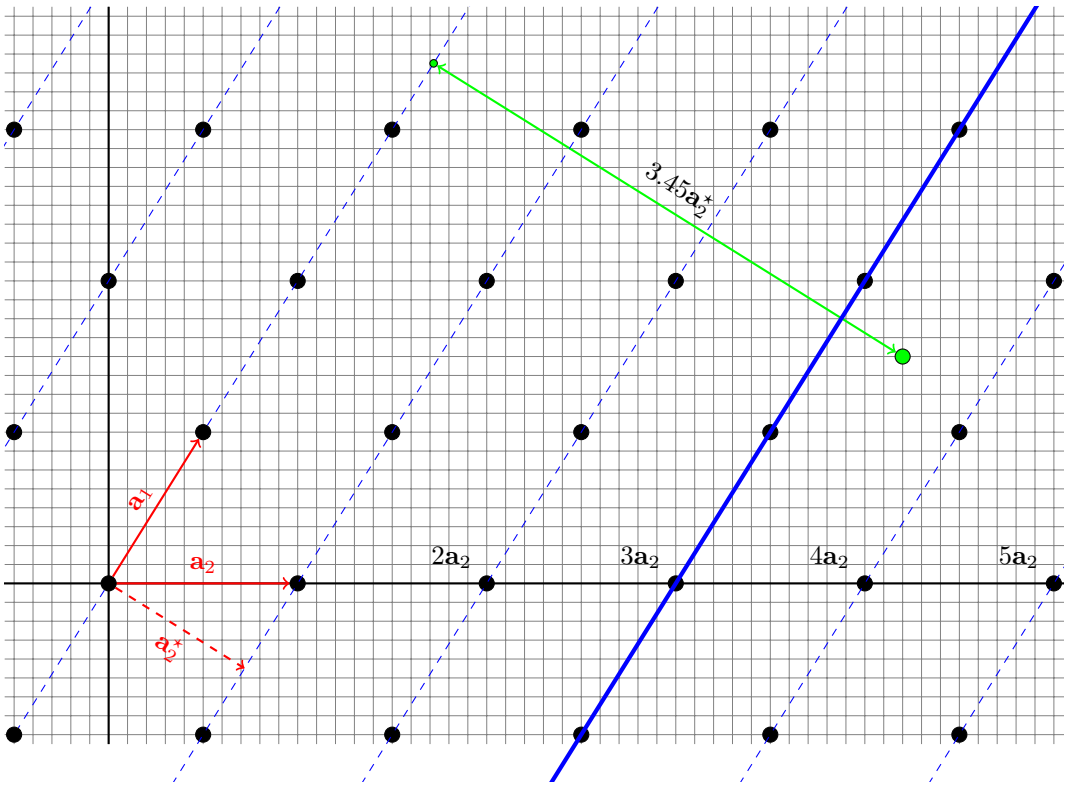

FIGURE 8 – Méthode de l'hyperplan le plus proche. Les lignes bleues pointillées sont les « hyperplans » orthogonaux à $\mathbf{a}_2^\star$. Il faut se déplacer de 3.45 fois $\mathbf{a}_2^\star$ pour projeter la cible orthogonalement sur l'espace engendré par $\mathbf{a}_1$. En arrondissant, on se restreint à chercher une solution dans l'hyperplan $3\mathbf{a_2} + \mathbb{Z}\mathbf{a}_1$ (en bleu gras).

0.5.3 (*) Méthode de l'hyperplan le plus proche

Dans la méthode de l'arrondi présentée ci-dessus, on fait tous les arrondis en même temps. On peut en fait obtenir de meilleurs résultats en les faisant les uns après les autres, séquentiellement (le suivant peut « corriger » le précédent dans une certaine mesure). Ceci aboutit à l'algorithme polynomial le plus utilisé pour approximer CVP.

L'algorithme nécessite le calcul des GSO. Il fonctionne de la façon suivante :

1. Poser $\mathbf{z} \leftarrow \mathbf{y}$.
2. Pour $j = n, n-1, \ldots, 2, 1$ faire :
   (a) Calculer $x \leftarrow (\mathbf{z} \cdot \mathbf{b}_j^\star)/\|\mathbf{b}_j^\star\|^2$
   (b) Mettre à jour $\mathbf{z} \leftarrow \mathbf{z} - \lfloor x \rceil \mathbf{b}_j$
3. Renvoyer $\mathbf{y} - \mathbf{z}$

La figure 8 illustre l'algorithme. L'idée générale est qu'à l'étape, on projette orthogonalement $\mathbf{z}$ sur l'hyperplan (c.a.d. le sous-espace vectoriel) engendré par $\mathbf{b_1}, \ldots, \mathbf{b_{j-1}}$; la projection est « parallèle » à $\mathbf{b}_j^\star$. Dit autrement : x est calculé de telle sorte que $\mathbf{z} - x\mathbf{b}_j^\star$ appartient au sous-espace engendré par $\mathbf{b_1}, \ldots, \mathbf{b_{j-1}}$. L'intérêt de la manoeuvre, c'est qu'on perd une dimension ; on pourrait donc envisager de calculer récursivement le vecteur le plus proche de $\mathbf{z} - x\mathbf{b}_j^\star$ dans le sous-réseau engendré par $(\mathbf{b_1}, \ldots, \mathbf{b_{j-1}})$. Mais ensuite, il faudrait en tirer une solution du problème de départ. Le « truc » consiste à dire que si $\mathbf{z} - x\mathbf{b}_j^\star$ tombe bien dans le sous-espace, alors $\mathbf{z} - \lfloor x \rceil \mathbf{b}_j$ ne doit pas tomber trop loin. Autrement dit, si $\tilde{\mathbf{y}}$ est un point du sous-réseau proche de $\mathbf{z} - x\mathbf{b}_j^\star$, alors $\tilde{\mathbf{y}} + \lfloor x \rceil \mathbf{b}_j$ est un point du réseau, probablement pas trop éloigné de $\mathbf{z}$.

Comme à chaque étape on déplace $\mathbf{z}$ d'un multiple entier de $\mathbf{b}_i$, le déplacement total $\mathbf{y} - \mathbf{z}$ est bien un élément du réseau.

En fait, le CVP appartient bien à l'hyperplan le plus proche de $\mathbf{z}$, mais identifier cet hyperplan n'est pas facile. Si la base de départ est mauvaise, alors il n'est pas du tout certain qu'on va identifier le bon hyperplan à chaque étape. En particulier, si $\mathbf{b}_n^\star$ est beaucoup plus court que $\mathbf{b}_n$, alors on risque de beaucoup se tromper lors de la première étape. Par contre, si on fait ça avec des bases LLL-réduite, l'erreur est contrôlée et on peut obtenir des garanties sur le facteur d'approximation.

Cette méthode offre une meilleure garantie que la méthode de l'arrondi. En plus, c'est plus facile à démontrer. Le théorème suivant montre que tout ce que LLL donne pour SVP, on peut l'obtenir aussi pour CVP.

Théorème 5 (Babai, 1986)**.** *La méthode de l'hyperplan le plus proche utilisé avec des bases LLL-réduites avec un facteur δ approxime CVP avec un facteur $\alpha^{n/2}$, où $\alpha = 1/(\delta - 1/4)$ comme dans LLL.*

Plus généralement, si une base a été réduite avec un « facteur de Hermite » de l'ordre de c^d, autrement dit si $\|b_1\| \leq c^d \operatorname{Vol}(\mathcal{L})^{\frac{1}{d}}$ alors l'algorithme de l'hyperplan le plus proche permet de résoudre CVP avec facteur d'approximation c^{2d}.

Bibliographie

[1] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 10–19. ACM, 1998.

[2] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Comb.*, 6(1) :1–13, 1986.

[3] Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is np-hard. *Comb.*, 23(2) :205–243, 2003.

[4] Steven D. Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 2012.

[5] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.

[6] Antoine Joux. *Algorithmic Cryptanalysis.* Chapman & Hall/CRC, 1st edition, 2009.

[7] Donald E. Knuth. *The Art of Computer Programming, Volume II : Seminumerical Algorithms.* Addison-Wesley, 1969.

[8] L. Lovasz. *An Algorithmic Theory of Numbers, Graphs, and Convexity.* CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1986.

[9] Daniele Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Trans. Inf. Theory*, 47(3) :1212–1215, 2001.

[10] Phong Q Nguyen. Public-key cryptanalysis. *Recent Trends in Cryptography, Contemp. Math*, 477 :67–120, 2009.

[11] Phong Q. Nguyen. Hermite's constant and lattice algorithms. In Phong Q. Nguyen and Brigitte Vallée, editors, *The LLL Algorithm - Survey and Applications*, Information Security and Cryptography, pages 19–69. Springer, 2010.

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

[13] Peter van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice.* Report. Department of Mathematics. University of Amsterdam. Department, Univ., 1981. Available online at `https://staff.fnwi.uva.nl/p.vanemdeboas/vectors/mi8104c.html`.