Chapitre 5

Réseaux euclidiens et applications

Question 1 : On considère le réseau engendré par les lignes de :

$$B = \begin{pmatrix} 11 & 1 & & \\ 19 & & 1 & \\ 29 & & & 1 \end{pmatrix}$$

Trouver un vecteur court (de norme 2) dans ce réseau.

5.1 Équations diophantiennes linéaires

On s'intéresse à l'équation linéaire sur les entiers *modulo* m :

$$a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} = x_n \mod m \qquad (\heartsuit)$$

où les coefficients (entiers $a_1, \ldots, a_{n-1}$) et m sont connus. On suppose que le PGCD des a_i vaut 1. La solution $x_1 \equiv \cdots \equiv x_n \equiv 0 \bmod n$ n'est pas considérée comme intéressante !

Question 2 : Est-il difficile de trouver une solution non-nulle de $(\heartsuit)$?

Question 3 : Expliquer pourquoi, si on sait résoudre $(\heartsuit)$, alors on peut aussi résoudre :

$$b_1x_1 + b_2x_2 + \cdots + b_nx_n = 0 \mod m \qquad (\clubsuit)$$

Question 4 : On considère le réseau $\mathcal{L}$ engendré par les lignes de :

$$B = \begin{pmatrix} 1 & & & & a_1 \\ & 1 & & & a_2 \\ & & \ddots & & \vdots \\ & & & 1 & a_{n-1} \\ & & & & m \end{pmatrix}$$

Justifier que $(x_1, \ldots, x_n)$ est une solution de $(\heartsuit)$ équivaut à dire que $\mathbf{x} = (x_1, \ldots, x_n)$ appartient au réseau.

Question 5 : En utilisant l'heuristique gaussienne, ou (mieux) la borne de Minkowski, montrer que $(\heartsuit)$ admet une solution à « petits » coefficients. Quelle est leur taille ?

Question 6 : Application « amusante » de la question précédente. Soit p un nombre premier ; on considère un entier $0 < a < p$. Démontrer qu'il existe deux entiers x, y tels que $a \equiv x/y \bmod p$ (p un nombre premier) avec $x, y \approx \sqrt{p}$.

Question 7 : On cherche maintenant à résoudre $(\heartsuit)$ avec des contraintes additionnelles : on impose que $|x_i| < X_i$, où les bornes X_i sont données (par rapport à l'exercice 5.5, on cherche une petite solution mais avec un déséquilibre potentiel entre les x_i) . On note $X = \prod X_i$.

On considère le réseau modifié engendré par les lignes de :

$$B' = \begin{pmatrix} \frac{X}{X_1} & & & & a_1\frac{X}{X_n} \\ & \frac{X}{X_2} & & & a_2\frac{X}{X_n} \\ & & \ddots & & \vdots \\ & & & \frac{X}{X_{n-1}} & a_{n-1}\frac{X}{X_n} \\ & & & & m\frac{X}{X_n} \end{pmatrix}$$

Supposons qu'il existe une solution de (♡) et que $X < m$. Montrer (en utilisant l'heuristique gaussienne), que résoudre SVP dans ce réseau devrait permettre de trouver une solution qui satisfait les bornes.

Question 8 : Suite de l'exercice 5.7. Supposons que $m < X$. Montrer (avec un argument de dénombrement) qu'une solution qui satisfait les bornes existe forcément.

5.2 Autour de RSA et de l'attaque de Wiener

On part d'une paire de clefs RSA, c'est-à-dire de e, d et $N = pq$ tels que $ed \equiv 1 \bmod \phi(N)$, où $\phi(N) = (p-1)(q-1)$. On sait donc que $\phi(N) = N - t$ avec $t \approx 2\sqrt{N}$.

Question 9 : Justifier qu'il existe un entier $k < \min(e, d)$ tel que $ed = 1 + k\phi(N)$.

Question 10 : Supposons qu'on ait une clef publique RSA. Pourquoi est-ce que tenter retrouver d en résolvant $ed = 1 + k(N - t)$ avec $0 \leq k < e$ et $t \leq 3\sqrt{N}$ ne marche pas ?

Question 11 : Peut-on avoir à la fois un « petit » exposant public et un « petit » exposant secret dans RSA ?

Question 12 : Pour empêcher l'attaque de Wiener, on peut remplacer e par $e + x\phi(N)$. Quelle taille de x garantit que l'attaque de Wiener ne marchera pas ?

5.3 Générateurs pseudo-aléatoires

5.3.1 Généralités

▷ **Question 13 :** On considère un PRG $\mathcal{G}$. Démontrer que sa sortie est (à peu près) uniformément distribuée, c.a.d. que pour polynôme p, et tout $x \in \{0,1\}^n$, on a $\mathbb{P}\left[\mathcal{G}(s, 1^n) = x\right] < 1/p(n)$ — la probabilité est prise sur le choix de la graine s.

▷ **Question 14 :** On considère un PRG $\mathcal{G}$, initialisé avec une graine s choisie aléatoirement dans $\{0,1\}^n$. Démontrer que la sortie de $\mathcal{G}$ est *imprédictible*, c.a.d. que pour tout t (polynomial en n), il n'existe pas d'algorithme polynomial qui reçoit $\mathcal{G}(s, 1^t)$ en entrée, capable de produire $\mathcal{G}(s, 1^{t+1})$ avec un avantage non-négligeable.

▷ **Question 15 :** On considère un PRG $\mathcal{G}$. Démontrer que pour tout polynôme p, la fonction $s \mapsto \mathcal{G}(s, 1^{p(|s|)})$ est à sens unique.

▷ **Question 16 :** Supposons qu'il existe une famille de fonctions à sens unique. Démontrer que $P \neq NP$.

▷ **Question 17 :** Une stratégie tentante pour construire un PRG est d'utiliser une famille de fonctions à sens unique en « mode compteur ». Par exemple, supposons qu'on ait des fonctions à sens unique $F_1, F_2, \ldots$ où $F_i : \{0,1\}^i \to \{0,1\}^i$. Le PRG reçoit une graine x de s bits ; il doit produire t bits. On suppose que $t = ks$ (ça simplifie et ce n'est pas un problème, quite à tronquer la sortie). La sortie est :

$$F_s(x), F_s(x+1), F_s(x+2), \ldots, F_s(x+t-1).$$

Ceci est-il forcément un bon PRG ?

5.3.2 Quelques exemples de PRG « conventionnels » (non-crypto)

▷ **Question 18 :** Les LFSR (*Linear Feedback Shift Registers*) sont une famille d'algorithmes qui génèrent un flot de bits à partir d'un état interne de taille fixe. Étant donné un état initial $a \in \{0,1\}^n$ et une séquence de coefficients $c \in \{0,1\}^n$, on produit la séquence de bits $(a_i)_{i \geq n}$ avec la relation $a_{n+i} = \sum_{j=0}^{n-1} c_{n-j} a_{i+j}$ (modulo 2). En supposant que la « graine » est formée de a_i pour $0 \leq i < n$ (et que les coefficients c sont public), au bout de combien de bits observés $(a_n, a_{n+1}, \dots)$ peut-on distinguer la sortie du LFSR de bits aléatoires ?

▷ **Question 19 :** Même question si les coefficients c font eux aussi partie de la graine.

▷ **Question 20 :** Démontrer que la fonction `xorshift128()` n'est pas un bon PRG.

```
u32 a, b, c, d;

u32 xorshift128()
{
    u32 t = d;
    u32 s = a;
    d = c;
    c = b;
    b = s;
    t ^= t << 11;
    t ^= t >> 8;
    s ^= s >> 19;
    a = t ^ s;
    return a;
}
```

▷ **Question 21 :** Démontrer que la fonction `xorshift128+`, PRG par défaut dans l'implémentation de JavaScript dans Chrome, Firefox, Safari, n'est pas de qualité cryptographique (nonobstant le fait que sa graine a une taille constante).

```
uint64_t state0, state1;
uint64_t xorshift128plus() {
        uint64_t s1 = state0, s0 = state1;
        state0 = s0;
        s1 ^= s1 << 23;
        s1 ^= s1 >> 17;
        s1 ^= s0;
        s1 ^= s0 >> 26;
        state1 = s1;
        return state0 + state1;
}
```

Indice : que peut-on dire du bit de poids faible de l'addition ?

▷ **Question 22 :** Démontrer que la fonction `splitmix64`, décrite en 2015 par Sebastiano Vigna, n'est pas un bon PRG.

```
u64 x;
u64 splitmix64() {
        x += 0x9E3779B97F4A7C15;
        u64 z = x;
        z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
        z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
        return z ^ (z >> 31);
}
```

▷ **Question 23 :** Le *Mersenne Twister* est un PRG populaire (par défaut dans Python et PHP). Il repose sur un tableau A de 624 entiers 32 bits. Pour sortir 32 bits pseudo-aléatoires : renvoyer $f(A[i])$ puis incrémenter i. Si $i = 624$, alors le tableau est rafraîchi avec une procédure qui n'est pas décrite ici.

```
def f(x):
        y ^= y >> 11
        y ^= (y << 7) & 2636928640
        y ^= (y << 15) & 4022730752
        y ^= y >> 18
        return y
```

Justifier que ce n'est pas un bon PRG.

▷ **Question 24 :** `xoshiro256**` est un autre PRG récent proposé (en 2019) par Vigna.

```
u64 rotl(u64 x, int k) {
        return (x << k) | (x >> (64 - k));
}
u64 s[4];
u64 next() {
        u64 result = rotl(s[1] * 5, 7) * 9;
        u64 t = s[1] << 17;
        s[2] ^= s[0];
        s[3] ^= s[1];
        s[1] ^= s[2];
        s[0] ^= s[3];
        s[2] ^= t;
        s[3] = rotl(s[3], 45);
        return result;
}
```

Une autre version, `xoshiro256++` est obtenu avec `result = rotl(s[0] + s[3], 23) + s[0];` tandis que `xoshiro256+` est fait avec `s[0] + s[3];`.

Parmis ces trois-là, deux sont manifestement de mauvais PRG, tandis que pour le troisième, il suffit probablement d'y réfléchir un peu plus. Qui est qui ?

5.3.3 Retour sur les générateurs congruentiels linéaires

▷ **Question 25 :** Démontrer que les générateurs congruentiels linéaires ne sont pas de bons PRG, même si tous les paramètres font partie de la graine et sont donc secrets.

Rappel : la séquence en sortie est $X_{i+1} = aX_i + b \bmod m$, et la graine est (X_0, a, b, m)

Indice °1 : comme faire pour éliminer b ?

Indice °2 : comme faire pour obtenir un multiple de m ?

▷ **Question 26 :** Refaire l'analyse de l'attaque des générateurs linéaires congruentiels tronqués en supposant qu'on ne peut résoudre CVP qu'approximativement avec un facteur d'approximation exponentiel (par exemple $\gamma = 2^{n/2}$).

5.3.4 Variante de LCG tronqué

On s'intéresse aux générateurs linéaires congruentiels tronqués *dans l'autre sens*, définis par :

$$X_{i+1} \leftarrow aX_i \bmod m \qquad \text{et} \qquad Y_i \leftarrow X_i \bmod k$$

On suppose que a, k, m sont connus et que $PGCD(k, m) = 1$ (par exemple, $m = 2^{61} - 1$ et $k = 256$). La graine du PRNG est X_0, et la séquence de ses sorties est $Y_1, Y_2, \ldots$ (c.a.d. on ne récupère que les bits de poids faible). S'agit-il de bons PRGs ?

▷ **Question 27 :** Combien de sorties faut-il observer au minimum pour pouvoir reconstituer la graine ?

▷ **Question 28 :** Justifier qu'il existe des entiers positifs ou nuls λ_i, μ_i satisfaisant :

$$Y_i = a^i X_0 - \lambda_i m - \mu_i k, \qquad \text{avec} \quad 0 \leq \mu_i < \frac{m}{k}. \tag{♠}$$

▷ **Question 29 :** Supposons qu'on nous révèle un indice i et une valeur de μ_i satisfaisant (♠). Cela permet-il de casser le PRG ?

▷ **Question 30 :** Au vu de la question précédente, il suffit de résoudre (♠). On considère la matrice :

$$B = \begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} \\ & m & & & \\ & & m & & \\ & & & \ddots & \\ & & & & m \\ k & & & & \\ & k & & & \\ & & k & & \\ & & & \ddots & \\ & & & & k \end{pmatrix}$$

Justifier qu'une solution de (♠) est en particulier une solution de $\mathbf{x}B = (Y_1, \dots, Y_n)$. Expliquer pourquoi cette observation ne permet *pas* de casser le PRG.

▷ **Question 31 :** Pour s'en tirer, il faut prendre en compte la borne sur la taille des μ_i. On considère pour cela le réseau de dimension $2n$ engendré par les lignes de :

$$G = \begin{pmatrix} 1 & a & a^2 & \dots & a^{n-1} & & & & \\ & m & & & & & & & \\ & & m & & & & & & \\ & & & \ddots & & & & & \\ & & & & m & & & & \\ k & & & & & 1 & & & \\ & k & & & & & 1 & & \\ & & k & & & & & 1 & \\ & & & \ddots & & & & & \ddots \\ & & & & k & & & & 1 \end{pmatrix}$$

Montrer qu'une solution de (♠) correspond à un vecteur du réseau « proche » d'un vecteur qu'on peut calculer à partir de la sortie du PRG. Quelle est la distance entre les deux ?

▷ **Question 32 :** Exhiber un vecteur particulièrement court dans le réseau. Comment sa taille se compare à ce que suggère l'heuristique gaussienne ?

▷ **Question 33 :** On tente de résoudre (♠), donc de casser le PRG, en résolvant l'instance de CVP suggérée par la question 5.31. Est-ce que ça va marcher (et pourquoi ?) ?

▷ **Question 34 :** Pour faire fonctionner l'idée du CVP malgré tout, il faudrait pouvoir modifier le réseau de sorte que 1) les vecteurs courts qui nous embêtent disparaissent et 1) que la distance entre les deux vecteurs qui nous intéressent ne change pas. On peut obtenir ceci en multipliant les n colonnes de gauche par une constante U.

Quelle est la bonne valeur de U ? Comment l'attaque fonctionne-t-elle alors ?