Université Paris 8 Année 2023–2024

Master 2 Mathématiques et applications

Parcours ACC (Arithmétique, Codage et Cryptologie)

Algorithmes Arithmétiques

Notes de cours

Julien Lavauzelle
julien.lavauzelle@univ-paris8.fr

11 octobre 2023

Table des matières

Quelques références utiles

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013

[BCG+17] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy, and Éric Schost. *Algorithmes Efficaces en Calcul Formel*. Frédéric Chyzak (auto-édit.), 2017

[LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications, 2nd edition*. Cambridge University Press, 1994

Chapitre 1

Introduction

Ce cours a pour objet l'étude d'**algorithmes avancés**, utilisés particulièrement en cryptographie. Ces algorithmes sont pour la plupart de nature **arithmétique**, dans le sens où ils manipulent ou bien des entiers relatifs et des rationnels, ou bien des éléments d'un corps fini.

L'approche de ces algorithmes sera à la fois théorique et pratique. Autant que possible, nous démontrerons la validité et nous donnerons la complexité théorique des algorithmes présentés. Puis, dans des exercices qui dépassent le cadre de ce document, nous expérimenterons leur implantation pratique.

Nous supposerons connus [1] certains algorithmes plus « élémentaires », non pas dans le sens où ils sont plus simples, mais dans le sens où ils résolvent des problèmes arithmétiques plus fondamentaux. Citons par exemple :

les techniques de multiplication rapide, à la Karatsuba ou à la Toom–Cook ;
les algorithmes de division rapide (binaire, Knuth, itérations de Newton) ;
les algorithmes d'exponentiation rapide ;
l'utilisation efficace du théorème des restes chinois ;
les tests de primalité (méthodes par crible, tests de Fermat, Euler, Miller–Rabin) ;
des algorithmes de génération de nombres pseudo-aléatoires.

Dans ce cours, nous nous focaliserons principalement sur 5 problèmes.

1. L'**algèbre linéaire rapide**, avec comme application la résolution de systèmes linéaires très creux.
2. L'**extraction de racines carrées** (et par extension, racines m-èmes), que ce soit dans les entiers naturels, les corps finis ou l'anneau $\mathbb{Z}/n\mathbb{Z}$.
3. La **factorisation de polynômes**, avec des coefficients dans $\mathbb{F}_q$, $\mathbb{Z}$ ou $\mathbb{Q}$.
4. La **factorisation d'entiers**, pour laquelle nous verrons des algorithmes génériques et d'autres spéciaux (*i.e.* efficaces seulement sur certains types d'instances).
5. Le calcul de **logarithmes discrets**, dans un groupe cyclique générique ou dans $\mathbb{F}_p^\times$.

D'autres notions pourront être abordées en fin de cours, comme la **recherche de vecteurs courts** dans un réseau euclidien, ou le **calcul de bases de Gröbner** par exemple.

1. ces algorithmes devraient avoir été étudiés lors du cours d'Algorithmes arithmétiques de M1

Chapitre 2

Algèbre linéaire

La résolution de nombreux problèmes arithmétiques comporte une phase dite d'*algèbre linéaire*, c'est-à-dire de résolution d'un système d'équations linéaires. En cryptanalyse, on la retrouve notamment dans les meilleurs algorithmes de factorisation d'entiers ou de polynômes, et de calcul de logarithme discret (voir chapitres suivants).

Dans ce chapitre, nous rappellerons la méthode générique de résolution par *élimination gaussienne*, puis nous étudierons l'algorithme de Wiedemann, particulièrement efficace lorsque les équations du système comportent une proportion importante de coefficients nuls.

2.1 Rappels

Dans tout le chapitre, $\mathbb{F}$ désigne un corps dont les éléments sont représentables sur une machine, et pour lequel on peut effectuer efficacement les opérations élémentaires (addition, multiplication, opposé, inverse, test à zéro, etc.). On dit que $\mathbb{F}$ est un *corps effectif*.

Les corps effectifs utilisés en pratique sont par exemple les corps finis $\mathbb{F}_q$, ou le corps $\mathbb{Q}$ des rationnels et ses extensions algébriques.

Notations. Les n-uplets et les vecteurs sont généralement représentés par un caractère gras et en minuscule : $\boldsymbol{a} \in \mathbb{Z}^n$ par exemple. On assimile les vecteurs à des matrices colonnes.

On note $\mathbb{F}^{n\times m}$ l'ensemble des matrices de taille $n \times m$. Généralement, on représente une matrice en gras et en majuscules. La i-ème ligne d'une matrice $\boldsymbol{A}$ est génériquement notée $\boldsymbol{A}_{i,*}$. Sa j-ème colonne est notée $\boldsymbol{A}_{*,j}$.

2.1.1 Le problème de la résolution de systèmes linéaires

Définition 2.1

Un système d'équations linéaires sur $\mathbb{F}$ est la donnée d'une matrice $\boldsymbol{A} \in \mathbb{F}^{m\times n}$ et d'un vecteur $\boldsymbol{b} \in \mathbb{F}^m$. Résoudre le système consiste à trouver l'ensemble des vecteurs $\boldsymbol{x} \in \mathbb{F}^n$ tels que $\boldsymbol{Ax} = \boldsymbol{b}$.

On rappelle le résultat fondamental décrivant la structure algébrique des solutions d'un système d'équations linéaires.

Proposition 2.2

L'ensemble des solutions d'un système d'équations linéaires $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ est
- *ou bien l'ensemble vide,*
- *ou bien un sous-espace affine $\mathcal{A}$ de $\mathbb{F}^n$, auquel cas $\mathcal{A}$ se décompose comme $\boldsymbol{x}^{(0)} + \mathcal{V}$ où $\boldsymbol{x}^{(0)}$ est une* solution particulière *du système et $\mathcal{V} = \ker \boldsymbol{A}$ est le noyau à droite de $\boldsymbol{A}$.*

Exemple 2.3

Sur $\mathbb{Q}$, on considère la matrice de taille 3×3 :

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & -1 & 0 \\ 1 & 3 & 4 \end{pmatrix}.$$

Si $\boldsymbol{b} = \begin{pmatrix} 6 \\ 0 \\ 8 \end{pmatrix}$, alors le système $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ admet une infinité de solutions. Ce sont les éléments de la forme

$$\begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} + \lambda \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}, \quad \lambda \in \mathbb{Q}.$$

Si $\boldsymbol{b} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$, alors il n'y a aucune solution au système $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

En général le nombre d'équations m et le nombre d'inconnues n ne sont pas égaux. Néanmoins, on peut se ramener au cas $n = m$ sans changer l'espace des solutions :

(i) en ajoutant $n - m$ lignes nulles à $\boldsymbol{A}$ et $n - m$ coefficients nuls à $\boldsymbol{b}$, dans le cas où $n > m$;
(ii) en ajoutant $m - n$ colonnes nulles et $\boldsymbol{A}$ et $m - n$ « inconnues virtuelles nulles », dans le cas où $n < m$.

Dans toute la suite, on supposera donc que $n = m$ et que la matrice $\boldsymbol{A}$ n'est pas nécessairement de rang plein. Notons qu'en pratique, il existe des algorithmes spécifiquement efficaces pour les cas où la matrice $\boldsymbol{A}$ est fortement déséquilibrée ($n \gg m$ ou $n \ll m$).

2.1.2 Le cas simple de la forme échelonnée

Étudions d'abord un cas simple, pour lequel la matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ a une structure facilitant la résolution du système.

Définition 2.4 (*Matrice sous forme échelonnée*)

On dit qu'une matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$" est sous forme (triangulaire supérieure) échelonnée si :
1. *toutes les lignes nulles de $\boldsymbol{A}$ sont placées en bas de la matrice ;*
2. *le nombre de zéros de chaque ligne de $\boldsymbol{A}$ forme une séquence strictement croissante (jusqu'à atteindre éventuellement n).*

Une matrice triangulaire supérieure échelonnée a par exemple la forme suivante :

$$\boldsymbol{A} = \begin{array}{|c|} \hline \times \\ \times \quad (*) \\ \quad \times \\ \quad \times \\ \quad\quad \times \\ \boldsymbol{0} \\ \hline \end{array}.$$

Apportons quelques précisions sur cette illustration : la zone contenant le symbole $(*)$ est constituée d'éléments quelconques de $\mathbb{F}$, excepté pour les positions indiquées par le symbole $\times$ qui représente nécessairement un élément non-nul de $\mathbb{F}$. Ces derniers éléments sont appelés des *pivots*. On dit que la matrice est *échelonnée réduite* si ses pivots sont tous égaux à 1.

Calcul du noyau. Lorsque la matrice $\boldsymbol{A}$ est échelonnée, on peut déterminer le noyau à droite de $\boldsymbol{A}$ grâce à l'Algorithme 1. L'idée est de transformer $\boldsymbol{A}$ en une matrice diagonale $\boldsymbol{D}$ par des opérations sur ses colonnes, autrement dit de calculer une matrice triangulaire supérieure $\boldsymbol{T}$ telle que $\boldsymbol{AT} = \boldsymbol{D}$. Le noyau de $\boldsymbol{D}$ est alors facile à calculer : ce sont les vecteurs $\boldsymbol{e}_i$ de la base canonique tels que $D_{i,i} = 0$. En déduit qu'une base du noyau de $\boldsymbol{A}$ est formée des vecteurs $\boldsymbol{Te}_i$ pour ces même indices i.

Algorithme 1 : Calcul du noyau à droite d'une matrice échelonnée

Entrée : Une matrice $\boldsymbol{A} \in \mathbb{F}^{n\times n}$ échelonnée.
Sortie : Une base du noyau à droite de $\boldsymbol{A}$.

```
T ← I_n
Calculer ℓ = max{i ∈ [1,n], A_{i,*} ≠ 0}.
Pour tout i allant de 1 à ℓ faire
    Calculer s tel que le pivot est en position (i,s).
    Pour tout j allant de s+1 à n faire
        Calculer α ← A_{i,j} / A_{i,s}
        Remplacer la colonne T_{*,j} par T_{*,j} − αT_{*,s}.
Retourner les colonnes T_{*,j} de T telles que j n'est pas la deuxième coordonnée d'un pivot de A.
```

Calcul d'une solution particulière. Ensuite, l'Algorithme 2 permet de calculer une solution particulière du système $\boldsymbol{Ax} = \boldsymbol{b}$. Dans les premières étapes (1–6), on traite les cas particuliers (aucune solution, solution triviale). Puis, l'étape itérative (7–10) permet de construire efficacement une solution particulière du système.

Algorithme 2 : Algorithme de résolution d'un système linéaire échelonné

Entrée : Une matrice $\boldsymbol{A} \in \mathbb{F}^{n\times n}$ sous forme échelonnée, et un vecteur $\boldsymbol{b} \in \mathbb{F}^n$.
Sortie : Une solution particulière de $\boldsymbol{Ax} = \boldsymbol{b}$.

```
1  Si A = 0 et b ≠ 0
2      retourner « aucune solution »
3  Si A = 0 et b = 0
4      retourner 0
5  Calculer ℓ = max{i ∈ [1,n], A_{i,*} ≠ 0}.
6  S'il existe i > ℓ tel que b_i ≠ 0, alors retourner « aucune solution ».
7  Assigner x = (x_1, ..., x_n) ← 0.
8  Pour tout i allant de ℓ à 1 faire
9      Calculer la position (i,s) du pivot.
10     Calculer x_s ← (1/A_{i,s}) (b_i − Σ_{j=s+1}^{n} A_{i,j} x_j).
11 Retourner x.
```

Remarque 2.5

On peut vérifier que la complexité de l'Algorithme 1 est en $O(n^3)$ opérations sur $\mathbb{F}$ tandis que celle de l'Algorithme 2 est en $O(n^2)$.

2.1.3 Algorithme d'élimination Gaussienne

Pour résoudre le problème dans le cas général, on cherche à transformer le système en un système sous forme échelonnée. La méthode est connue sous le nom d'éliminiation Gaussienne ou d'algorithme de Gauss–Jordan si l'on obtient une matrice sous forme échelonnée réduite. Dans la présentation donnée par l'Algorithme 3, on obtient même un système *triangulaire*, c'est-à-dire dont les pivots sont placés sur la diagonale.

Algorithme 3 : Algorithme d'élimination Gaussienne

Entrée : Une matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$
Sortie : Un système échelonné équivalent à $\boldsymbol{Ax} = \boldsymbol{b}$, c'est-à-dire trois matrices $\boldsymbol{U}$, $\boldsymbol{T}$ et $\boldsymbol{P}$ telles que $\boldsymbol{UAP} = \boldsymbol{T}$ et $\boldsymbol{T}$ est échelonnée.

$\boldsymbol{T} \leftarrow \boldsymbol{A}$, $\boldsymbol{U} \leftarrow \boldsymbol{I}_n$, $\boldsymbol{P} \leftarrow \boldsymbol{I}_n$
$\ell \leftarrow n$, $i \leftarrow 1$
Tant que $i \leq \ell$ **faire**
  Rechercher un élément inversible (pivot) dans la i-ème ligne de $\boldsymbol{T}$.
  Si *aucun pivot n'a été trouvé :*
    Échanger les lignes ℓ et i dans les matrices $\boldsymbol{T}$ et $\boldsymbol{U}$.
    $\ell \leftarrow \ell - 1$
  Sinon
    Noter j l'indice de colonne du pivot.
    Échanger les colonnes i et j dans les matrices $\boldsymbol{T}$ et $\boldsymbol{P}$.
    Pour tout k *allant de* $i+1$ *à* ℓ **faire**
      Calculer $\alpha = \frac{T_{k,i}}{T_{i,i}}$.
      Remplacer la ligne $\boldsymbol{T}_{k,*}$ de $\boldsymbol{T}$ par $\boldsymbol{T}_{k,*} - \alpha \boldsymbol{T}_{i,*}$.
      Remplacer la ligne $\boldsymbol{U}_{k,*}$ de $\boldsymbol{U}$ par $\boldsymbol{U}_{k,*} - \alpha \boldsymbol{U}_{i,*}$.
    $i \leftarrow i + 1$.
Retourner $\boldsymbol{T}, \boldsymbol{U}, \boldsymbol{P}$.

On doit maintenant comprendre comment évolue l'espace des solutions du système $\boldsymbol{Ax} = \boldsymbol{b}$ lors de l'exécution de l'Algorithme 3. Pour cela, on introduit des matrices élémentaires qui représentent les transformations sur les lignes et les colonnes effectuées par l'élimination Gaussienne.

Définition 2.6

On définit les matrices élémentaires suivantes :
— *la matrice élémentaire générique*

$$\boldsymbol{M}_{i,j}\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) := \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & a & & b & \\ & & & 1 & & \\ & & & & 1 & \\ & & c & & d & \\ & & & & & 1 \end{bmatrix}$$

où a est en position (i,i), b est en position (i,j), c est en position (j,i) et d est en position (j,j) ;
— *la matrice de permutation $\boldsymbol{P}_{i,j} = \boldsymbol{M}_{i,j}\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$;*
— *la matrice l'élimination $\boldsymbol{E}_{i,j}(\alpha) := \boldsymbol{M}_{i,j}\left(\begin{smallmatrix} 1 & \alpha \\ 0 & 1 \end{smallmatrix}\right)$ pour $\alpha \in \mathbb{F}$.*

L'action à droite $\boldsymbol{A} \mapsto \boldsymbol{A}\boldsymbol{P}_{i,j}$ d'une matrice de permutation élémentaire $\boldsymbol{P}_{i,j}$ a pour effet d'échanger les colonnes i et j de $\boldsymbol{A}$. Similairement, son action à gauche ($\boldsymbol{A} \mapsto \boldsymbol{P}_{i,j}\boldsymbol{A}$) permute les lignes de $\boldsymbol{A}$.

En ce qui concerne la matrice d'élimination, l'action à droite $\boldsymbol{A} \mapsto \boldsymbol{A}\boldsymbol{E}_{i,j}(\alpha)$ remplace la j-ème colonne $\boldsymbol{A}_{*,j}$ de $\boldsymbol{A}$ par $\boldsymbol{A}_{*,j} + \alpha\boldsymbol{A}_{*,i}$. De même, la transformation $\boldsymbol{A} \mapsto \boldsymbol{E}_{i,j}(\alpha)\boldsymbol{A}$ remplace la i-ème ligne $\boldsymbol{A}_{i,*}$ de $\boldsymbol{A}$ par $\boldsymbol{A}_{i,*} + \alpha\boldsymbol{A}_{j,*}$.

On peut donc modéliser toutes les opérations effectuées dans l'Algorithme 3 par des produits impliquant des matrices élémentaires. Notons que sur $\boldsymbol{P}$, seuls des échanges de colonnes sont effectués ; ainsi $\boldsymbol{P}$ est une matrice de permutation (non-élémentaire) : chaque ligne et chaque colonne de $\boldsymbol{P}$ contient un unique coefficient non-nul égal à 1. En pratique, cela permet d'avoir un stockage très compact de la matrice $\boldsymbol{P}$.

L'Algorithme 3 produit une matrice $\boldsymbol{T}$ pour laquelle la résolution d'un système linéaire est plus aisée (voir section précédente). On doit maintenant faire le lien entre les solutions de ce nouveau système linéaire échelonné, et les solutions du système impliquant $\boldsymbol{A}$. Pour cela, nous avons besoin des deux résultats suivants dont les preuves sont élémentaires.

Lemme 2.7

Soit $\boldsymbol{P} \in \mathbb{F}^{n\times n}$ une matrice de permutation et $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ un système linéaire. Alors, $\boldsymbol{x}^{(0)}$ est solution de $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ si et seulement si $\boldsymbol{y}^{(0)} := \boldsymbol{P}^{-1}\boldsymbol{x}^{(0)}$ est solution du système $(\boldsymbol{A}\boldsymbol{P})\boldsymbol{y} = \boldsymbol{b}$.

Lemme 2.8

Soit $\boldsymbol{U} \in \mathbb{F}^{n\times n}$ une matrice inversible et $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ un système linéaire. Alors, $\boldsymbol{x}^{(0)}$ est solution de $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ si et seulement si $\boldsymbol{x}^{(0)}$ est solution du système $(\boldsymbol{U}\boldsymbol{A})\boldsymbol{x} = \boldsymbol{U}\boldsymbol{b}$.

Le Lemme 2.7 indique que la permutation des colonnes effectuée lors de l'étape 10 de l'Algorithme 3 se transcrit en une permutation inversée des coordonnées des solutions du système. Le Lemme 2.8 assure que les étapes 6 et 13 de l'Algorithme 3 préservent l'ensemble des solutions du système.

Donnons maintenant la méthode générique pour résoudre le système linéaire $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$:

1. Procéder à une élimination gaussienne, c'est-à-dire calculer $\boldsymbol{U}$, $\boldsymbol{T}$ et $\boldsymbol{P}$ tels que $\boldsymbol{U}\boldsymbol{A}\boldsymbol{P} = \boldsymbol{T}$.
2. Calculer une solution particulière $\boldsymbol{y}^{(0)}$ de $\boldsymbol{T}\boldsymbol{y} = \boldsymbol{U}\boldsymbol{b}$.
   Si elle n'existe pas, **terminer l'algorithme** avec « pas de solution ».
3. Calculer $\mathcal{K} = \ker(\boldsymbol{T})$ le noyau à droite de $\boldsymbol{T}$.
4. **Retourner** une description de l'espace affine $\boldsymbol{P}(\boldsymbol{y}^{(0)} + \mathcal{K}) = \{\boldsymbol{P}(\boldsymbol{y}^{(0)} + \boldsymbol{u}), \boldsymbol{u} \in \mathcal{K}\}$.

En collectant tous les résultats précédents, on obtient un algorithme permettant de calculer les solutions de n'importe quel système d'équations linéaires.

Proposition 2.9

L'algorithme d'élimination Gaussienne, couplé à la résolution d'un système échelonné, permet de résoudre tout système linéaire carré ($n = m$) sur $\mathbb{F}$ en $O(n^3)$ opérations élémentaires sur $\mathbb{F}$.

Démonstration : Pour la validité de la méthode, voir la discussion précédente.

Concernant sa complexité, on peut vérifier que les Algorithmes 1 et 3 sont les plus coûteux et nécessitent $O(n^3)$ opérations. En effet, par exemple l'étape 13 de l'Algorithme 3 est effectuée $O(n^2)$ fois (pour $i = 1, \ldots, n$ si $\boldsymbol{A}$ est de rang n, et pour $k = i+1, \ldots, n$), et elle coûte individuellement $O(n)$ opérations sur $\mathbb{F}$. Une analyse de complexité plus précise est laissée en exercice. ■

Dans cette section, nous avons donc vu qu'il existe une méthode générique pour résoudre un système d'équations liénaires en temps cubique en le nombre d'équations et d'inconnues. Dans le but d'améliorer cette complexité dans le contexte de systèmes dite *creux*, nous allons nous intéresser à la notion de suite récurrentes linéaires.

2.2 Suites récurrentes linéaires

Les suites récurrentes linéaires sont des objets courants en informatique. En cryptographie, on utilise notamment des LFSR (*linear feedback shift register*, registre à décalage rebouclé linéairement) pour engendrer des séquences de nombres pseudo-aléatoires. Dans la suite, nous allons également voir que ces suites permettent d'accélérer la résolution de systèmes linéaires dont les équations comportent peu de coefficients non-nuls.

2.2.1 Premières définitions

Nous nous plaçons d'abord dans le cadre des suites *scalaires*, c'est-à-dire dont les termes appartiennent à un corps $\mathbb{F}$.

Définition 2.10 (*suite récurrente linéaire*)

Une suite $\boldsymbol{u} = (u_n)_{n\in\mathbb{N}} \in \mathbb{F}^{\mathbb{N}}$ *est dite* récurrente linéaire *s'il existe deux entiers* $0 \leq D \leq L$ *et* D *coefficients* $(c_1, \dots, c_D) \in \mathbb{F}^D$ *tels que :*

$$\forall n \geq L, \quad u_n + \sum_{i=1}^{D} c_i u_{n-i} = 0\,. \tag{2.1}$$

Le plus petit D *qui satisfait* (2.1) *est appelé* ordre *de la suite. Les* L *premiers éléments* $u_0, \dots, u_{L-1}$ *de la suite sont appélés* termes initiaux.

Exemple 2.11

Sur $\mathbb{R}$*, la suite* $\boldsymbol{u} = (-1, 1, -1, 1, -1, 1, -1, 1, \dots)$ *est récurrente linéaire. En effet, on peut l'écrire*

$$u_0 = -1 \quad \textit{et} \quad u_n + u_{n-1} = 0, \quad \forall n \geq 1\,.$$

Son ordre est donc $D = 1$ *(car seule la suite nulle à partir d'un certain rang a un ordre égal à* 0*), et la suite admet* $L = 1$ *terme initial, à savoir* $u_0 = -1$*.*

Exemple 2.12

Sur le corps fini $\mathbb{F}_2$*, la suite*

$$\boldsymbol{v} = (1, \underbrace{0, 1, 1, 1, 0, 1, 0}_{\textit{séquence périodique}}, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0 \dots)$$

est récurrente linéaire. En effet, on peut l'écrire

$$u_0 = 1,\ u_1 = 0, u_2 = 1, u_3 = 1 \quad \textit{et} \quad u_n + u_{n-1} + u_{n-3} = 0, \quad \forall n \geq 4\,.$$

Son ordre est donc ≤ 3 *(bien qu'il y ait* 4 *coefficients initiaux). On pourra vérifier par la suite que l'ordre de la suite est exactement* 3*.*

Description par une série formelle. Toute suite $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ peut être écrite comme une *série formelle*, en associant à $\boldsymbol{u}$ la série

$$U(X) := \sum_{n\in\mathbb{N}} u_n X^n .$$

On note $\mathbb{F}[\![X]\!]$ l'algèbre des séries formelles en X. On rappelle que pour $U,V \in \mathbb{F}[\![X]\!]$, on définit la somme et le produit de séries comme :

$$(U+V)(X) := \sum_{n\in\mathbb{N}} (u_n + v_n)X^n,$$
$$(U\cdot V)(X) := \sum_{n\in\mathbb{N}} \left(\sum_{i=0}^{n} u_i v_{n-i}\right) X^n .$$

Observons qu'un polynôme $P \in \mathbb{F}[X]$ peut être vu comme une série formelle, dont les coefficients sont nuls à partir d'un certain rang.

Lemme 2.13

Soit $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ une suite récurrente linéaire d'ordre D avec L termes initiaux. La série formelle $U \in \mathbb{F}[\![X]\!]$ de $\boldsymbol{u}$ satisfait $UP = Q$, où P et Q sont des polynômes tels que $\deg(P) \le D$ et $\deg(Q) < L$.

Démonstration : Par définition de la suite $\boldsymbol{u}$, il existe des coefficients $c_1, \dots, c_d$ tels que

$$u_n + \sum_{i=1}^{d} c_i u_{n-1} = 0, \quad \forall n \ge L .$$

Posons $P(X) := \sum_{i=0}^{d} c_i X^i$ avec $c_0 = 1$. Alors :

$$PU = \sum_{n\in\mathbb{N}} \left(\sum_{i=0}^{n} c_i u_{n-i}\right) X^n$$

et le coefficient de degré n de PU est donc nul pour tout $n \ge L$. Autrement dit, PU est un polynôme de degré $\le L-1$. ∎

Le résultat précédent permet d'exprimer la série formelle d'une suite récurrente linéaire comme une fraction rationnelle $\frac{Q}{P} \in \mathbb{F}(X)$.

Exemple 2.14

Sur le corps $\mathbb{F}_3$, on considère la suite $\boldsymbol{u}$ définie par

$$u_0 = 1,\ u_1 = 0,\ u_2 = 0,\ u_3 = 1, \quad \text{et} \quad u_n = u_{n-1} + 2u_{n-2}, \forall n \ge 4 .$$

Les 20 premiers termes de cette suite sont :

$$(1,0,0,1,1,0,2,2,0,1,1,0,2,2,0,1,1,0,2,2,\dots) .$$

Le polynôme P associé à la suite est $P(X) = 1 - X - 2X^2 = 1 + 2X + X^2$, et on peut calculer

$$\begin{aligned} Q := PU &= c_0u_0 + (c_1u_0 + c_0u_1)X + (c_2u_0 + c_1u_1 + c_0u_2)X^2 + (c_2u_1 + c_1u_2 + c_0u_3)X^3 \\ &= 1 + 2X + 2X^2 + X^3 . \end{aligned}$$

Puis,

$$U(X) = \frac{Q(X)}{P(X)} = \frac{1 + 2X + X^2 + X^3}{1 + 2X + X^2} .$$

Définition 2.15

Dans l'écriture $UP = Q$ du Lemme 2.13, le polynôme P est appelé un polynôme de connexion *de la suite $\boldsymbol{u}$ représentée par $U(X)$.*

A priori, le polynôme de connexion d'une suite récurrente linéaire n'est pas unique. Néanmoins, l'ensemble des polynômes de connexion d'une même suite possède une structure algébrique non-triviale.

Lemme 2.16

L'ensemble des polynômes de connexion d'une suite $\boldsymbol{u}$ forme un idéal de $\mathbb{F}[X]$. En particulier, il existe un polynôme de connexion P de degré minimal, tel que $P(0) = 1$ et tel que tout polynôme de connexion est un multiple de P. Ce polynôme est appelé polynôme de connexion minimal *de la suite $\boldsymbol{u}$.*

Démonstration : En exercice. ■

Remarque 2.17

Le Lemme 2.13 assure que toute suite linéaire récurrente peut se modéliser comme une fraction rationnelle. Réciproquement, toute fraction rationnelle $F(X) = \frac{Q(X)}{P(X)} \in \mathbb{F}(X)$ avec $P(0) = 1$ définit une suite récurrente linéaire $\boldsymbol{u}$. Le polynôme P au dénominateur définit une relation de récurrence régissant la suite (c'est un polynôme de connexion), et il reste ensuite à calculer les termes initiaux. Pour cela, on définit

$$F_0(X) := F(X) \quad \text{et} \quad F_{i+1}(X) = \frac{F_i(X) - F_i(0)}{X} \ \text{pour tout } i \geq 0\,.$$

Alors, les $L := \deg(Q) - 1$ premiers termes de la suite sont $F_0(0), \ldots, F_{L-1}(0)$.

Dans la suite de cette section, notre objectif va être de répondre aux questions suivantes :

1. Étant donnée une suite récurrente $\boldsymbol{u}$, comment retrouver efficacement son polynôme de connexion minimal ?
2. Combien de termes de la suite $\boldsymbol{u}$ sont-ils nécessaires pour obtenir ce polynôme ?

Nous verrons en Section 2.2.3 que l'algorithme de Berlekamp–Massey est une méthode itérative permettant de répondre à ces questions. L'idée est de construire de proche en proche les polynômes de connexion minimaux de sous-suites tronquées de $\boldsymbol{u}$.

Avant de décrire cet algorithme, nous avons besoin d'introduire le concept de registre à décalage, aussi couramment utilisé en informatique, par exemple en cryptographie symétrique.

2.2.2 Registres à décalage

Les registres à décalage permettent de représenter un flux de données engendré par une suite récurrente linéaire. En voici une définition formelle.

Définition 2.18 (*LFSR*)

*Un LFSR (*linear feedback shift register, *registre à décalage rebouclé linéairement) est la donnée d'un polynôme $P \in \mathbb{F}[X]$ tel que $P(0) = 1$, et d'un entier $L \geq \deg(P)$. L'entier L est appelé la* dimension *du LFSR.*

Usuellement, les LFSR sont présentés dans le cas binaire ($\mathbb{F} = \mathbb{F}_2$) ou plus rarement dans le cas où $\mathbb{F}$ est un corps fini quelconque.

Définition 2.19 (*suite engendrée par un LFSR*)

Soit $\mathcal{L} = (P, L)$ un LFSR, avec $P(X) = 1 + \sum_{i=1}^{D} c_i X^i$. La suite engendrée par $\mathcal{L}$ sur le registre initial *$(u_0, \ldots, u_{L-1}) \in \mathbb{F}^L$ est la suite $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$:*

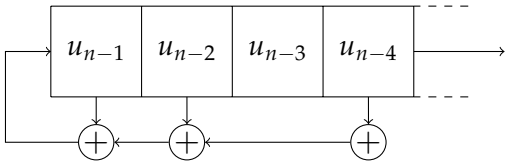
— *dont les L premiers termes sont $u_0, \ldots, u_{L-1}$,*

— *dont les termes suivants sont définis par la relation de récurrence*

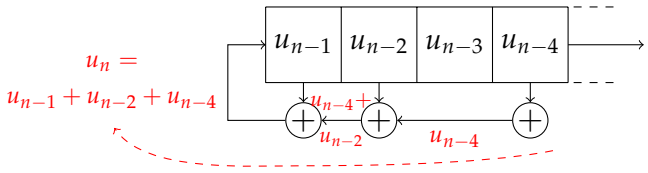
$$u_n + \sum_{i=1}^{D} c_i u_{n-i} = 0, \quad \forall n \geq L.$$

Remarquons que si (P, L) est un LFSR qui engendre une suite $\boldsymbol{u}$, alors P est un polynôme de connexion de $\boldsymbol{u}$.

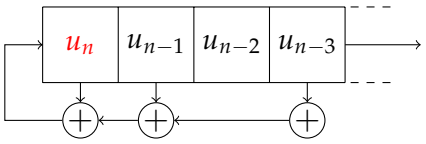
Représentation « circuit » d'un LFSR sur $\mathbb{F}_2$. Sur le corps fini $\mathbb{F}_2$, on utilise souvent une représentation des LFSR sous forme de circuit. Par exemple, le LFSR $(1 + X + X^2 + X^4, 4)$ est représenté comme suit :



L'idée de la représentation est la suivante : au polynôme $P(X) = 1 + X + X^2 + X^4$ est associé l'équation de récurrence $u_n = u_{n-1} + u_{n-2} + u_{n-4}$ pour tout $n \geq 4$. À chaque pas de temps n, quatre bits $u_{n-1}, u_{n-2}, u_{n-3}, u_{n-4}$ sont stockés dans le *registre*. Pour créer le nouveau bit u_n, on effectue la somme de certains des bits précédents, à savoir $u_{n-1}, u_{n-2}, u_{n-4}$ (cette somme est représentée dans le circuit par les portes $\oplus$), puis on décale le registre d'un cran vers la droite.



état à la fin de l'étape n



état au début de l'étape $n+1$

Dans l'exemple suivant, on peut observer qu'une suite peut être engendrée par des LFSR de polynômes différents et de dimensions différentes.

Exemple 2.20

La suite binaire $\boldsymbol{v} = (1010101010....) \in \mathbb{F}_2^{\mathbb{N}}$ peut être vue comme

$$v_0 = 1,\ v_1 = 0, \qquad v_n = v_{n-2} \quad \forall n \geq 2$$

ou comme

$$v_0 = 1,\ v_1 = 0,\ v_2 = 1, \qquad v_n = v_{n-1} + v_{n-2} + v_{n-3} \quad \forall n \geq 3$$

Dans le premier cas, la suite $\boldsymbol{v}$ est engendrée par le LFSR $(1 + X^2, 2)$ avec le registre initial $(1, 0)$. Dans le second, elle est engendrée par le LFSR $(1 + X + X^2 + X^3, 3)$ avec le registre initial $(1, 0, 1)$.

Comme pour le polynôme de connexion, on peut alors se demander s'il existe un LFSR de dimension minimale qui engendre une suite, et le cas échéant, comment le construire. Pour cela, intéressons-nous aux sous-suites constituées des premiers termes, aussi appelées sous-suites tronquées.

Suites tronquées. En pratique, nous n'avons pas accès à tous les termes d'une suite linéaire récurrente, mais seulement à ses premiers termes. Les suites rencontrées sont donc « tronquées » à un certain rang. Pour une suite $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ et un entier $k \geq 1$, on note

$$T_k(\boldsymbol{u}) \coloneqq (u_0, u_1, \ldots, u_{k-1}, 0, 0, \ldots, 0, \ldots) \in \mathbb{F}^{\mathbb{N}}$$

la suite où l'on remplace les termes d'indice $\geq k$ par 0. Cette suite sera souvent assimiliée au k-uplet de ses k permiers termes.

Remarque 2.21

Si $\boldsymbol{u}$ est une suite récurrente linéaire de série formelle $U \in \mathbb{F}[\![X]\!]$, alors la série formelle de $T_k(\boldsymbol{u})$ est le polynôme obtenu par la division euclienne de U par X^k.

Par la suite, on notera $\mathrm{rem}(A,B)$ le reste de la division euclidienne du polynôme A par le polynôme B.

Définition 2.22 (*suite engendrée sur un nombre fini de termes*)

Soient $\mathcal{L} = (P, L)$ un LFSR, avec $P(X) = 1 + \sum_{i=1}^{D} c_i X^i$, et $N \in \mathbb{N}$. On dit que $\mathcal{L}$ engendre une suite $\boldsymbol{u} \in \mathbb{F}^N$ sur N termes si

$$u_n + \sum_{i=1}^{D} c_i u_{n-i} = 0, \quad \forall L \leq n < N.$$

Lemme 2.23

Soit $\boldsymbol{u}$ une suite linéaire récurrente de série formelle U. Un LFSR (P, L) engendre $\boldsymbol{u}$ sur $N > L$ termes si et seulement si le reste de la division euclidienne du polynôme PU par X^N est de degré $< L$.

Démonstration : Notons $P(X) = \sum_{i=0}^{D} c_i X^i$ et effectuons la division euclidienne du polynôme PU par X^N : on a $PU = R + QX^N$ avec $\deg(R) < N$. Plus précisément,

$$R(X) = \sum_{n=0}^{N-1} \left(\sum_{i=0}^{n} c_i u_{n-i} \right) X^n .$$

Si (P, L) engendre $\boldsymbol{u}$ sur N termes, alors pour tout $L \leq n < N$ on a $u_n + \sum_{i=1}^{D} c_i u_{n-i} = 0$. Autrement dit, les coefficients des monômes de degré $\geq L$ de $R(X)$ sont nuls, donc $R(X)$ est de degré $< L$.
Réciproquement, si $\sum_{i+j=n} c_i u_{n-i} = 0$ pour tout $L \leq n < N$, alors on observe que (P, L) est bien un LFSR qui engendre $\boldsymbol{u}$ sur N termes : les L premiers termes peuvent être fixés à u_0, u_1, …, u_{L-1}, et les suivants satisfont la relation de récurrence donnée par P. ∎

Exemple 2.24

Soit $\boldsymbol{w} \in \mathbb{F}_2^{\mathbb{N}}$ de série formelle $W(X) = \frac{1+X+X^2}{1+X^4}$. On peut vérifier que les premiers termes de $\boldsymbol{w}$ sont :

$$\boldsymbol{w} = (1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, \ldots, \ldots)$$

Cette suite est complètement engendrée par le LFSR $\mathcal{L} = (1 + X^4, 4)$ *avec le registre initial* $(1, 1, 1, 0)$. *En particulier,* $\mathcal{L}$ *engendre* $\boldsymbol{w}$ *sur* k *termes pour tout* $k \in \mathbb{N}$.
Considérons maintenant le LFSR $\mathcal{L}' = (1 + X, 1)$. *Il engendre* $\boldsymbol{w}$ *sur ses trois premiers termes ; en effet* $w_2 = w_1 = w_0 = 1$, *autrement dit (on travaille modulo 2)* $w_2 + w_1 = 0$ *et* $w_1 + w_0 = 0$. *En revanche,* $\mathcal{L}'$ *n'engendre pas* $\boldsymbol{w}$ *sur 4 termes, car* $w_3 + w_2 = 1$.

Définition 2.25

On note $\mathcal{R}(\boldsymbol{u})$ *l'ensemble des LFSR qui engendrent une suite récurrente linéaire* $\boldsymbol{u}$, *et* $\mathcal{R}_k(\boldsymbol{u})$ *l'ensemble des LFSR qui engendrent* $\boldsymbol{u}$ *sur* k *termes.*

Lemme 2.26

Soit $\boldsymbol{u}$ *une suite récurrente linéaire. Alors, la suite* $(\mathcal{R}_k(\boldsymbol{u}))_{k\in\mathbb{N}}$ *est décroissante pour la relation d'inclusion, et constante et égale à* $\mathcal{R}(\boldsymbol{u})$ *à partir d'un certain rang.*

Démonstration : Laissée en exercice. ■

Définition 2.27 (*complexité linéaire, profil de complexité*)

Soit $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ *une suite quelconque et* $k \geq 1$ *un entier. La* complexité linéaire *de* $\boldsymbol{u}$ *à l'ordre* k *est la plus petite dimension* L *d'un LFSR* (P, L) *qui engendre* $\boldsymbol{u}$ *sur* k *termes. On la note*

$$\ell_k(\boldsymbol{u}) := \min\{L \in \mathbb{N} \mid \exists P \in \mathbb{F}[X], (P, L) \in \mathcal{R}_k(\boldsymbol{u})\}.$$

Par convention on pose $\ell_0(\boldsymbol{u}) := 0$.
On définit également le profil de complexité linéaire *de* $\boldsymbol{u}$ *comme la suite d'entiers* $\boldsymbol{\ell} := (\ell_k(\boldsymbol{u}))_{k\in\mathbb{N}}$.

On dit également que (P, L) est un LFSR *minimal* pour $\boldsymbol{u}$ sur k termes si $(P, L) \in \mathcal{R}_k(\boldsymbol{u})$ et $L = \ell_k(\boldsymbol{u})$.

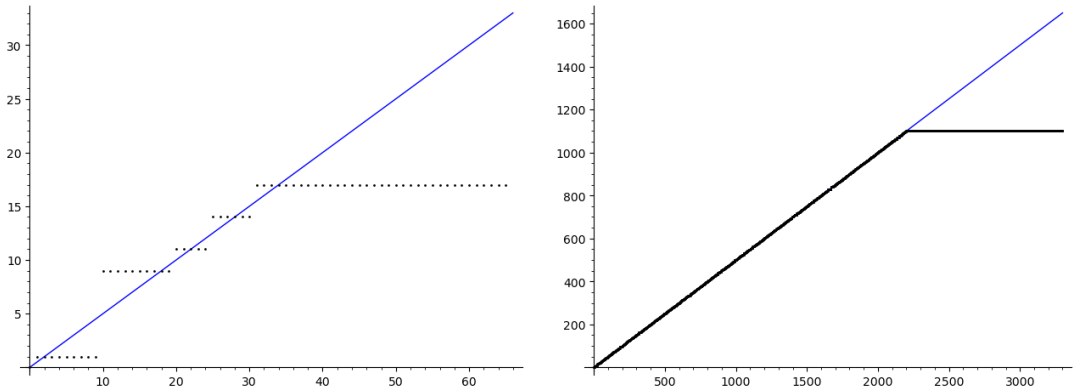


FIGURE 2.1 – Profils de complexité linéaire (en noir) de deux suites récurrentes linéaires d'ordre ≤ 22 (à gauche) et ≤ 1100 (à droite). En bleu, on a tracé la droite $k \mapsto k/2$.

Remarque 2.28

On observe que le profil de complexité linéaire $(\ell_k(\boldsymbol{u}))_k$ *est une suite croissante qui évolue par « sauts ». Par ailleurs, comme la suite* $(\mathcal{R}_k(\boldsymbol{u}))_k$ *est stationnaire, le profil l'est également : il atteint une valeur limite qui est la dimension minimale du registre qui permet d'engendre* $\boldsymbol{u}$. *Enfin,*

une analyse précise permet de montrer que, durant sa phase de croissance, le profil $(\ell_k(\boldsymbol{u}))_k$ croît approximativement comme $k/2$. Cela s'observe par exemple dans la Figure 2.1.

Exemple 2.29

Soit $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ une suite commençant par k zéros et telle que $u_k = 1$. Alors, on a $\ell_0(\boldsymbol{u}) = 0$ par convention, puis $\ell_i(\boldsymbol{u}) = 0$ pour tout $i \in \{1, \dots, k\}$. En effet, en posant $P_i(X) = 1$, on obtient la relation de récurrence linéaire d'ordre 0 donnée par $u_i = 0$.
Puis, on a ensuite $\ell_{k+1}(\boldsymbol{u}) = k + 1$, une nouvelle fois avec $P_k(X) = 1$ (par exemple), car $u_k \neq 0$ ne peut pas être calculé comme une combinaison linéaire de termes nuls.

Exemple 2.30

On considère la suite binaire $\boldsymbol{u}$ définie par ses premiers termes

$$\boldsymbol{u} = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, \dots).$$

Cherchons à déterminer les premiers termes du profil de complexité linéaire.
• *Ordre 0 : Par convention $\ell_0 = 0$.*
• *Ordre 1 : On a $\ell_1 = 1$, car il faut définir le premier terme de la suite.*
• *Ordre 2 : Pour le calcul de ℓ_2, on s'intéresse aux deux premiers termes $(u_0, u_1) = (1, 1)$. On observe qu'ils sont égaux, donc il suffit d'un seul terme initial $u_0 = 1$. On a donc $\ell_2 = 1$ et le terme suivant u_1 est engendré par le polynôme $P = 1 + X$.*
• *Ordre 3 : Pour le calcul de ℓ_3, on a $(u_0, u_1, u_2) = (1, 1, 1)$, donc le cas précédent se répète : $\ell_3 = 1$ avec le registre $u_0 = 1$ et le polynôme $P = 1 + X$.*
• *Ordre 4 : Le calcul de ℓ_4 diffère car on a $u_3 = 0 \neq u_2$. D'abord, notons qu'aucune relation de récurrence d'ordre 1 ne peut exprimer u_3 en fonction de u_2, il faut donc chercher une relation d'ordre au moins 2. On remarque que $P = 1 + X + X^2$ convient avec un registre de taille 3. Peut-on trouver un registre initial plus petit ? La réponse est non. En effet, il est impossible d'avoir un registre de taille 2 car les seules relations liant u_2 à u_1 et u_0 proviennent des polynômes $1 + X$ et $1 + X^2$, et aucun de ces polynômes ne permet d'engendrer u_3 à partir de u_2 et u_1. On a donc $\ell_4 = 3$.*

En poursuivant le calcul, on obtient les valeurs suivantes (les polynômes P n'étant pas nécessairement uniques) :

$\ell_0 = 0$	$(P, L) = (1, 0)$ *par convention*	$\ell_4 = 3$	$(P, L) = (1 + X + X^3, 3)$
$\ell_1 = 1$	$(P, L) = (1, 1)$	$\ell_5 = 3$	$(P, L) = (1 + X + X^3, 3)$
$\ell_2 = 1$	$(P, L) = (1 + X, 1)$	$\ell_6 = 3$	$(P, L) = (1 + X + X^2, 3)$
$\ell_3 = 1$	$(P, L) = (1 + X, 1)$	$\ell_7 = 3$	$(P, L) = (1 + X + X^2, 3)$

Le calcul à l'ordre 4 de l'exemple précédente nous montre que l'obtention d'un LFSR de dimenion minimale n'est pas triviale. Dans la prochaine section, nous allons présenter un algorithme qui permet d'effectuer cette tâche.

2.2.3 Algorithme de Berlekamp–Massey

Rappelons que nous cherchons à calculer le polynôme de connexion minimal d'une suite récurrente linéaire $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$. L'algorithme de Berlekamp–Massey permet d'obtenir ce polynôme par une méthode itérative : plus précisément, l'algorithme calcule une suite de LFSR $((P_k, L_k))_{k \in \mathbb{N}}$ telle que (P_k, L_k) est de dimension minimale dans $\mathcal{R}_k(\boldsymbol{u})$.

Avant de décrire l'algorithme, présentons quelques résultats théoriques.

Proposition 2.31

Soit $\mathcal{L} = (P, L)$ et $\mathcal{L}' = (P', L')$ deux LFSR et $k \geq L + L'$. Supposons que $\mathcal{L}$ et $\mathcal{L}'$ engendrent une même suite récurrente linéaire $\boldsymbol{u}$ sur k termes. Alors, il existe une suite récurrente $\boldsymbol{v}$ telle que $\mathcal{L}$ et $\mathcal{L}'$ engendrent $\boldsymbol{v}$ complètement.

Démonstration : Soit U le polynôme associé à la suite tronquée $T_k(\boldsymbol{u})$. D'après le Lemme 2.23 on a

$$PU = R + X^k Q \quad \text{et} \quad P'U = R' + X^k Q',$$

avec $\deg(R) < L$ et $\deg(R') < L'$. L'égalité

$$PP'U = RP' + X^k QP' = R'P + X^k Q'P$$

permet de déduire que $RP' - R'P = X^k(Q'P - QP')$. Remarquons maintenant que

$$\deg(RP' - R'P) \leq \max\{\deg(R) + \deg(P'), \deg(R') + \deg(P)\} < L + L' \leq \deg(X^k)\,.$$

Nécessairement, on a donc $RP' - R'P = 0$. Il suffit maintenant de définir $\boldsymbol{v}$ comme la suite linéraire récurrente de série formelle $V = \frac{R}{P} = \frac{R'}{P'}$ (voir Remarque 2.17). Comme $PV = R$ et $P'V = R'$, les deux LFSR (P, L) et (P', L') engendrent bien $\boldsymbol{v}$. ∎

Corollaire 2.32

Si $(P, L) \in \mathcal{R}_k(\boldsymbol{u}) \setminus \mathcal{R}_{k+1}(\boldsymbol{u})$, alors pour tout $(P', L') \in \mathcal{R}_{k+1}(\boldsymbol{u})$, on a $L' \geq k + 1 - L$. En particulier, si $\ell_k(\boldsymbol{u}) \neq \ell_{k+1}(\boldsymbol{u})$, alors $\ell_{k+1}(\boldsymbol{u}) \geq k + 1 - \ell_k(\boldsymbol{u})$.

Démonstration : C'est la contraposée de la Proposition 2.31. ∎

La proposition suivante montre qu'on peut maintenant construire explicitement un LFSR engendrant $k+1$ termes de $\boldsymbol{u}$ à partir de certains LFSR engendrant strictement moins de termes.

Proposition 2.33

Soient $\boldsymbol{u} \in \mathbb{F}^{\mathbb{N}}$ une suite et $U(X) \in \mathbb{F}[\![X]\!]$ sa série formelle associée. Soient également $(P, L) \in \mathcal{R}_k(\boldsymbol{u}) \setminus \mathcal{R}_{k+1}(\boldsymbol{u})$ et $(P', L') \in \mathcal{R}_{k'}(\boldsymbol{u}) \setminus \mathcal{R}_{k'+1}(\boldsymbol{u})$ où $k' < k$. On note α le coefficient de degré k de UP et α' le coefficient de degré k' de UP'. Alors, le LFSR donné par

$$\left(P - \frac{\alpha}{\alpha'} X^{k-k'} P' \;,\; \max\{L, L' + k - k'\}\right)$$

est dans $\mathcal{R}_{k+1}(\boldsymbol{u})$.

Démonstration : Effectuons les divisions euclidiennes de UP par X^{k+1} et UP' par $X^{k'+1}$. Par définition de P et P' et d'après le Lemme 2.23, on obtient :

$$UP = Q + \alpha X^k + X^{k+1} R \quad \text{et} \quad UP' = Q' + \alpha' X^{k'} R'$$

avec $\deg Q < L$ et $\deg Q' < L'$. Par conséquent :

$$U(P - \frac{\alpha}{\alpha'} X^{k-k'} P') = Q - \frac{\alpha}{\alpha'} X^{k-k'} Q' + X^{k+1}(R - \frac{\alpha}{\alpha'} R').$$

Notons que le degré de $Q - \frac{\alpha}{\alpha'} X^{k-k'} Q'$ est plus petit que $\max\{\deg Q, \deg Q' + k - k'\} < \max\{L, L' + k - k'\}$. En utilisant une nouvelle fois le Lemme 2.23, on obtient le résultat escompté. ∎

Comme corollaire, on obtient une forme générale pour le profil de complexité linéaire.

Corollaire 2.34

Si $\ell_k(\boldsymbol{u}) < \ell_{k+1}(\boldsymbol{u})$, alors $\ell_{k+1}(\boldsymbol{u}) = k+1-\ell_k(\boldsymbol{u})$.

Démonstration : On a montré que $\ell_{k+1}(\boldsymbol{u}) \geq k+1-\ell_k(\boldsymbol{u})$ dans le Corollaire 2.32. Montrons l'inégalité inverse par récurrence.
• **Initialisation.** Soit i le premier indice pour lequel $u_i \neq 0$. Par convention on a posé $\ell_0(\boldsymbol{u}) = 0$, et on observe que $\ell_1(\boldsymbol{u}) = \cdots = \ell_i(\boldsymbol{u}) = 1$, puis nécessairement $\ell_{i+1} = i$ (voir Exemple 2.29). Par conséquent on a bien $\ell_0(\boldsymbol{u}) + \ell_1(\boldsymbol{u}) = 1$ d'une part, et $\ell_{i+1}(\boldsymbol{u}) + \ell_i(\boldsymbol{u}) = i+1$ d'autre part.
• **Induction.** Soient k et k' tels que

$$\ell_{k'}(\boldsymbol{u}) < \ell_{k'+1}(\boldsymbol{u}) = \cdots = \ell_k(\boldsymbol{u}) < \ell_{k+1}(\boldsymbol{u})\,.$$

Par hypothèse de récurrence, on a $\ell_{k'}(\boldsymbol{u}) = k'+1-\ell_{k'+1}(\boldsymbol{u}) = k'+1-\ell_k(\boldsymbol{u})$. En utilisant la proposition précédente, on peut construire un LFSR qui engendre $\boldsymbol{u}$ sur $k+1$ termes, et dont la dimension est $\leq \max\{\ell_k(\boldsymbol{u}), \ell_{k'}(\boldsymbol{u}) + k - k'\} = \ell_{k'}(\boldsymbol{u}) + k - k'$. Par conséquent

$$\ell_{k+1}(\boldsymbol{u}) \leq \ell_{k'}(\boldsymbol{u}) + k - k' = k+1-\ell_k(\boldsymbol{u})\,.$$

■

Par conséquent, le nouveau LFSR construit dans la Proposition 2.33 est de dimension minimale si les LFSR (P, L) et (P', L') qui permettent de le construire sont eux-mêmes de dimension minimale. On en déduit ainsi l'Algorithme 4, nommé algorithme de Berlekamp–Massey, qui permet de construire une séquence de LFSR de dimension minimale engendrant la suite $\boldsymbol{u}$ sur un nombre de termes incrémental.

Algorithme 4 : Algorithme de Berlekamp–Massey

Entrée : Le polynôme $U(X) = \sum_{i=0}^{N-1} u_i X^i \in \mathbb{F}[X]$ correspondant à la suite tronquée $T_N(\boldsymbol{u})$.
Sortie : Une séquence de LFSR $((P_k, L_k))_{0 \leq k < N}$ telle que $(P_k, L_k) \in \mathcal{R}_k(\boldsymbol{u})$ et $L_k = \ell_k(\boldsymbol{u})$

```
1  i ← 0, P_0 ← 1, L_0 ← 0
2  Tant que u_i = 0 faire
3  |  i ← i + 1
4  |  P_i = 1,    L_i = 0
5  i ← i + 1
6  P_i ← 1,    L_i ← i,    k' ← i − 1,    α' ← u_{i−1}
7  Pour tout k ∈ {i, ..., N − 1} faire
8  |  α ← coefficient de degré k du polynôme produit UP_k
9  |  Si α = 0                          ▷ Le polynôme P_k engendre toujours u sur k + 1 termes
10 |  |  P_{k+1} ← P_k
11 |  |  L_{k+1} ← L_k
12 |  Sinon
13 |  |  P_{k+1} ← P_k − (α/α') X^{k−k'} P_{k'}      ▷ Le polynôme P_k n'engendre plus u au (k + 1)-ème terme
14 |  |  Si L_k > k/2
15 |  |  |  L_{k+1} ← L_k
16 |  |  Sinon
17 |  |  |  L_{k+1} ← k + 1 − L_k
18 |  |  |  k' ← k
19 |  |  |  α' ← α
20 Retourner la séquence ((P_k, L_k)).
```

L'idée de l'algorithme de Berlekamp–Massey est la suivante. On garde tout au long de l'algorithme la donnée des LFSR minimaux précédemment calculés (les premiers étant par conven-

tion $(P = 1, L = 0)$). Puis, pour des valeurs de k croissantes, on calcule de nouveaux LFSR qui engendrent la suite sur k termes.

Deux cas se présentent. Ou bien le dernier LFSR calculé engendre toujours la suite jusqu'au rang k (lignes 9 à 11), auquel cas le LFSR minimal d'indice k est égal au dernier LFSR calculé. Ou bien il n'engendre plus la suite pour le k-ème terme (lignes 12 à 19) ; il faut alors construire un nouveau LFSR minimal au moyen de la Proposition 2.33.

Notons que l'assignation $L_{k+1} \leftarrow L_k$ de l'étape 15 se justifie par le résultat suivant.

Lemme 2.35

Si $\ell_k(\boldsymbol{u}) > k/2$, alors $\ell_{k+1}(\boldsymbol{u}) = \ell_k(\boldsymbol{u})$.

Démonstration : On montre la contraposée. Supposons $\ell_{k+1}(\boldsymbol{u}) \neq \ell_k(\boldsymbol{u})$. Alors par croissance de la suite $(\ell_k(\boldsymbol{u}))_{k\in\mathbb{N}}$, on a $\ell_{k+1}(\boldsymbol{u}) \geq \ell_k(\boldsymbol{u}) + 1$, puis d'après le Corollaire 2.34,

$$\ell_k(\boldsymbol{u}) = k + 1 - \ell_{k+1}(\boldsymbol{u}) \leq k - \ell_k(\boldsymbol{u})\,.$$

Par conséquent, $\ell_k(\boldsymbol{u}) \leq k/2$. ■

Exemple 2.36

On considère une suite binaire dont les 11 *premiers termes sont :*

$$(0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0)$$

L'algorithme de Berlekamp–Massey produit alors la séquence de LFSR suivante :

k	P_k	$\ell_k(\boldsymbol{u})$
0	1	0
1	1	0
2	1	0
3	1	3
4	1	3
5	1	3

k	P_k	$\ell_k(\boldsymbol{u})$
6	$X^3 + 1$	3
7	$X^3 + 1$	3
8	$X^5 + X^3 + 1$	5
9	$X^5 + X^3 + 1$	5
10	$X^3 + X^2 + 1$	5
11	$X^3 + X^2 + 1$	5

2.2.4 Suites récurrentes vectorielles

Dans cette sous-section, on étend les résultats précédents au cas où les termes des suites sont des éléments d'un espace vectoriel plutôt que d'un corps. On appelle les suites obtenues des *suites vectorielles*.

Définition 2.37 (*Suite de vecteurs récurrente linéaire*)

Soit $n \geq 1$. Une suite de vecteurs $\boldsymbol{v} = (\boldsymbol{v}_k)_{k\in\mathbb{N}} \in (\mathbb{F}^n)^{\mathbb{N}}$ est dite récurrente linéaire *s'il existe des entiers $0 \leq D \leq L$ et des éléments $c_0, \ldots, c_D \in \mathbb{F}$ avec $c_0 \neq 0$, $c_d \neq 0$, tels que*

$$c_0\boldsymbol{v}_k + c_1\boldsymbol{v}_{k-1} + \cdots + c_D\boldsymbol{v}_{k-D} = \boldsymbol{0}, \quad \forall k \geq L\,.$$

Le polynôme $P(X) = c_0 + c_1X + \cdots + c_DX^D \in \mathbb{F}[X]$ est appelé polynôme de connexion *de la suite.*

Si $n = 1$, on dit que la suite est *scalaire* et on retrouve la définition des suites récurrentes linéaires vue dans la section précédente.

Remarque 2.38

Soit $\boldsymbol{v} = (\boldsymbol{v}_k)_{k\in\mathbb{N}} \in (\mathbb{F}^n)^{\mathbb{N}}$ une suite vectorielle récurrente linéaire de polynôme de connexion P. Pour tout $j \in \{1,\dots,n\}$, on note $\boldsymbol{v}^{(j)} = (v_k^{(j)})_{k\in\mathbb{N}} \in \mathbb{F}^{\mathbb{N}}$ la suite scalaire constituée des j-ème coordonnées des vecteurs $\boldsymbol{v}_k \in \mathbb{F}^n$. Alors la suite $\boldsymbol{v}^{(j)}$ est une suite récurrente linéaire de polynôme de connexion P (quitte à le diviser par son terme constant).

Exemple 2.39

Pour $n = 2$, on considère la suite définie par

$$\boldsymbol{v}_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \text{et par} \quad \begin{pmatrix} v_k^{(1)} \\ v_k^{(2)} \end{pmatrix} = \begin{pmatrix} v_{k-1}^{(2)} \\ v_{k-1}^{(1)} \end{pmatrix}, \quad \forall k \geq 1\,.$$

Observons que l'on peut l'écrire

$$\boldsymbol{v}_k = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \boldsymbol{v}_{k-1}, \quad \forall k \geq 1\,.$$

La suite $\boldsymbol{v}$ a alors pour polynôme de connexion X^2+1, car $v_k^{(1)} + v_{k-2}^{(1)} = 0$ et $v_k^{(2)} + v_{k-2}^{(2)} = 0$ pour tout $k \geq 1$.

Comme pour le cas scalaire, l'ensemble des polynômes de connexion d'une suite vectorielle a une structure non-triviale.

Lemme 2.40

L'ensemble $\mathcal{I}_{\boldsymbol{v}}$ des polynômes de connexion d'une suite vectorielle récurrente linéaire $\boldsymbol{v} \in (\mathbb{F}^n)^{\mathbb{N}}$ forme un idéal de $\mathbb{F}[X]$.

Démonstration : Si $P, Q \in \mathcal{I}_{\boldsymbol{v}}$, alors on peut vérifier que pour tout $\lambda \in \mathbb{F}$, les polynômes $\lambda P(X)$, $P(X)+Q(X)$, et $XP(X)$ sont également dans $\mathcal{I}_{\boldsymbol{v}}$:

— $\sum_{i=0}^{D} \lambda p_i \boldsymbol{v}_{k-i} = \lambda \sum_{i=0}^{D} p_i \boldsymbol{v}_{k-i} = \boldsymbol{0}$,

— $\sum_{i=0}^{D} (p_i + q_i)\boldsymbol{v}_{k-i} = \sum_{i=0}^{D} p_i \boldsymbol{v}_{k-i} + \sum_{i=0}^{D} q_i \boldsymbol{v}_{k-i} = \boldsymbol{0}$,

— comme $XP(X) = \sum_{i=1}^{D+1} p_{i-1} X^i$, on a $\sum_{i=1}^{D+1} p_{i-1}\boldsymbol{v}_{k+1-i} = \sum_{i=0}^{D} p_i \boldsymbol{v}_{k-i} = \boldsymbol{0}$ pour $k \geq L$. ■

Comme $\mathbb{F}[X]$ est euclidien (donc principal), on peut donner un sens à la notion de polynôme de connexion *minimal*.

Définition 2.41

Le polynôme (de connexion) minimal d'une suite vectorielle récurrente linéaire $\boldsymbol{v} \in (\mathbb{F}^n)^{\mathbb{N}}$ est l'unique polynôme de connexion de $\boldsymbol{v}$ unitaire et de plus petit degré. C'est un générateur de $\mathcal{I}_{\boldsymbol{v}}$, que l'on note $P_{\boldsymbol{v}}(X)$.

Comme pour le cas scalaire, on montre aisément que $P_{\boldsymbol{v}}(0) \neq 0$ car sinon $\mathcal{I}_{\boldsymbol{v}}$ contiendrait $P_{\boldsymbol{v}}(X)/X$ qui est de degré strictement inférieur à $P_{\boldsymbol{v}}(X)$.

On peut relier le polynôme de connexion minimal d'une suite vectorielle récurrente linéaire, à ceux de ses suites « coordonnées ».

Lemme 2.42

Soit $\boldsymbol{v} \in (\mathbb{F}^n)^{\mathbb{N}}$ une suite vectorielle linéaire récurrente et $P_{\boldsymbol{v}^{(i)}}(X)$ le polynôme de connexion minimal de $\boldsymbol{v}^{(i)}$, pour tout $i \in \{1, \dots, n\}$. Alors, le polynôme de connexion minimal de $\boldsymbol{v}$ est

$$P_{\boldsymbol{v}}(X) = \mathrm{ppcm}(P_{\boldsymbol{v}^{(1)}}(X), \dots, P_{\boldsymbol{v}^{(n)}}(X)) .$$

Démonstration : Pour obtenir ce résultat, il suffit de démontrer que

$$\mathcal{I}_{\boldsymbol{v}} = \mathcal{I}_{\boldsymbol{v}^{(1)}} \cap \cdots \cap \mathcal{I}_{\boldsymbol{v}^{(n)}} .$$

En effet, l'intersection $\mathcal{I}_{\boldsymbol{v}^{(1)}} \cap \cdots \cap \mathcal{I}_{\boldsymbol{v}^{(n)}}$ est un idéal de générateur $A(X) = \mathrm{ppcm}(P_{\boldsymbol{v}^{(1)}}(X), \dots, P_{\boldsymbol{v}^{(n)}}(X))$.
En écrivant l'équation $\sum_{i=0}^{D} p_i \boldsymbol{v}_{n-i}$ sur chacune des coordonnées des vecteurs, il est clair que tout polynôme de connexion $P(X)$ de $\boldsymbol{v}$ est également un polynôme de connexion de $\boldsymbol{v}^{(j)}$, pour tout $j \in \{1, \dots, n\}$.
Réciproquement, tout polynôme $Q(X)$ qui est un polynôme de connexion de chacune des suites $\boldsymbol{v}^{(j)}$ est également un polynôme de connexion de $\boldsymbol{v}$: l'équation $\sum_{i=0}^{D} p_i \boldsymbol{v}_{n-i}$ sera vérifiée dès lors que n est suffisamment grand. ■

Remarque 2.43

On peut étendre les résultats précédents en remplaçant les suites $\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(n)} \in \mathbb{F}^{\mathbb{N}}$ par les suites issues de produits scalaires $\langle \boldsymbol{v}, \boldsymbol{w}_1 \rangle, \dots, \langle \boldsymbol{v}, \boldsymbol{w}_n \rangle \in \mathbb{F}^{\mathbb{N}}$, où $\boldsymbol{w}_1, \dots, \boldsymbol{w}_n$ sont des vecteurs fixes de $\mathbb{F}^n$ qui en forment une base. La preuve est laissée en exercice.

Suites itérées. Une manière de construire des suites vectorielles récurrentes linéaires et d'appliquer successivement une matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ à un vecteur initial $\boldsymbol{b} \in \mathbb{F}^n$. On obtient une suite $\boldsymbol{v} = (\boldsymbol{A}^k \boldsymbol{b})_{k \in \mathbb{N}} \in (\mathbb{F}^n)^{\mathbb{N}}$ qu'on appelle *suite itérée* de matrice $\boldsymbol{A}$ et de vecteur initial $\boldsymbol{b}$.

On va voir que le polynôme de connexion minimal de la suite $\boldsymbol{v}$ est naturellement lié au polynôme annulateur minimal de la matrice $\boldsymbol{A}$. On rappelle que le *polynôme (annulateur) minimal* d'une matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ est le polynôme unitaire $P(X) = p_0 + p_1 X + \cdots + p_d X^d$ de plus petit degré tel que $P(\boldsymbol{A}) := p_0 \boldsymbol{I} + p_1 \boldsymbol{A} + \cdots + p_d \boldsymbol{A}^d = \boldsymbol{0}$. On note $\mu_{\boldsymbol{A}}(X)$ le polynôme minimal de la matrice $\boldsymbol{A}$.

Pour tout polynôme $P(X) = p_0 + p_1 X + \cdots + p_d X^d \in \mathbb{F}[X]$, définissons

$$P^*(X) := X^d P(1/X) = p_d + p_{d-1} X + \cdots + p_0 X^d$$

son *polynôme réciproque*. Observons que $P \mapsto P^*$ est un isomorphisme d'anneaux, et que $A(X) | B(X)$ si et seulement si $A^*(X) | B^*(X)$.

Lemme 2.44

Soit $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ non-nulle. Pour tout $\boldsymbol{x} \in \mathbb{F}^n$ non-nul, la suite $\boldsymbol{v} = (\boldsymbol{A}^k \boldsymbol{x})_{k \in \mathbb{N}}$ est une suite récurrente linéaire vectorielle. Par ailleurs, son polynôme de connexion minimal $P_{\boldsymbol{v}}(X)$ divise $\mu^*_{\boldsymbol{A}}(X)$.

Démonstration : Les vecteurs $\{\boldsymbol{x}, \boldsymbol{A}\boldsymbol{x}, \dots, \boldsymbol{A}^n \boldsymbol{x}\}$ sont au nombre de $n+1$, donc forment une famille liée de $\mathbb{F}^n$: il existe des coefficients $\lambda_0, \dots, \lambda_n \in \mathbb{F}$, l'un d'entre eux au moins n'étant pas nul, tels que

$$\sum_{i=0}^{n} \lambda_i \boldsymbol{A}^i \boldsymbol{x} = \boldsymbol{0} .$$

Autrement dit, on a $\sum_{i=0}^{n} \lambda_i \boldsymbol{v}_i = \boldsymbol{0}$ que l'on peut réécrire $\sum_{i=0}^{n} p_i \boldsymbol{v}_{n-i} = \boldsymbol{0}$ en posant $p_i = \lambda_{n-i}$.

Cette équation reste vérifiée pour les termes suivants de la suite $\boldsymbol{v}$. En effet en appliquant $\boldsymbol{A}^k$ à gauche, on obtient :

$$\forall k \geq 0, \quad \sum_{i=0}^{n} p_i \, \boldsymbol{v}_{k+n-i} = \boldsymbol{0} \, .$$

La suite $\boldsymbol{v}$ est donc récurrente, elle admet donc un polynôme de connexion minimal, noté $P_{\boldsymbol{v}}(X)$, de degré $\leq n$.

Le polynôme minimal $\mu_{\boldsymbol{A}}(X) = \sum_{i=0}^{d} c_i X^i \in \mathbb{F}[X]$ de la matrice $\boldsymbol{A}$ vérifie $\mu_{\boldsymbol{A}}(\boldsymbol{A}) \cdot \boldsymbol{x} = \boldsymbol{0} \cdot \boldsymbol{x} = \boldsymbol{0}$. Par conséquent,

$$\sum_{i=0}^{d} c_i \boldsymbol{A}^i \boldsymbol{x} = \sum_{i=0}^{d} c_{d-i} \boldsymbol{v}_{d-i} = \boldsymbol{0} \, ,$$

et on vérifie que cette relation de récurrence est satisfaite à tout ordre en multipliant l'équation par unne puissance de $\boldsymbol{A}$. Ainsi, on a montré que $\mu^*_{\boldsymbol{A}}(X) \in \mathcal{I}_{\boldsymbol{v}}$. Il en résulte que $P_{\boldsymbol{v}}(X)$ divise $\mu^*_{\boldsymbol{A}}(X)$. ■

Exemple 2.45

Sur $\mathbb{F}_2$, on définit

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad \text{et} \quad \boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} .$$

Le polynôme minimal de $\boldsymbol{A}$ est $X^3 + X^2$. Son polynôme réciproque est $X + 1$. Par ailleurs, on a

$$\boldsymbol{A}\boldsymbol{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{et} \quad \boldsymbol{A}^2\boldsymbol{x} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Puis, on note que

$$\boldsymbol{A}^k\boldsymbol{x} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \forall k \geq 2 \, .$$

Donc le polynôme minimal de la suite $\boldsymbol{v} = (\boldsymbol{A}^k\boldsymbol{x})_{k \in \mathbb{N}}$ est $X + 1$, car $\boldsymbol{v}$ est constante à partir d'un certain rang.

2.3 Algèbre linéaire creuse

2.3.1 Motivation et définitions

Dans la Section 2.1, on a vu qu'il existait un algorithme générique qui permet de calculer l'ensemble des solutions d'un système linéaire $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ en temps $O(n^3)$.

Dans certaines applications scientifiques, des algorithmes avancés se ramènent à la résolution efficace de systèmes linéaires de taille importante, pour lesquels la matrice $\boldsymbol{A}$ est, ou bien structurée, ou bien contient beaucoup de zéros (on parle de système *creux*). C'est le cas par exemple de la phase d'algèbre linéaire des algorithmes modernes de factorisation et de calcul de logarithmes discret, que nous verrons dans les chapitres ultérieurs.

Si l'on utilise la méthode par éliminiation gaussienne présentée dans la Section 2.1, alors la mise sous forme échelonnée de la matrice lui fait perdre son caractère creux. Cela s'observe très clairement dans l'illustration de la Figure 2.2.

À titre d'exemple, dans la factorisation de RSA-768 en 2009 par Kleinjung et ses coauteurs [KAF+10], un système linéaire d'environ 192 millions d'inconnues et équations a dû

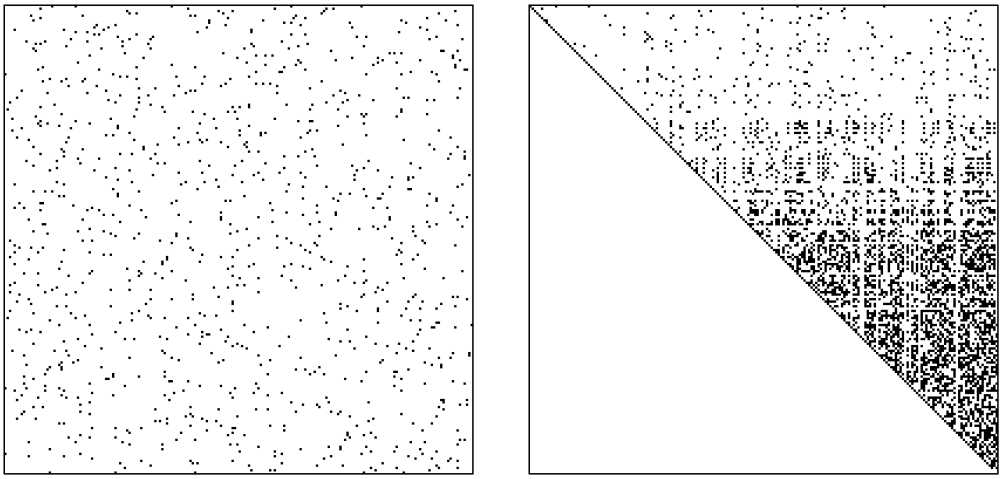


FIGURE 2.2 – À gauche : une matrice binaire $\boldsymbol{A}$ aléatoire de taille 200×200, avec 5 coefficients non-nuls par ligne (représentés par des carrés noirs). À droite sa forme échelonnée après exécution de l'algorithme de Gauss. On observe une densification de la partie inférieure droite de la matrice : la forme échelon d'une matrice creuse n'est pas creuse.

être résolu. Avec un algorithme générique de résolution du système, et un stockage dense de la matrice associée, plus de 4000 To de mémoire vive et l'équivalent de 2^{80} opérations auraient été nécessaires, ce qui rend la tâche impossible.

Heureusement, la matrice du système contenait « seulement » $\simeq$ 27 milliards de coefficients non-nuls, soit un peu moins d'un coefficient non-nul par million d'entrées de la matrice. Cette matrice a donc pu être stockée de manière « creuse », avec seulement une centaine de giga-octets (Go). Grâce à un algorithme dérivé de l'algorithme de Wiedemann que nous verrons dans ce chapitre, la résolution du système linéaire a pu être conduite en un temps estimé à environ 2000 années de calculs sur un unique processeur classique (AMD Opteron 2,2GHz). Grâce à l'utilisation de clusters de plusieurs dizaines de processeurs, le calcul a donc pu être mené sur un temps raisonnable.

Avant de préciser les objectifs de cette section, commençons par définir la notion de matrice creuse, c'est-à-dire ayant très peu de coefficients non-nuls.

Définition 2.46

Soit $1 \leq t \leq n$. On dit qu'une matrice $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ est t-creuse si chaque ligne de $\boldsymbol{A}$ contient moins de t coefficients non-nuls.

Pour des systèmes linéaires impliquant une matrice t-creuse, nos objectifs sont les suivants.

- **Objectif 1 :** obtenir une *solution particulière* du système $\boldsymbol{Ax} = \boldsymbol{b}$ en temps $O(tn^2)$ et en espace $O(tn)$, où $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ est t-creuse et $\boldsymbol{0} \neq \boldsymbol{b} \in \mathbb{F}^n$.
- **Objectif 2 :** obtenir un *élément du noyau à droite* de $\boldsymbol{A}$ en temps $O(tn^2)$ et en espace $O(tn)$, où $\boldsymbol{A} \in \mathbb{F}^{n \times n}$ est t-creuse.

Pour ce faire, nous allons considérer la suite vectorielle itérée de matrice $\boldsymbol{A}$ et de vecteur initial $\boldsymbol{b}$. Ainsi jusqu'à la fin de ce chapitre on va noter $\boldsymbol{v} \in (\mathbb{F}^n)^{\mathbb{N}}$ la suite définie par

$$\boldsymbol{v} \coloneqq (\boldsymbol{A}^k \boldsymbol{b})_{k \in \mathbb{N}}\,.$$

2.3.2 Calcul d'une solution particulière

Dans cette section, nous allons nous placer dans cas d'étude simplifié où **la matrice $\boldsymbol{A}$ est inversible**.

Notons que si $\boldsymbol{A}$ n'est pas inversible, il est possible se ramener au cas inversible par plusieurs moyens : l'un d'entre eux est par exemple de rajouter un nombre suffisant de lignes et de colonnes aléatoires de faible poids jusqu'à obtenir une matrice $\boldsymbol{A}'$ inversible, et déduire une solution de $\boldsymbol{Ax} = \boldsymbol{b}$ d'une solution de $\boldsymbol{A}'\boldsymbol{x}' = \boldsymbol{b}'$ pour un $\boldsymbol{b}'$ judicieusement choisi. Voir [Wie86] pour les détails.

Pour décrire la méthode de calcul efficace d'une solution particulière de $\boldsymbol{Ax} = \boldsymbol{b}$, commençons par une observation. Si $P = \sum_{i=0}^{d} p_i X^i \in \mathbb{F}[X]$ vérifie $p_0 = P(0) \neq 0$ et $P(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}$, alors on a :

$$p_0\boldsymbol{b} + p_1\boldsymbol{Ab} + \cdots + p_d\boldsymbol{A}^d\boldsymbol{b} = \boldsymbol{0}\,.$$

Puis, en isolant $\boldsymbol{b}$ dans l'équation ci-dessus, il résulte que

$$\boldsymbol{b} = \boldsymbol{A}Q(\boldsymbol{A})\boldsymbol{b}, \quad \text{avec} \quad Q(X) = \frac{1}{X}\left(1 - \frac{P(X)}{P(0)}\right).$$

Autrement dit, le vecteur $\boldsymbol{x} = Q(\boldsymbol{A})\boldsymbol{b}$ est une solution particulière de l'équation $\boldsymbol{Ax} = \boldsymbol{b}$.

Remarque 2.47

Il est important de noter que, si $\boldsymbol{A}$ est t-creuse et $Q(X)$ est de degré $\leq d-1$, alors le calcul de $Q(\boldsymbol{A})\boldsymbol{b}$ s'effectue en temps $O(dnt)$ par une méthode de type « évaluation de Horner ».
Plus précisément, dans notre cas $Q(X) = \sum_{i=0}^{d-1} q_i X^i$, alors

$$Q(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{A}(\boldsymbol{A}\cdots\boldsymbol{A}(\boldsymbol{A}(q_{d-1}\boldsymbol{Ab} + q_{d-2}\boldsymbol{b}) + q_{d-3}\boldsymbol{b}) + \cdots + q_1\boldsymbol{b}) + q_0\boldsymbol{b}\,,$$

et chaque opération $\boldsymbol{u} \mapsto \boldsymbol{Au}$ coûte $O(nt)$ opérations (les sommes de vecteurs étant en $O(n)$). Par ailleurs, on observe que les calculs se font « en place », donc la complexité en espace reste en $O(nt)$.

Il nous reste donc à trouver un algorithme efficace qui, étant donnés $\boldsymbol{A}$ et $\boldsymbol{b}$, calcule un polynôme $P(X) \in \mathbb{F}[X]$ satisfaisant $P(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}$ et $P(0) \neq 0$.

Une première idée est de s'intéresser à $\mu_{\boldsymbol{A}}(X)$, le polynôme minimal de $\boldsymbol{A}$. Il est clair que $\mu_{\boldsymbol{A}}(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}\cdot\boldsymbol{b} = \boldsymbol{0}$, et par ailleurs on a le résultat suivant.

Lemme 2.48

La matrice $\boldsymbol{A}$ est inversible si et seulement si $\mu_{\boldsymbol{A}}(0) \neq 0$, où $\mu_{\boldsymbol{A}}(X)$ est le polynôme minimal de $\boldsymbol{A}$.

Démonstration : On sait que toute valeur propre de $\boldsymbol{A}$ est racine de son polynôme minimal $\mu_{\boldsymbol{A}}(X)$. Or, 0 est valeur propre de $\boldsymbol{A}$ si et seulement si $\boldsymbol{A}$ est non-inversible, ce qui conclut la preuve. ∎

Nous verrons dans une section ultérieure comment calculer efficacement $\mu_{\boldsymbol{A}}(X)$. En réalité, nous pouvons chercher un polynôme « plus petit » que $\mu_{\boldsymbol{A}}(X)$. Pour cela, observons que, par définition, le polynôme de connexion minimal $P_{\boldsymbol{v}}(X) = \sum_{i=0}^{r} \lambda_i X^i$ satisfait

$$\boldsymbol{0} = \lambda_0\boldsymbol{v}_n + \lambda_1\boldsymbol{v}_{n-1} + \cdots + \lambda_r\boldsymbol{v}_{n-r}$$

pour tout n supérieur ou égal à un certain rang L. Notons que cette écriture et équivalente à

$$\boldsymbol{A}^{n-r}P^*_{\boldsymbol{v}}(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}\,, \quad \forall n \geq L$$

et quitte à multiplier/diviser par $\boldsymbol{A}^{n-r}$ inversible, on obtient

$$P_v^*(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}\,.$$

Grâce au Lemme 2.44, on sait par ailleurs que $P_v^*(X)$ divise $\mu_{\boldsymbol{A}}(X)$. Ainsi, $P_v^*(X)$ est un polynôme de plus petit degré que $\mu_{\boldsymbol{A}}(X)$, et qui vérifie également $P_v^*(0) \neq 0$; c'est le polynôme que l'on va chercher à calculer.

On obtient donc la méthode de résolution suivante.

1. Calculer le polynôme minimal $P_{\boldsymbol{v}}(X) = \sum_{j=0}^{d} \lambda_j X^j$ de $\boldsymbol{v} = (\boldsymbol{A}^k\boldsymbol{b})_{k\in\mathbb{N}}$, et en déduire son polynôme réciproque $P_v^*(X)$.
2. Calculer $Q(X) = \frac{1}{X}(1 - \frac{P_v^*(X)}{P_v^*(0)})$.
3. Retourner $\boldsymbol{x} = Q(\boldsymbol{A})\boldsymbol{b}$.

Il nous reste à trouver un algorithme efficace pour calculer le polynôme de connexion minimal $P_{\boldsymbol{v}}(X)$. Le Lemme 2.42 (et la remarque qui en découle) nous indique que le calcul de $P_{\boldsymbol{v}}(X)$ peut se réduire à un calcul de ppcm et à n calculs de polynômes de connexion minimaux scalaires. On obtient ces polynômes de connexion scalaires par l'algorithme de Berlekamp–Massey qui s'exécute en temps $O(n^2)$: par cette méthode, la complexité totale est donc en $O(n^3)$, ce qui est trop important.

L'idée va donc être d'espérer obtenir $P_{\boldsymbol{v}}(X)$ en calculant le ppcm d'un nombre constant de polynômes scalaires, de degré de plus en plus petit. Pour cela, on utilise conjointement les deux idées suivantes :

— on calcule les polynômes de connexion minimaux de suites scalaires de la forme $\boldsymbol{u} = (\langle \boldsymbol{y}, \boldsymbol{v}_k\rangle)_{k\in\mathbb{N}}$, où $\boldsymbol{y} \in \mathbb{F}^n$ est tiré aléatoirement; on espère que ces polynômes de connexion forment des diviseurs de $P_{\boldsymbol{v}}(X)$ de grand degré;
— on utilise les polynômes précédemment calculés pour obtenir des suites récurrentes d'ordre plus petits.

Précisons ces deux idées avec quelques résultats théoriques.

Lemme 2.49

Soit $P(X) \in \mathbb{F}[X]$. Alors,

$$P(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0} \quad \Longleftrightarrow \quad P_{\boldsymbol{v}}(X) \text{ divise } P^*(X)\,.$$

Démonstration : à faire, mais essentiellement déjà faite ■

Lemme 2.50

Soit $\boldsymbol{v} = (\boldsymbol{A}^k\boldsymbol{b})_{k\in\mathbb{N}}$ et $P(X)$ un diviseur de $P_{\boldsymbol{v}}(X)$. Alors,

$$\frac{P_{\boldsymbol{v}}(X)}{P(X)} = P_{\boldsymbol{v}'}(X)$$

où $\boldsymbol{v}' = (\boldsymbol{A}^k\boldsymbol{b}')_{k\in\mathbb{N}}$ avec $\boldsymbol{b}' = P^*(\boldsymbol{A})\boldsymbol{b}$.

Démonstration : On montre que $P_{\boldsymbol{v}'}(X)P(X) = P_{\boldsymbol{v}}(X)$ par divisibilité réciproque. D'une part on a

$$(P_{\boldsymbol{v}'}^* \cdot P^*)(\boldsymbol{A})\boldsymbol{b} = P_{\boldsymbol{v}'}^*(\boldsymbol{A})P^*(\boldsymbol{A})\boldsymbol{b} = P_{\boldsymbol{v}'}^*(\boldsymbol{A})\boldsymbol{b}' = \boldsymbol{0}$$

donc le polynôme $P_{\boldsymbol{v}}(X)$ divise le produit $P_{\boldsymbol{v}'}(X)P(X)$ d'après le Lemme 2.49. D'autre part, soit $Q(X) = P_{\boldsymbol{v}}(X)/P(X)$. On observe que

$$Q^*(\boldsymbol{A})\boldsymbol{b}' = Q^*(\boldsymbol{A})P^*(\boldsymbol{A})\boldsymbol{b} = P_{\boldsymbol{v}}^*(\boldsymbol{A})\boldsymbol{b} = \boldsymbol{0}$$

donc $P_{v'}(X)$ divise $Q(X)$, puis $P_{v'}(X)P(X)$ divise $P_v(X)$. ■

La méthode de calcul de $P_v(X)$ est finalement résumée dans l'Algorithme 5.

Algorithme 5 : Algorithme de calcul du polynôme minimal $P_v(X)$ d'une suite itérée $v = (A^k b)_{k\in\mathbb{N}}$.

Entrée : Une matrice creuse $A \in \mathbb{F}^{n\times n}$, un vecteur $b \in \mathbb{F}^n$
Sortie : Le polynôme de connexion minimal $P_v(X)$ où $v = (A^k b)_{k\in\mathbb{N}}$

1 $b' \leftarrow b$
2 $P \leftarrow 1$
3 **Tant que** $b' \neq 0$ **faire**
4 Choisir aléatoirement $y \in \mathbb{F}^n$ non-nul.
5 Calculer les $2n$ premiers termes de la suite scalaire u définie par $u_k = \langle y, A^k b\rangle$.
6 **Si** u *est nulle*
7 Revenir à l'étape 4.
8 $Q \leftarrow$ `BerlekampMassey`(u)
9 $P \leftarrow P \times Q^*$
10 $b' \leftarrow Q^*(A)b'$
11 **Retourner** P

Theorème 2.51

Soit $v \in (\mathbb{F}^n)^{\mathbb{N}}$ la suite itérée de matrice $A \in \mathbb{F}^{n\times n}$ inversible et de vecteur initial $b \in \mathbb{F}^n$. L'Algorithme 5 termine et calcule $P_v(X)$ en effectuant, en moyenne, $O(tn^2)$ opérations dans le corps $\mathbb{F}$.

Démonstration : À venir. ■

2.3.3 Calcul d'un élément du noyau.

Supposons que $A \in \mathbb{F}^{n\times n}$ est creuse est non-inversible. On cherche un élément non-nul $x \in \mathbb{F}^n$ tel que $Ax = 0$. Pour cela, notons d'après le Lemme 2.48 que le polynôme minimal $\mu_A(X)$ s'écrit comme $X^r Q(X)$, avec $Q(0) \neq 0$ et $r \geq 1$ car 0 est valeur propre de A.

L'idée de l'algorithme de Wiedemann est alors la suivante : avec bonne probabilité sur le tirage de $u \in \mathbb{F}^n$, le vecteur $x = A^{r-1}Q(A)u$ est non-nul, et il est dans le noyau de A car $AA^{r-1}Q(A) = \mu_A(A) = 0$.

On va donc chercher à calculer $Q(X)$ par un procédé similaire à celui de la partie précédente. On engendre des suites scalaires aléatoires de la forme $a = (\langle v, A^k w\rangle)_{k\in\mathbb{N}}$, et on calcule le ppcm des polynômes de connexion minimaux $P_a(X)$ obtenus grâce à l'algorithme de Berlekamp–Massey.

Une fois $Q(X)$ calculé, comme la valeur de l'exposant $r \geq 1$ est a priori inconnu, l'astuce est la suivante. On tire aléatoirement $u \in \mathbb{F}^n$, et on calcule $x' = Q(A)u$. Si cet élément x' est non-nul, alors il suffit de calculer itérativement Ax', A^2x', ... jusqu'à obtenir un vecteur nul (disons, $A^{r'}x' = 0$). Alors, en posant $x = A^{r'-1}x'$, on a $Ax = 0$.

Algorithme 6 : Algorithme de Wiedemann

Entrée : une matrice creuse $\boldsymbol{A} \in \mathbb{F}^{n\times n}$
Sortie : un élément aléatoire du noyau de $\boldsymbol{A}$

1 Initialiser $Q^*(X) \leftarrow 1$.
2 **Tant que** *la suite des polynômes Q n'a pas convergé* **faire**
3 Choisir aléatoirement $\boldsymbol{v}$ et $\boldsymbol{w}$ non-nuls dans $\mathbb{F}^n$.
4 Calculer les $2n$ premiers termes de $\boldsymbol{a} := (\langle \boldsymbol{v}, \boldsymbol{A}^k\boldsymbol{w}\rangle)_{k\in\mathbb{N}}$.
5 Calculer $P_{\boldsymbol{a}}(X)$ grâce à l'algorithme de Berlekamp–Massey.
6 Calculer $Q^*(X) \leftarrow \mathrm{ppcm}(Q^*(X), P_{\boldsymbol{a}}(X))$.
7 Calculer $Q(X)$ à partir de son polynôme réciproque $Q^*(X)$.
8 Assigner $\boldsymbol{x}' \leftarrow \boldsymbol{0}$.
9 **Tant que** $\boldsymbol{x}' = \boldsymbol{0}$ **faire**
10 Tirer aléatoirement $\boldsymbol{u} \in \mathbb{F}^n \setminus \{\boldsymbol{0}\}$.
11 Calculer $\boldsymbol{x}' = Q(\boldsymbol{A})\boldsymbol{u}$.
12 Assigner $\boldsymbol{x} \leftarrow \boldsymbol{x}'$.
13 **Tant que** $\boldsymbol{x} \neq \boldsymbol{0}$ **faire**
14 $\boldsymbol{x}' \leftarrow \boldsymbol{x}$.
15 $\boldsymbol{x} \leftarrow \boldsymbol{A}\boldsymbol{x}$.
16 **Retourner** $\boldsymbol{x}'$.

Si l'on suppose $\mathbb{F}$ fini de cardinal q, on peut effectuer un tirage uniforme lors de l'étape 10 de l'Algorithme 6. Alors, la probabilité que $\boldsymbol{x}' = \boldsymbol{0}$ après un passage dans la boucle est :

$$\mathbb{P}(\boldsymbol{x}' = \boldsymbol{0}) = \mathbb{P}(\boldsymbol{u} \in \ker Q(\boldsymbol{A})) = \frac{q^{\dim\ker Q(\boldsymbol{A})} - 1}{q^n - 1} \simeq \frac{1}{q^{\mathrm{rang}(\boldsymbol{A})}}\,.$$

Theorème 2.52

Soit $\boldsymbol{A} \in \mathbb{F}^{n\times n}$ une matrice t-creuse. Alors, l'algorithme de Wiedemann calcule un élément du noyau de $\boldsymbol{A}$ en temps moyen $O(tn^2)$.

Démonstration : À venir. ■

Des exemples à venir.

Chapitre 3

Calcul de racines carrées

Ce chapitre a pour objet l'extraction effective de racines carrées. Dans une première section, nous étudierons le problème dans l'ensemble des entiers naturels. Puis, nous nous tournerons vers les corps finis et les anneaux d'entiers modulaires, pour lesquels le calcul de racines carrées a des applications pratiques que nous découvrirons dans les chapitres suivants.

3.1 Racines carrées entières

On s'intéresse d'abord au problème du calcul de racine carrée dans les entiers naturels, qui s'avère sensiblement plus simple que dans d'autres structures algébriques. Explicitons d'abord le problème.

Problème 1

Étant donné $n \in \mathbb{N}$, calculer le plus grand entier $x \in \mathbb{N}$ tel que $x^2 \leq n$.

On appelle *racine carrée entière* de n l'entier x trouvé, et *reste* l'entier $n - x^2 \geq 0$. Par exemple, $n = 103$ a pour racine carrée entière $x = 10$ et pour reste $r = 3$.

Remarque 3.1

La racine carrée x et le reste r (de n) forment l'unique couple d'entiers tels que

$$\begin{cases} n = x^2 + r \\ r \leq 2x \end{cases}$$

3.1.1 Méthode élémentaire

La méthode exhaustive (par incrémentation de x et test) pour calculer la racine carrée a une complexité en $O(\sqrt{n})$ opérations entières. Une première manière de réaliser ce calcul plus efficacement consiste à calculer de proche en proche la décomposition en base 2 de la racine carrée. Pour cela, on utilise la caractérisation suivante.

Lemme 3.2

Soient $n \geq 0$ et $n' = 4n + m$ où $m \in \{0, 1, 2, 3\}$. Notons x la racine carrée entière de n et r son reste. Alors :

- *lorsque $4r + m \leq 4x$, l'entier n' admet comme racine carrée $2x$ et comme reste $4r + m$;*
- *lorsque $4r + m \geq 4x + 1$, l'entier n' admet comme racine carrée $2x + 1$ et comme reste $4r + m - 4x - 1$.*

Démonstration : On écrit

$$n' = 4n + m = 4(x^2 + r) + m = (2x)^2 + (4r + m)$$

puis on teste si $4r + m$ est un reste potentiel de n', grâce à la Remarque 3.1. Si $4r + m \leq 2x$, alors on a le bon reste. Sinon il faut ajouter 1 à $2x$ et on obtient l'écriture $n' = (2x + 1)^2 + (4(r - x) + m - 1)$. ■

Le Lemme 3.2 induit une méthode itérative pour le calcul d'une racine carrée entière, présentée dans l'Algorithme 7.

Algorithme 7 : Méthode élémentaire pour le calcul de racine carrée entière

Entrée : Un entier $n = \sum_{j=0}^{d} n_j 4^j \in \mathbb{N}$

Sortie : Le plus grand entier $x = \sum_{j=0}^{d} x_j 2^j \in \mathbb{N}$ tel que $x^2 \leq n$.

```
Initialiser x_d ← 0 et r_d ← 0.
Pour tout i allant de d à 0 faire
    Calculer t ← 4r_i + n_i
    Si t > 4x_i
        x_{i-1} ← 2x_i + 1
        r_{i-1} ← t − 4x_i − 1
    Sinon
        x_{i-1} ← 2x_i
        r_{i-1} ← t
Retourner x.
```

Lemme 3.3

La « méthode élémentaire » de l'Algorithme 7 calcule une racine carrée entière d'un entier n en temps $O(\log(n))$.

Démonstration : Chaque étape de l'algorithme produit un chiffre en base 2 de la racine carrée. La complexité approximative de l'Algorithme 7 est donc en $O(\log_4(n)) = O(\log(n))$ opérations sur les entiers. Notons plus précisément que les opérations sont, ou bien l'addition de deux entiers (de coût $O(\log(n))$ opérations binaires), ou bien la multiplication par 2 ou 4 (également de coût $O(\log(n))$ opérations binaires). On évite donc les opérations plus coûteuses comme la multiplication « complète » ou la division, et on a donc une complexité binaire en $O(\log_2^2(n))$. ■

Exemple 3.4

On prend $n = 4724$, c'est-à-dire $n = \sum_{j=0}^{d} n_j 4^j$ avec $d = 6$ et $(n_d, \ldots, n_0) = (1, 0, 2, 1, 3, 1, 0)$. On note $\overline{x}_i$ et $\overline{r}_i$ les valeurs de x et r calculées à l'étape i. On indique également par $\overline{n}_i$ le quotient de la division de x par 4^i, afin de remarquer que $\overline{x}_i^2 + \overline{r}_i = \overline{n}_i$ à chaque tour de boucle. Voici le déroulement du calcul :

i	$\overline{n}_i$	$\overline{x}_i$	$\overline{r}_i$	t
7		0	0	1
6	$1 = (1)_4$	$1 = (1)_2$	0	0
5	$4 = (10)_4$	$2 = (10)_2$	0	2
4	$18 = (102)_4$	$4 = (100)_2$	2	9
3	$73 = (1021)_4$	$8 = (1000)_2$	9	39
2	$295 = (10213)_4$	$17 = (10001)_2$	6	25
1	$1181 = (102131)_4$	$34 = (100010)_2$	25	100
0	$4724 = (1021310)_4$	$68 = (1000100)_2$	100	

3.1.2 Méthode de Héron

Une seconde manière pour extraire une racine carrée entière consiste à effectuer des itérations de la méthode de Newton. Rappelons rapidement cette méthode (certainement vue dans un cours de Licence/Master 1).

Étant donnée une fonction $f : \mathbb{R} \to \mathbb{R}$ de classe $\mathcal{C}^2$, on souhaite obtenir une valeur approchée d'un élément x tel que $f(x) = 0$. Pour des raisons techniques, on suppose également que $f''(x) \neq 0$. On construit alors une suite récurrente

$$x_0 \text{ arbitraire} \quad \text{et} \quad x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

et on observe que la suite converge vers un élément $a \in \mathbb{R}$ tel que $f(a) = 0$.

Pour le cas du calcul d'une racine carrée entière, la méthode est appelée *méthode de Héron* ou *méthode babylonnienne*. On choisit alors la fonction $f(x) = x^2 - n$, afin que $a = \sqrt{n}$ soit une limite possible de la suite récurrente définie comme :

$$x_0 \text{ arbitraire} \quad \text{et} \quad x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right).$$

Un choix de $x_0 > \sqrt{n}$ permet de s'assurer que la suite converge vers $\sqrt{n}$, et non $-\sqrt{n}$. Par ailleurs, on sait également estimer la vitesse de convergence de la suite.

Proposition 3.5

La vitesse de convergence de la suite $(x_k)_{k\in\mathbb{N}}$ est quadratique. Pour $x_0 > 0$, on a précisément

$$\left|\frac{x_{k+1} - \sqrt{n}}{\sqrt{n}}\right| < \frac{1}{2}\left(\frac{x_k - \sqrt{n}}{\sqrt{n}}\right)^2.$$

Autrement dit, à chaque itération dela méthode de Héron, on double le nombre de chiffres significatifs obtenus pour $\sqrt{n}$.

Démonstration : Adoptons un raisonnement un peu plus générique que le cas présent. Rappelons que l'on définit $f : x \mapsto x^2 - n$ de sorte que $f(\sqrt{n}) = 0$. Soit maintenant $x > \sqrt{n}$. La formule de Taylor–Lagrange nous assure alors qu'il existe $a \in [\sqrt{n}, x]$ tel que

$$0 = f(\sqrt{n}) = f(x) + f'(x) \cdot (\sqrt{n} - x) + \frac{f''(x)}{2} \cdot (\sqrt{n} - x)^2$$

Soit maintenant $x_k > \sqrt{n}$ la valeur obtenue à la k-ème itération de la méthode de Newton. Par définition, on a :

$$\begin{aligned} x_{k+1} - \sqrt{n} &= x_k - \frac{f(x_k)}{f'(x_k)} - \sqrt{n} \\ &= x_k - \frac{-f'(x_k) \cdot (\sqrt{n} - x_k) + \frac{f''(a)}{2} \cdot (\sqrt{n} - x_k)^2}{f'(x_k)} - \sqrt{n} \\ &= x_k + \sqrt{n} - x_k + \frac{f''(a)}{2f'(x_k)} (\sqrt{n} - x_k)^2 - \sqrt{n} \end{aligned}$$

Par conséquent,

$$|x_{k+1} - \sqrt{n}| = \left| \frac{f''(a)}{2f'(x_k)} \right| \cdot |x_k - \sqrt{n}|^2$$

Reste maintenant à majorer $|f''(a)|$ et minorer $|f'(x_k)|$. Dans notre cas, on a $f''(a) = 2$ et $f'(x_k) = 2x_k > 2\sqrt{n}$. On en déduit :

$$|x_{k+1} - \sqrt{n}| \leq \frac{1}{2\sqrt{n}} \cdot |x_k - \sqrt{n}|^2$$

ce qui équivaut au résultat annoncé. ■

Remarque 3.6

Le choix du terme initial x_0 de la suite récurrente est arbitraire. Cependant, on peut essayer d'éviter des calculs supplémentaires en prenant x_0 de l'ordre de grandeur de $\sqrt{n}$, et ce, sans faire de calcul. Par exemple, si n est codé sur m bits ($2^{m-1} \leq n < 2^m$), alors on peut choisir $x_0 = 2^{\lceil m/2 \rceil}$.

La condition d'arrêt de la méthode de Héron est la suivante. Notons a la racine carrée entière de n et r son reste. Comme les x_i sont toujours supérieurs à $\sqrt{n}$ (facile à démontrer), on observe que si $\lfloor x_k \rfloor = a$, alors $\lfloor x_k \rfloor^2 = a^2 \leq n$. D'autre part, si $\lfloor x_k \rfloor^2 > n$, alors $\lfloor x_k \rfloor$ ne peut pas être une racine carrée entière de a. La condition d'arrêt est donc le test $\lfloor x_k \rfloor^2 \leq n$.

Algorithme 8 : Méthode de Héron pour le calcul de racine carrée entière

Entrée : Un entier $n \in \mathbb{N}$
Sortie : Le plus grand entier $a \in \mathbb{N}$ tel que $a^2 \leq n$.

1 Noter m le nombre de bits de n.
2 Initialiser $x \leftarrow 2^{\lceil m/2 \rceil}$.
3 **Tant que** $\lfloor x \rfloor^2 > n$ **faire**
4 Calculer $x \leftarrow \frac{1}{2}(x + \frac{n}{x})$.
5 **Retourner** $\lfloor x \rfloor$.

Exemple 3.7

Pour calculer la racine carrée de $n = 3141592653589793$, *on obtient le résultat* $a = 56049912$ *en* 5 *itérations :*

78812208,6341
59336979,9265
56140958,3757
56049985,9908
56049912,164

La méthode élémentaire demande 26 *étapes.*

3.2 Racines carrées dans les corps finis

3.2.1 Définitions

Commençons par spécifier le problème à étudier.

Définition 3.8

Soit $a \in \mathbb{F}_q$. Une racine carrée de a est un élément $x \in \mathbb{F}_q$ tel que $x^2 = a$.

Problème 2

Étant donné $a \in \mathbb{F}_q$, calculer l'ensemble des racines carrées $x \in \mathbb{F}_q$ de a.

Remarquons au préalable que tout élément $a \in \mathbb{F}_q$ admet au plus deux racines carrées dans un corps fini (car un corps est intègre). Ces racines sont opposées l'une de l'autre : on peut donc se ramener au calcul d'**une** racine carrée.

Traitons dès à présent le cas où $\mathbb{F}_q$ est de caractéristique $p = 2$. Observons que dans ce cas, la racine carrée et son opposée sont identiques.

Proposition 3.9

Dans $\mathbb{F}_q$ avec $q = 2^k$, tout élément est un carré et toute racine carrée peut être extraite en $O(\log q)$ élévations au carré.

Démonstration : Si $q = 2^k$, alors pour tout $a \in \mathbb{F}_q$:

$$\left(a^{2^{k-1}}\right)^2 = a^{2^k} = a\,.$$

Ainsi, $x = a^{2^{k-1}}$ est l'unique racine carrée de a (unique car $x = -x$ sur $\mathbb{F}_q$) et se calcule en $O(\log(q))$ élévations au carré dans le corps $\mathbb{F}_q$:

$$a \;\mapsto\; a^2 \;\mapsto\; (a^2)^2 = a^{2^2} \;\mapsto\; ((a^2)^2)^2 = a^{2^3} \;\mapsto\; \cdots \mapsto a^{2^{k-1}}\,.$$

■

Comme une méthode a été trouvée en caractéristique $p := \mathrm{car}(\mathbb{F}_q) = 2$, dans toute la suite on suppose que $p \neq 2$. On notera également $q = p^k$.

3.2.2 Le cas $q \equiv 3 \bmod 4$.

Comme p est supposé impair, $q = p^k$ l'est aussi.

Lemme 3.10

Soit $a \in \mathbb{F}_q^\times$ avec q impair. Alors, a est un carré dans $\mathbb{F}_q$ si et seulement si $a^{(q-1)/2} = 1$.

Démonstration : Soit $\mathcal{C} = \{u \in \mathbb{F}_q^\times, u \text{ est un carré }\}$ et $\mathcal{R} = \{v \in \mathbb{F}_q, v^{(q-1)/2} = 1\}$. Montrons que $\mathcal{C} = \mathcal{R}$.

- Si $u \in \mathcal{C}$, alors $u = x^2$ donc $u^{(q-1)/2} = x^{q-1} = 1$. Donc $\mathcal{C} \subseteq \mathcal{R}$.
- Par ailleurs, $|\mathcal{R}| \leq \frac{q-1}{2}$ car ce sont les racines d'un polynôme de degré $\frac{q-1}{2}$. D'autre part, $|\mathcal{C}| \geq \frac{q-1}{2}$, car l'application $t \mapsto t^2$ a au plus 2 antécédents ($\mathbb{F}_q$ est un corps).

■

Comme corollaire immédiat du Lemme 3.10, pour tout carré $a \in \mathbb{F}_q$, on a donc

$$a^{(q+1)/2} = a\,.$$

Cela nous mène à observer une situation favorable pour le calcul de racine carrée : dans le cas où $q \equiv 3 \bmod 4$, l'entier $\frac{q+1}{2}$ est pair, donc on peut calculer $x = a^{(q+1)/4}$. On a alors $x^2 = a^{(q+1)/2} = a$.

Proposition 3.11

Si $q \equiv 3 \bmod 4$, alors on peut calculer les racines carrées d'un carré $a \in \mathbb{F}_q$ en $O(\log q)$ opérations dans $\mathbb{F}_q$.

Démonstration : D'après ce qui précède, il suffit de calculer l'exponentiation $a^{(q+1)/4}$, ce qui s'effectue en $\leq \lceil \log_2(q) \rceil$ multiplications et carrés. ■

3.2.3 Autres cas spécifiques

Dans la même veine que le cas $q \equiv 3 \mod 4$, il existe des algorithmes spéciaux pour extraire une racine carrée lorsque :

— $q \equiv 5 \mod 8$: c'est l'algorithme d'Atkin [Atk92], voir exercices ;
— $q \equiv 9 \mod 16$: par Müller, puis Kong-Cai-Yu-Li [KCY+06].

Ces algorithmes s'exécutent également en $O(\log(q))$ opérations dans le corps $\mathbb{F}_q$.

Néanmoins, à l'heure actuelle il n'y a pas d'algorithme déterministe efficace permettant de réoudre le problème de l'extraction de racine carrée dans tous les cas où $q \equiv 1 \mod 16$. Nous allons donc découvrir, dans la prochaine section, deux algorithmes probabilistes calculant des racines carrées dans un cadre général.

3.2.4 Algorithmes génériques

Avant d'introduire ces algorithmes, effectuons quelques rappels sur la résiduosité quadratique.

3.2.4.1 Rappels

Symbole de Legendre. Le symbole de Legendre $\left(\frac{a}{p}\right)$ permet de representer le fait qu'un entier a soit (ou non) un carré modulo un nombre premier p.

Définition 3.12 (*Symbole de Legendre*)

Soit p un nombre premier et $a \in \mathbb{Z}/p\mathbb{Z}$. Le symbole de Legendre de a modulo p est

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & \text{si } a \text{ est un carré dans } \mathbb{F}_p, \\ -1 & \text{sinon.} \end{cases}$$

La résiduosité quadratique est une propriété multiplicative : le produit de deux carrés est un carré. Par ailleurs, modulo un nombre premier p, le produit de deux non-carrés devient un carré. On a donc $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$. D'autres propriétés de la résiduosité quadratique sont énoncées dans les deux propriétés suivante.

Algorithme 9 : Une méthode de calcul du symbole de Legendre

Entrée : $a \in \mathbb{Z}$ et $p \geq 2$ premier
Sortie : le symbole de Legendre $\left(\frac{a}{p}\right)$

```
Si a = 0
    Retourner 0
Si a = 1 ou p = 2
    Retourner 1
Si a = p − 1
    Si a ≡ 0 mod 4
        Retourner 1
    Sinon
        Retourner −1
Si a ≡ 0 mod 2
    Si p ≡ 1 mod 8 ou p ≡ 3 mod 8
        Retourner Legendre(a/2, p)
    Sinon
        Retourner (−1)× Legendre(a/2, p)
Si a ≥ p
    Retourner Legendre(a mod p,p)
Si a ≡ 1 mod 4 ou p ≡ 1 mod 4
    Retourner Legendre(p, a)
Sinon
    Retourner (−1)× Legendre(p, a)
```

Proposition 3.13

Soit p premier, et a, b deux entiers. Alors

1. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$,
2. $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$,
3. $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$,
4. *si a est impair, alors* $\left(\frac{a}{p}\right) = (-1)^t$ *où* $t = \sum_{j=0}^{(p-1)/2} \lfloor \frac{aj}{p} \rfloor$.

Proposition 3.14 (*Loi de réciprocité quadratique*)

Si p et q sont premiers, alors

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{(p-1)(q-1)/4}$$

Ces résultats permettent d'aboutir à un algorithme qui calcule le symbole de Legendre $\left(\frac{a}{p}\right)$ en $O(\log a \log p)$ opérations binaires, soit un temps quadratique en la taille de l'entrée. Voir Algorithme 9 pour les détails.

Symbole de Jacobi. Par multiplicativité, on peut étendre la définition du symbole de Legendre au cas où le module n'est plus premier.Pour cela on définit par induction

$$\left(\frac{a}{n_1 n_2}\right) := \left(\frac{a}{n_1}\right)\left(\frac{a}{n_2}\right).$$

Le symbole obtenu est appelé *symbole de Jacobi*. Dans ce cas, si a est un carré modulo n, alors $\left(\frac{a}{n}\right) = 1$, mais la réciproque n'est plus vraie.

Résiduosité dans $\mathbb{F}_q$. Enfin, pour tester si un élément est un carré dans $\mathbb{F}_q$, avec q non premier, il n'existe pas de symbole de Legendre ou de Jacobi. Néanmoins, on peut se ramener au sous-corps premier de $\mathbb{F}_q$ pour effectuer ce test.

Proposition 3.15

Soit $q = p^k$, avec $k \geq 1$. Il existe un algorithme permettant de tester si $a \in \mathbb{F}_q$ est un carré en $O(\log(q))$ opérations dans $\mathbb{F}_q$ et $O(\log(p)^2)$ opérations binaires.

Démonstration : Le test se résume à calculer une norme $\mathbb{F}_q/\mathbb{F}_p$ et un symbole de Legendre modulo p. Voir détails en exercice. ■

3.2.4.2 Algorithme de Cipolla

Sans cette section, nous présentons un premier algorithme pour l'extraction de racines carrées dans $\mathbb{F}_q$: l'algorithme de Cipolla. Cet algorithme s'éxécute uniquement dans le cas où q est **impair**, ce que nous supposons dorénavant.

Rappelons quelques définitions concernant les extensions quadratiques de corps.

Définition 3.16

Soit $\mathbb{F}_{q^2} = \mathbb{F}_q(\beta)$ une extension quadratique de $\mathbb{F}_q$. Pour $z = u + \beta v \in \mathbb{F}_{q^2}$, on définit :
- *sa trace comme $\mathrm{Tr}(z) := z + z^q \in \mathbb{F}_q$, qui vérifie alors $\mathrm{Tr}(z) = 2u + \mathrm{Tr}(\beta)v$;*
- *sa norme comme $\mathrm{N}(z) := z^q z = z^{q+1} \in \mathbb{F}_q$, qui vérifie alors $\mathrm{N}(z) = u^2 + \mathrm{Tr}(\beta)uv + \mathrm{N}(\beta)v^2$.*

L'algorithme de Cipolla repose alors sur la remarque suivante.

Remarque 3.17

Pour $z \in \mathbb{F}_{q^2}$, si $\mathrm{N}(z)$ est un carré, alors les racines carrées de $\mathrm{N}(z)$ sont $\pm z^{(q+1)/2}$, et se calculent donc en $O(\log(q))$ opérations dans $\mathbb{F}_{q^2}$.

Pour calculer une racine carrée de $a \in \mathbb{F}_q$, une idée est alors d'essayer d'écrire a comme la norme d'un élément de $\mathbb{F}_{q^2}$. Notons que l'objectif est raisonnable car l'application norme $\mathrm{N} : \mathbb{F}_{q^2} \to \mathbb{F}_q$ est surjective.

Pour obtenir l'écriture de a comme $N(z)$ avec $z = \mathbb{F}_{q^2}$, on va choisir une réprésentation de $\mathbb{F}_{q^2}/\mathbb{F}_q$ de la forme $\mathbb{F}_q[X]/(X^2 - c)$, où c n'est pas un carré dans $\mathbb{F}_q$. Autrement dit, on choisit c de sorte que $P(X) = X^2 - c$ soit irréductible dans $\mathbb{F}_q[X]$.

Alors les racines de $P(X)$ dans $\mathbb{F}_{q^2}$ sont deux élements conjugués β et β^q, et on a donc $P(X) = (X - \beta)(X - \beta^q) = X^2 - \mathrm{Tr}(\beta)X + \mathrm{N}(\beta)$. Autrement dit,

$$\mathrm{Tr}(\beta) = 0 \quad \text{et} \quad \mathrm{N}(\beta) = -c\,.$$

Pour ce choix de représentation de l'exetsnion quadratique $\mathbb{F}_{q^2}$, les normes de certains éléments s'écrivent très simplement. En effet, si $z \in \mathbb{F}_{q^2}$ est de la forme $z = u + \beta$ où $u \in \mathbb{F}_q$, alors on a

$$\mathrm{N}(z) = u^2 + \mathrm{Tr}(\beta)u + \mathrm{N}(\beta) = u^2 - c\,.$$

L'idée de l'algorithme de Cipolla est maintenant la suivante : si u et c ont été choisis de sorte que $u^2 - a$ soit un non-carré c de $\mathbb{F}_q$, alors on a exprimé a comme une norme $N(z)$.

Algorithme 10 : Algorithme de Cipolla

Entrée : $a \in \mathbb{F}_q$ un carré
Sortie : $x \in \mathbb{F}_q$ tel que $x^2 = a$

1 Tirer $u \in \mathbb{F}_q$ et calculer $c = u^2 - a$.
2 **Si** *c est un carré*
3 Revenir à l'étape 1
4 **Sinon**
5 Construire $\mathbb{F}_{q^2} = \mathbb{F}_q[X]/(X^2 - c)$ et noter β la classe de X.
6 **Retourner** $x = (u + \beta)^{(q+1)/2}$.

Remarque 3.18

On laisse au lecteur le soin de chercher à quel moment la condition q impair est nécessaire à la réussite de l'algorithme.

Rappelons que tester si c est un carré dans $\mathbb{F}_q$ peut être réduit à un calcul de norme dans $\mathbb{F}_q/\mathbb{F}_p$ puis un calcul de symbole de Legendre. On peut également montrer que le nombre de tirages de u à effectuer est constant. [plus de détails à venir sur ce point] On a donc le résultat de complexité suivant.

Proposition 3.19

L'algorithme de Cipolla calcule une racine carrée dans $\mathbb{F}_q$, avec $q = p^k$ impair, avec en moyenne $O(\log q)$ opérations dans $\mathbb{F}_{q^2}$ et $O(\log q \log p)$ opérations dans $\mathbb{F}_q$.

Exemple 3.20

Voici un exemple d'éxecution de l'algorithme de Cipolla, avec $q = 100000007$ (premier) et $a = 40598710$.

1. Premier tirage : $u = 9871141$ *donne* $c = 77222420$, *mais c est un carré.*
2. Second tirage : $u = 19298050$ *donne* $c = 67134768$, *et c n'est pas un carré.*

On construit donc $P(X) = X^2 - c = X^2 + 32865239$, puis on calcule dans $\mathbb{F}_q[X]/(P)$ la valeur de $(u + X)^{50000004}$.
On obtient le résultat $x = 31460202$, et on peut vérifier que

$$31460202^2 \equiv 40598710 \mod 100000007\,.$$

3.2.4.3 Algorithme de Tonelli–Shanks

Pas vu en cours ; détails à venir.

3.3 Racines carrées dans $\mathbb{Z}/N\mathbb{Z}$

On se place maintenant dans l'anneau $\mathbb{Z}/N\mathbb{Z}$, avec $N = \prod_{i=1}^{k} p_i^{e_i}$ non premier. La définition d'une racine carrée dans cet anneau reste naturelle :

Définition 3.21

Soit $a \in \mathbb{Z}/N\mathbb{Z}$. Une racine carrée de a modulo N est un entier $x \in \mathbb{Z}/N\mathbb{Z}$ tel que $x^2 \equiv a \mod N$.

Néanmoins, il faut prendre garde que le nombre de racines carrées n'est plus limité à 2 : il peut atteindre 2^k où k le nombre de premiers distincts divisant N.

Pour calculer une racine carrée modulo N, notre stratégie sera la suivante.

1. Pour tout i, calculer les racines carrées de a modulo p_i.
2. En déduire, pour tout i, les racines carrées de a modulo $p_i^{e_i}$.
3. En déduire les racines carrées de $N = \prod_{i=1}^{k} p_i^{e_i}$.

L'étape 1 a été traitée dans la section précédente. L'étape 2 sera traitée dans les Sections 3.3.2 (p_i impair) et 3.3.3 ($p_i = 2$). L'étape 3 est traitée dans la section qui suit.

3.3.1 Le cas $N = N_1N_2$, avec N_1 et N_2 premiers entre eux

Dans le cas où N est le produit de deux nombres premiers entre eux, l'idée est d'utiliser le théorème des restes chinois pour reconstruire les racines carrées modulo N à partir des racines carrées modulo N_1 et N_2.

Proposition 3.22

Soit $N = N_1N_2$ où N_1 et $N_2 \geq 2$ sont deux entiers que l'on suppose premiers entre eux. Soient $u, v \in \mathbb{Z}$ tels que $uN_1 + vN_2 = 1$. Soit enfin $a \in \{0, \ldots, N-1\}$ un carré modulo N. Alors,

1. *l'entier a est également un carré dans $\mathbb{Z}/N_1\mathbb{Z}$ et dans $\mathbb{Z}/N_2\mathbb{Z}$;*
2. *si $\mathcal{R}(a, N_1)$ et $\mathcal{R}(a, N_2)$ forment les ensembles des racines carrées de a dans $\mathbb{Z}/N_1\mathbb{Z}$ et dans $\mathbb{Z}/N_2\mathbb{Z}$, alors a admet comme racines carrées modulo N l'ensemble :*

$$\mathcal{R}(a, N) = \{N_1 \cdot u \cdot x_2 + N_2 \cdot v \cdot x_1 \mid x_1 \in \mathcal{R}(a, N_1), x_2 \in \mathcal{R}(a, N_2)\} \subseteq \mathbb{Z}/N\mathbb{Z}\,.$$

Démonstration : Soit r une racine carrée de a dans $\mathbb{Z}/N\mathbb{Z}$. On a $a = r^2 + kN_1N_2$ pour $k \in \mathbb{Z}$, donc $a = r^2$ dans $\mathbb{Z}/N_1\mathbb{Z}$ et dans $\mathbb{Z}/N_2\mathbb{Z}$.
Si r s'écrit sous la forme $N_1ux_2 + N_2vx_1$, alors on trouve

$$r^2 \equiv x_1^2 \equiv a \mod N_1 \quad \text{et} \quad r^2 \equiv r_2^2 \equiv a \mod N_2\,.$$

D'après le théorème des restes chinois, ces valeurs sont les uniques racines de a dans $\mathbb{Z}/N\mathbb{Z}$. ■

3.3.2 Le cas $N = p^k$, avec $p \neq 2$

Dans le cas où N est la puissance d'un nombre premier p, on ne peut plus appliquer le théorème des restes chinois.

L'idée est alors de « remonter » les racines carrées de a modulo p en des racines carrées modulo p^2, p^3, jusqu'à $N = p^k$. Cette technique s'apparente à une méthode plus générale issue du lemme de Hensel (*Hensel lifting lemma*) que nous reverrons dans les chapitres suivants.

Soit $a \in \mathbb{Z}/N\mathbb{Z}$ dont on cherche une racine carrée. Supposons pour l'instant que a et N (donc a et p) sont premiers entre eux.

Lemme 3.23

Soit $p \geq 3$ premier et $a \in \{0, \ldots, p^i - 1\}$ premier avec p, pour un certain $i \geq 2$. Si $x \in \{0, \ldots, p^i - 1\}$ est une racine carrée de a modulo p^i, alors x s'écrit $y + tp^{i-1}$ avec
- *un entier $y \in \{0, \ldots, p^{i-1} - 1\}$ qui forme une racine carrée de a modulo p^{i-1}, et*
- *un entier $t \in \{0, \ldots, p-1\}$ tel que*

$$t = \frac{a - y^2}{p^{i-1}} \times ((2y)^{-1} \mod p).$$

Démonstration : Écrivons de manière unique $x = y + tp^{i-1}$ avec $y < p^{i-1}$ et $t < p$. En élevant au carré on obtient

$$x^2 = y^2 + p^{i-1}(2ty + t^2 p^{i-1}).$$

On note alors que

$$x^2 \equiv a \mod p^i \iff y^2 \equiv a \mod p^{i-1}.$$

Trouvons maintenant les valeurs de t possibles. Si $x^2 \equiv a \mod p^i$, alors il résulte que

$$\frac{a - y^2}{p^{i-1}} \equiv 2ty \mod p \tag{3.1}$$

Comme $\text{pgcd}(a, p) = 1$, les entiers y^2 et p (donc y et p) sont également premiers entre eux.
Donc l'unique solution de (3.1) est

$$t = \frac{a - y^2}{p^{i-1}} \times ((2y)^{-1} \mod p).$$

■

Dans le cas où a et p ne sont pas premiers entre eux, la remontée de solution est plus aisée.

Lemme 3.24

Soit a un entier et p un nombre premier tels que $a = pa'$. Soit également $i \geq 3$. On a les deux résultats suivants
1. *S'il existe $x \in \{0, \ldots, p^i - 1\}$ satisfaisant $x^2 \equiv a \mod p^i$, alors p^2 divise a et p divise x.*
2. *S'il existe $x' \in \{0, \ldots, p^{i-2} - 1\}$ satisfaisant $x'^2 \equiv a' \mod p^{i-2}$, alors pour tout $j \in \{0, \ldots, p-1\}$, on a $(px' + jp^{i-1})^2 \equiv p^2 a' \mod p^i$.*

Autrement dit, si a est un multiple de p qui admet une racine carrée x modulo p^i, alors ses racines carrées sont de la forme $p(x' + jp^{i-2})$, où x' est une racine de a/p^2 (qui est bien un entier) et $j \in \{0, \ldots, p-1\}$.

Démonstration :
1. Écrivons $a = x^2 + kp^i$ avec $k \in \mathbb{Z}$. Aors $x^2 = vp - kp^i$, donc p divise x puis en écrivant $x = up$ on a $u^2p^2 = vp - kp^i$. Comme $i \geq 2$, ceci implique que p^2 divise $vp = a$.
2. On vérifie simplement que pour tout $j \in \{0, \ldots, p-1\}$,

$$(px' + jp^{i-1})^2 = p^2 x'^2 + 2jx'p^i + j^2 p^{2i-2} \equiv p^2 a' \mod p^i,$$

car p^i divise p^{2i-2} ($i \geq 3$). ■

Remarque 3.25

On vérifie que si $a = 0$, les racines carrées de 0 modulo p^i sont bien tous les multiples de $p^{\lceil i/2 \rceil}$.

L'Algorithme 11 utilise les deux lemmes précédents pour calculer toutes les racines carrées de a modulo p^k, $p \neq 2$.

Algorithme 11 : Calcul de l'ensemble des racines carrées modulo $N = p^k$

Entrée : p premier impair, $k \geq 1$ et a un carré modulo p^k
Sortie : l'ensemble des $x \in \{0, \ldots, p^k - 1\}$ tels que $x^2 \equiv a \mod p^k$

Si $a = 0$
  Retourner $\{jp^{\lceil k/2 \rceil} \mid j \in \{0, \ldots, p^{\lfloor k/2 \rfloor} - 1\}$.
$r \leftarrow 1$
$a' \leftarrow a$
$k' \leftarrow k$
Tant que p *divise* a' **faire**
  $a' \leftarrow a'/p^2$
  $r \leftarrow rp$
  $k' \leftarrow k' - 2$
Calculer $\mathcal{X} = \{-x, x\}$ les deux racines de carrées de a' modulo p (par exemple avec l'algorithme de Cipolla)
Initialiser $\mathcal{R} = \varnothing$
Pour tout $x \in \mathcal{X}$ **faire**
  Pour tout i *allant de* 2 *à* k' **faire**
    Calculer $z \leftarrow (a' - x^2)/p^{i-1}$ dans $\mathbb{Z}$
    Calculer $t \leftarrow z \times (2x)^{-1} \mod p$
    Calculer $x \leftarrow x + tp^{i-1}$
  Ajouter x à $\mathcal{R}$.
Retourner $\{ r(x + jp^{k'}) \mid x \in \mathcal{R}, j \in \{0, \ldots, p^{(k-k')/2} - 1\} \}$.

Exemple 3.26

Todo: faire un joli example

3.3.3 Le cas $N = 2^k$

Pour $N = 2^k$, on raisonne de manière similaire.

On traite d'abord les petits cas, qui se trouvent être particuliers.

1. Dans $\mathbb{Z}/2\mathbb{Z}$:

x	0	1
x^2	0	1

2. Dans $\mathbb{Z}/4\mathbb{Z}$:

x	0	1	2	3
x^2	0	1	0	1

3. Dans $\mathbb{Z}/8\mathbb{Z}$:

x	0	1	2	3	4	5	6	7
x^2	0	1	4	1	0	1	4	1

Pour le cas a pair, on utilise le Lemme 3.24 (également valable pour $p = 2$) afin d'« éliminer » la partie paire de a. On suppose maintenant a impair.

Lemme 3.27

Soient $k \geq 2$, a un nombre impair et r tel que $r^2 \equiv a \mod 2^k$. Alors, l'équation $x^2 \equiv a \mod 2^{k+1}$ admet exactement 4 solutions

$$x \in \{r,\ -r,\ r - 2^{k-1},\ -r + 2^{k-1}\}.$$

Démonstration : On raisonne par induction. C'est vrai pour $k = 2$ d'après le tableau précédent.
Soit maintenant $x = u + \epsilon 2^k$ où $0 \leq u < 2^k$ et $\epsilon \in \{0,1\}$.
Si $x^2 \equiv a \mod 2^{k+1}$, alors on a $u^2 \equiv a \mod 2^k$. Autrement dit, les solutions à l'ordre $k+1$ ne peuvent être que :

$$r,\ -r, 2^{k-1} - r, r - 2^{k-1} \qquad \text{(solutions à l'ordre } k\text{)}$$

et

$$r + 2^k,\ -r + 2^k, 2^{k-1} - r + 2^k, r - 2^{k-1} + 2^k\,.$$

En raisonnant modulo 2^{k+1}, on peut réécrire cet ensemble comme

$$\{r, -r, -r + 2^k, r - 2^k, 2^{k-1} - r, -r + 2^{k-1}, r + 2^{k-1}, -r - 2^{k-1}\}\,.$$

On vérifie par un calcul simple que les 4 premières valeurs sont solutions dans $\mathbb{Z}/p^{k+1}\mathbb{Z}$. On va voir dans le lemme suivant que les 4 dernières solutions ne le sont pas. ■

Lemme 3.28

Si a est impair et x tel que $x^2 \equiv a \mod 2^k$, alors :

$$x^2 \equiv a \mod 2^{k+1} \iff (x + 2^{k-1})^2 \not\equiv a \mod 2^{k+1}\,.$$

Démonstration : On a

$$(x + 2^{k-1})^2 \equiv x^2 + 2^k x + 2^{2k-2} \equiv x^2 + 2^k x \equiv x^2 + 2^k \mod 2^{k+1}$$

(pour la dernière équivalence, voir que a impair $\implies$ x impair).
Si $x^2 \equiv a \mod 2^{k+1}$, alors

$$(x + 2^{k-1})^2 \equiv a + 2^k \not\equiv a \mod 2^{k+1}\,.$$

Si $x^2 \not\equiv a \mod 2^{k+1}$, alors, comme $x^2 \equiv a \mod 2^k$, on a

$$x^2 \equiv a + 2^k \mod 2^{k+1}\,.$$

Puis,

$$(x + 2^{k-1})^2 \equiv a + 2^k + 2^k \equiv a \mod 2^{k+1}\,.$$

■

L'Algorithme 13 résume la méthode d'extraction de racines carrées modulaires lorsque $N = 2^k$.

Enfin, terminons par écrire l'algorithme qui calcule l'ensemble des racines carrées dans le cas général.

Algorithme 12 : Calcul de l'ensembl des racines carrées modulo $N = 2^k$

Entrée : p premier impair, $k \geq 1$ et a un carré modulo 2^k
Sortie : l'ensemble des $x \in \{0, \ldots, 2^k - 1\}$ tels que $x^2 \equiv a \mod 2^k$

Si $a = 0$
  Retourner $\{j2^{\lceil k/2 \rceil} \mid j \in \{0, \ldots, 2^{\lfloor k/2 \rfloor} - 1\}$.
Si $k = 1$
  Retourner $\{a\}$
Si $k = 2$ *et* $a = 1$
  Retourner $\{1, 3\}$
Si $k = 3$ *et* $a = 1$
  Retourner $\{1, 3, 5, 7\}$
Si $k = 3$ *et* $a = 4$
  Retourner $\{2, 6\}$
$r \leftarrow 1$
$a' \leftarrow a$
$k' \leftarrow k$
Tant que p *divise* a' **faire**
  $a' \leftarrow a'/4$
  $r \leftarrow 2r$
  $k' \leftarrow k' - 2$
Si $k' \leq 3$
  Faire un appel récursif avec $a = a'$ et $k = k'$, et obtenir un ensemble $\mathcal{S}$.
  Retourner $\{r(x + j2^{k'}) \mid x \in \mathcal{S}, j \in \{0, \ldots, p^{(k-k')/2} - 1\}$
Sinon
  Poser $x = 1$
  Pour tout i *allant de* 3 *à* $k' - 1$ **faire**
    Si $x^2 \not\equiv a' \mod 2^{i+1}$
      $x \leftarrow x + 2^{i-1}$
  $\mathcal{S} = \{x, -x, x - 2^{k'-1}, 2^{k'-1} - x\}$
  Retourner $\{ r(x + jp^{k'}) \mid x \in \mathcal{S}, j \in \{0, \ldots, p^{(k-k')/2} - 1\} \}$.

Algorithme 13 : Calcul de l'ensemble des racines carrées modulo $N = \prod_{i=1}^{m} p_i^{k_i}$

Rzemarque : la procédure `SQRT` calcule une racine carrée modulo une puissance de nombre premier (voir Algorithmes 11 et 13), et la procédure `CRT` effectue la reconstruction par restes chinois de la Proposition 3.22.

Entrée : $\mathcal{F} = \{(p_1, k_1), \ldots, (p_m, k_m)\}$ la factorisation d'un entier N, et a un carré modulo N
Sortie : l'ensemble des $x \in \{0, \ldots, N - 1\}$ tels que $x^2 \equiv a \mod N$

$\mathcal{R} \leftarrow \texttt{SQRT}(a, p_1^{k_1})$
$N' \leftarrow p_1^{k_1}$ **Pour tout** $i = 2, \ldots, m$ **faire**
  $\mathcal{L} \leftarrow \texttt{SQRT}(a, p_i^{k_i})$
  $\mathcal{R} \leftarrow \texttt{CRT}(\mathcal{R}, \mathcal{L}, N', N' \times p_i^{k_i})$
  $N' \leftarrow N' \times p_i^{k_i}$
Retourner $\mathcal{R}$

Chapitre 4

Factorisation de polynômes

Ce chapitre est dédié à la factorisation de polynômes à une variable dans différentes structures algébriques : les corps finis dans la Section 4.1, les rationnels et les entiers dans la Section 4.2.

4.1 Factorisation de polynômes dans les corps finis

Soit $\mathbb{F}_q[X]$ l'algèbre des polynômes à une variable sur le corps fini $\mathbb{F}_q$. Posons d'abord le problème de la factorisation de tels polynômes.

Définition 4.1 (*Factorisation en polynômes irréductibles*)

Soit $P \in \mathbb{F}_q[X]$ de degré $n \geq 1$. Une factorisation de P en polynômes irréductibles est une écriture

$$P = \alpha \prod_{i=1}^{k} P_i^{m_i}$$

où les $P_i \in \mathbb{F}_q[X]$ sont unitaires, irréductibles et deux-à-deux distincts, et où les m_i sont des entiers strictement positifs. Pour tout $i \in \{1, \ldots, k\}$, l'entier $m_i \geq 1$ est appelé la multiplicité du facteur P_i. L'élément $\alpha^\times \in \mathbb{F}_q$ est appelé le coefficient dominant de P.

Exemple 4.2

Dans $\mathbb{F}_5[X]$, le polynôme $P(X) = 2X^{11} + 3X^{10} + 2X^7 + X^6 + 3X^5 + 4X^4 + 3X^2 + 2X + 3$ se factorise comme $2 \times (X+1)^3 \times (X^2 + 2X + 3)^2 \times (X^4 + 2X^3 + X^2 + 3X + 1)$, autrement dit :

$$\alpha = 2 \quad et \quad \begin{cases} P_1(X) = X + 1 & avec\ m_1 = 3 \\ P_1(X) = X^2 + 2X + 3 & avec\ m_2 = 2 \\ P_1(X) = X^4 + 2X^3 + X^2 + 3X + 1 & avec\ m_3 = 1 \end{cases}$$

Remarque 4.3

L'anneau $\mathbb{F}_q[X]$ est factoriel, donc toute factorisation en polynôme irréductible est unique à l'ordre près des facteurs.

L'objectif de cette section est de calculer efficacement une factorisation en polynômes irréductibles, autrement dit de résoudre le problème suivant.

Problème 3

Étant donné un polynôme $P \in \mathbb{F}_q[X]$ de degré $n \geq 1$, trouver une séquence $[(P_1, m_1), \ldots, (P_k, m_k)]$ où les $P_i \in \mathbb{F}_q[X]$ sont unitaires, irréductibles et deux-à-deux distincts, les m_i sont des entiers

strictement positifs, ainsi qu'un élément $\alpha \in \mathbb{F}_q^\times$, tels que

$$P(X) = \alpha \prod_{i=1}^{k} P_i(X)^{m_i}$$

Si l'on dispose d'un algorithme résolvant le Problème 3, on mesurera sa complexité en fonction de la taille de l'entrée, qui ici est $O(n \log q)$. En effet, le polynôme $P(X)$ est constitué de $n+1$ polynômes dont les coefficients sont de taille $\log q$ bits. Notons que la taille de la sortie est également en $O(n \log q)$.

Pour évaluer cette complexité, nous noterons $\mathsf{M}(n)$ une borne supérieure sur la complexité (en nombre d'opérations élémentaires sur $\mathbb{F}_q$) de la multiplication de deux polynômes de degré $\leq n$. Alors, on rappelle que :

— $n = o(\mathsf{M}(n))$ en général,
— $\mathsf{M}(n) = O(n^2)$ par une multiplication naïve, $\mathsf{M}(n) = O(n^{1,59})$ grâce à l'algorithme de Karatsuba et $\mathsf{M}(n) = O(n \log n)$ avec des algorithmes avancés dans certains corps finis,
— la division euclidienne d'un polynôme de degré n par un polynôme de degré $\leq n$ a un coût $O(\mathsf{M}(n))$,
— les additions et produits modulo un polynôme de degré n ont un coût $O(\mathsf{M}(n))$,
— l'algorithme d'Euclide évalué sur deux polynômes de degré $\leq n$ requiert $O(n^2)$ opérations, mais il existe des améliorations pour calculer un pgcd étendu (donc une inversion modulaire) en $O(\mathsf{M}(n) \log n)$ opérations.

Dans la suite, nous aurons besoin des notions de *facteur propre* et de *polynôme sans carré*.

Définition 4.4

Soient P, Q deux polynômes. On dit que Q est un facteur propre *(ou* diviseur propre *de P si Q divise P et* $\deg(Q) \notin \{0, \deg P\}$.

Définition 4.5

Soit $P \in \mathbb{F}_q[X]$. On dit que le polynôme P est sans carré *si la multiplicité m_i de tous ses facteurs irréductibles P_i est égale à 1. Autrement dit, P est un polynôme* sans-carré *si pour tout facteur propre Q de P, le polynôme Q^2 ne divise pas P.*

Si P est un polynôme quelconque, la partie sans carré *de P est alors le plus grand polynôme unitaire sans carré qui divise P.*

Exemple 4.6

Si on reprend l'Exemple 4.2, la partie sans carré de $P(X)$ est

$$\begin{aligned}\widetilde{P}(X) &= (X+1)(X^2+2X+3)(X^4+2X^3+X^2+3X+1) \\ &= X^7+2X^5+4X^4+X^3+X^2+4X+3\end{aligned}$$

Stratégie de résolution. Pour résoudre le Problème 3, nous allons adopter la stratégie suivante :

1. extraire le coefficient dominant α de P,
2. extraire de P sa partie sans carré $\widetilde{P}$,

3. extraire de $\widetilde{P}$ un facteur propre de P (étape à itérer),
4. calculer la multiplicité des facteurs irréductibles obtenus.

La première étape de la stratégie de résolution est immédiate : il suffit de lire le coefficient de plus haut degré de P.

De même, la quatrième étape s'effectue efficacement : si P_i est un facteur irréductible de P, il suffit (par exemple) de chercher sa multiplicité m_i en divisant successivement P par P_i. Pour ce faire, notons que tester si un polynôme Q divise un polynôme P correspond à effectuer une division euclidienne entre ces deux polynômes, ce qui se réalise donc en $O(\mathsf{M}(n)\log n)$ opérations sur $\mathbb{F}_q$. On obtient donc une complexité en pire cas en $O(n\mathsf{M}(n)\log n)$ opérations sur $\mathbb{F}_q$ pour cette quatrième étape.

Il nous reste donc à nous intéresser aux étapes 2. et 3. de la stratégie donnée ci-dessus.

4.1.1 Extraction de la partie sans carré

Formalisons le sous-problème de l'extraction de la partie sans carré d'un polynôme.

Problème 4 (*Extraction de la partie sans carré d'un polynôme sur les corps finis*)

Étant donné un polynôme $P = \prod_{i=1}^{k} P_i^{m_i} \in \mathbb{F}_q[X]$ où les P_i sont unitaires, irréductibles et deux-à-deux distincts, calculer la partie sans carré $\widetilde{P} = \prod_{i=1}^{k} P_i$ de P.

Dans le but d'extraire la partie sans carré d'un polynôme P, une idée importante est de comparer P avec son polynôme dérivé P', par un calcul de pgcd.

En effet, écrivons

$$P'(X) = \sum_{i=1}^{k} m_i P_i'(X) \frac{P(X)}{P_i(X)} \tag{4.1}$$

et plaçons-nous d'abord dans le cas favorable où, pour tout $i \in \{1,\dots,k\}$, l'entier $m_i \neq 0$ dans $\mathbb{F}_q$ (autrement dit, la caractéristique p de $\mathbb{F}_q$ ne divise aucun des m_i). On note alors que pour tout $j \in \{1,\dots,k\}$, le polynôme $P_j(X)^{m_j-1}$ divise chacun des quotients $\frac{P(X)}{P_i(X)}$, mais que $P_j(X)^{m_j}$ ne divise pas $P_i'(X)\frac{P(X)}{P_i(X)}$ lorsque $i = j$. Il résulte que

$$\operatorname{pgcd}(P(X), P'X)) = \prod_{i=1}^{k} P_i(X)^{m_i-1}\,.$$

On a alors (dans ce cas favorable) :

$$\widetilde{P}(X) = \frac{P(X)}{\operatorname{pgcd}(P(X), P'X))}\,.$$

Dans le cas où $p \coloneqq \operatorname{car}(\mathbb{F}_q)$ divise l'un des m_i, la situation est un peu plus complexe. Notons

$$I \coloneqq \{i \in \{1,\dots,k\},\ p \text{ divise } m_i\}\,.$$

Pour $i \in I$, on a $m_i P_i'(X)\frac{P(X)}{P_i(X)} = 0$ dans $\mathbb{F}_q[X]$, donc le polynôme $P_i(X)^{m_i}$ divise tous les termes de la somme définissant $P'(X)$ dans l'équation (4.1). Par conséquent, $P_i(X)^{m_i}$ divise également $\operatorname{pgcd}(P(X), P'(X))$.

D'un autre côté, si $i \notin I$, alors (comme précédemment) le polynôme $P_i(X)^{m_i-1}$ divise $\operatorname{pgcd}(P(X), P'(X))$. Le résultat suivant résume la situation.

Lemme 4.7

Soit $P(X) = \prod_{i=1}^{k} P_i(X)^{m_i} \in \mathbb{F}_q[X]$ *et* $I = \{i \in \{1, \ldots, k\},\ p \text{ divise } m_i\}$ *où* $p = \text{car}(\mathbb{F}_q)$. *Alors :*

$$\text{pgcd}(P(X), P'(X)) = \prod_{i=1}^{k} P_i^{m_i - \delta_i}$$

où $\delta_i = 0$ *si* $i \in I$ *et* $\delta_i = 1$ *sinon.*

Par conséquent, dans le cas général le polynôme

$$U(X) := \frac{P(X)}{\text{pgcd}(P(X), P'(X))} = \prod_{i \notin I} P_i(X)\,.$$

n'est pas nécessairement la partie sans carré de $P(X)$. C'en est néanmoins un diviseur, et il nous reste à déterminer $\prod_{i \in I} P_i(X)$.

Pour obtenir ce facteur, calculons d'abord

$$V(X) := \frac{P(X)}{\text{pgcd}(U(X)^{\deg P}, P(X))} = \frac{P(X)}{\prod_{i \notin I} P_i(X)^{m_i}} = \prod_{i \in I} P_i(X)^{m_i}\,.$$

Comme les éléments $i \in I$ sont tels que $p \mid m_i$, on a :

$$V(X) = (\prod_{i=1}^{k} P_i(X)^{m_i/p})^p$$

Supposons un instant que l'on arrive à calculer $W(X) := \prod_{i=1}^{k} P_i(X)^{m_i/p}$ à partir de $V(X)$. On remarque alors que la partie sans carré de $W(X)$ est exactement $\prod_{i \in I} P_i(X)$, et que, comparé à $V(X)$, les multiplicités des polynômes P_i dans W ont été divisées par p. On peut donc réitérer le procédé (calcul de U, calcul de V, etc.) sur le polynôme W jusqu'à ce que $V = 1$. L'algorithme 14 résume formellement la méthode.

Algorithme 14 : Extraction de la partie sans-carré

Entrée : Un polynôme $P(X) \in \mathbb{F}_q[X]$ unitaire
Sortie : Une liste de polynômes $Q_1(X), \ldots, Q_r(X) \in FF_q[X]$ tels que le produit $Q_1(X) \ldots Q_r(X)$ est la partie sans-carré de $P(X)$

1 Calculer le polynôme dérivé $P'(X)$.
2 Calculer $D(X) = \text{pgcd}(P(X), P'(X))$.
3 Calculer $U(X) = P(X)/D(X)$.
4 Calculer $V(X) = P(X)/\text{pgcd}(U(X)^{\deg P}, P(X))$.
5 **Si** $V(X) = 1$
6 **Retourner** la liste $[U(X)]$.
7 **Sinon**
8 Calculer une racine p-ème $W(X)$ de $V(X)$.
9 Effectuer un appel récursif avec $W(X)$, pour obtenir une liste L.
10 **Retourner** $L \cup [U(X)]$.

Pour terminer, il nous reste à expliquer comment on peut calculer efficacement $W(X)$ à partir de $V(X) = W(X)^p$, sur un corps fini de caractéristique p. Écrivons $W(X) = \sum_{i=0}^{d} w_i X^i$. On a alors :

$$V(X) = W(X)^p = \left(\sum_{i=0}^{d} w_i X^i\right)^p = \sum_{i=0}^{d} w_i^p X^{pi}\,.$$

Si $V(X) = \sum_{j=0} v_j X^j$, on note alors que seuls les v_j avec $p \mid j$ sont non-nuls, et que

$$v_{pi} = w_i^p, \quad \forall i \in \{0, \ldots, \deg(V)/p\}\,.$$

Pour obtenir $W(X)$, il suffit donc d'extraire des racines p-èmes dans $\mathbb{F}_q$ de caractéristique p. De manière assez similaire au calcul de racines carrées dans $\mathbb{F}_{2^k}$ (voir Proposition 3.9), cela s'effectue élégamment en calculant

$$v_{ip}^{q/p} = (w_i^p)^{q/p} = w_i^q = w_i\,.$$

Proposition 4.8

Il existe un algorithme déterministe permettant d'extraire la partie sans carré d'un polynôme unitaire $P(X) \in \mathbb{F}_q[X]$ de degré n en $O(n\mathsf{M}(n)\log n)$ opérations dans $\mathbb{F}_q$.

Démonstration : La méthode est décrite dans l'Algorithme 14, auquel il faut ajouter l'extraction de racine p-ème présentée ci-dessus. Concernant la complexité de l'algorithme, on a au plus n appels récursifs, chacun avec une complexité $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{F}_q$. ■

Donnons un exemple d'exécution de l'Algorithme 14.

Exemple 4.9

Dans $\mathbb{F}_3[X]$, on cherche à obtenir la partie sans-carré du polynôme

$$P(X) = X^{12} + 2\cdot X^{11} + X^{10} + 2\cdot X^8 + X^7 + 2\cdot X^5 + X^4 + 2\cdot X^2 + X + 2\,.$$

Notons que la factorisation de $P(X)$ est $(X+1)^3(X+2)(X^2+1)^4$, mais qu'elle est inconnue en entrée de l'algorithme. Déroulons maintenant l'Algorithme 14.

- *Premier appel :*

$$\begin{aligned}
P(X) &= X^{12} + 2\cdot X^{11} + X^{10} + 2\cdot X^8 + X^7 + 2\cdot X^5 + X^4 + 2\cdot X^2 + X + 2\\
P'(X) &= X^{10} + X^9 + X^7 + X^6 + X^4 + X^3 + X + 1\\
\operatorname{pgcd}(P(X), P'(X)) &= X^9 + X^6 + X^3 + 1\\
U &= \frac{P(X)}{\operatorname{pgcd}(P(X),P'(X))} = X^3 + 2\cdot X^2 + X + 2\\
V &= \frac{P(X)}{\operatorname{pgcd}(P(X), U(X)^{\deg P})} = X^3 + 1
\end{aligned}$$

  On a donc obtenu une partie de la décomposition en facteurs sans carré :

$$U(X) = X^3 + 2\cdot X^2 + X + 2\,.$$

  Notons que, bien qu'on n'en ait pas connaissance dans le déroulé de l'algorithme, $U(X)$ est le produit $(X+2)(X^2+1)$.
  Comme $V(X) \neq 1$, l'algorithme ne se termine pas. Il faut calculer une racine cubique de $V(X)$. Après un calcul rapide, on obtient $V(X) = (X+1)^3$.

- *On relance l'algorithme avec le nouveau polynôme $P(X) = X + 1$, ce qui donne :*

$$\begin{aligned}
P(X) &= X + 1\\
P'(X) &= 1\\
\operatorname{pgcd}(P(X), P'(X)) &= 1\\
U(X) &= X + 1\\
V(X) &= 1
\end{aligned}$$

Comme $V(X) = 1$, l'algorithme s'arrête. Notons que $X + 1$ est irréductible et on obtient le dernier facteur $X + 1$.

La liste renvoyée est $[X^3 + 2 \cdot X^2 + X + 2, X + 1]$, et le produit des éléments de cette liste forme bien la partie sans-carrée de $X^{12} + 2 \cdot X^{11} + X^{10} + 2 \cdot X^8 + X^7 + 2 \cdot X^5 + X^4 + 2 \cdot X^2 + X + 2$.

4.1.2 Factorisation de la partie sans carré : algorithme de Berlekamp

Après avoir obtenu le coefficient dominant et la partie sans carré du polynôme $P(X) \in \mathbb{F}_q[X]$ à factoriser, intéressons-nous à l'extraction de facteurs propres.

Problème 5

Étant donné un polynôme $P(X) = \prod_{i=1}^{k} P_i(X) \in \mathbb{F}_q[X]$ où les $P_i(X)$ sont unitaires, irréductibles dans $\mathbb{F}_q[X]$ et distincts, trouver les polynômes $P_i(X)$.

On suppose donc maintenant que $P = \prod_{i=1}^{k} P_i$ où les P_i sont unitaires, irréductibles dans $\mathbb{F}_q[X]$ et distincts. On note également $n = \deg P$.

Pour obtenir un facteur propre de P, une idée est de reformuler le problème de la façon suivante : dans l'anneau-quotient $\mathbb{F}_q[X]/(P)$, un facteur propre de P correspond à un élément Q tel qu'il existe R vérifiant $QR = 0 \mod P$. Autrement dit, on cherche des diviseurs de 0 dans $\mathbb{F}_q[X]/(P)$.

Cela nous amène donc à étudier la structure de l'anneau-quotient $\mathbb{F}_q[X]/(P)$.

Proposition 4.10

L'anneau quotient $\mathbb{F}_q[X]/(P)$:

1. *est aussi un espace vectoriel (donc une $\mathbb{F}_q$-algèbre) de dimension n sur $\mathbb{F}_q$;*
2. *se décompose comme*

$$\mathbb{F}_q[X]/(P) \simeq \left(\mathbb{F}_q[X]/(P_1)\right) \times \cdots \times \left(\mathbb{F}_q[X]/(P_k)\right)$$

où les $\mathbb{F}_q[X]/(P_i)$ sont des extensions du corps $\mathbb{F}_q$, de degré $\deg P_i$.

Démonstration : Laissée en exercice. ■

Pour factoriser P, E. Berlekamp [Ber67, Ber70] propose d'étudier l'endomorphisme :

$$\begin{array}{rccc} \phi : & \mathbb{F}_q[X]/(P) & \to & \mathbb{F}_q[X]/(P) \\ & A & \mapsto & A^q - A \end{array}$$

L'isomorphisme $\mathbb{F}_q[X]/(P) \simeq \mathbb{F}_q[X]/(P_1) \times \cdots \times \mathbb{F}_q[X]/(P_k)$ induit un endomorphisme $\widetilde{\phi}$ de $\mathbb{F}_q[X]/(P_1) \times \cdots \times \mathbb{F}_q[X]/(P_k)$. Étudions le noyau de $\widetilde{\phi}$.

Lemme 4.11

On a

$$\ker \widetilde{\phi} = \{(\lambda_1, \ldots, \lambda_k) \mid \lambda_i \in \mathbb{F}_q\}.$$

Par conséquent, le noyau de ϕ a dimension k.

Démonstration : Si $A \in \mathbb{F}_q[X]/(P)$, alors on a

$$\phi(A) = 0 \iff \forall i,\ A^q \equiv A \mod P_i$$

Or, dans le corps $\mathbb{F}_q[X]/(P_i)$ de cardinal $q^{\deg P_i}$, les solutions de $x^q = x$ sont exactement les éléments du sous-corps $\mathbb{F}_q$. Donc, $\widetilde{\phi}$ s'annule exactement sur les éléments de la forme $(\lambda_1, \ldots, \lambda_k)$ où $\lambda_j \in \mathbb{F}_q$. ■

Remarque 4.12

Notons que $\ker \phi$ *est une sous-algèbre de* $\mathbb{F}_q[X]/(P)$. *On l'appelle* algèbre de Berlekamp *associée à* P.

Fixons maintenant une base monomiale de $\mathbb{F}_q[X]/(P)$: si α est la classe de X modulo P, une telle base s'écrit $\{1, \alpha, \ldots, \alpha^{n-1}\}$. Tout élement $B(\alpha) \in \ker \phi$ peut donc être identifié à un polynôme $B \in \mathbb{F}_q[X]$ de degré $< n$.

Theorème 4.13 (*Berlekamp*)

Si $P(X) = \prod_{i=1}^{k} P_i(X) \in \mathbb{F}_q[X]$ *est un produit de polynômes irréductibles distincts, et* $B(X) \in \mathbb{F}_q[X]$ *est un polynôme de degré* $< n$ *tel que* $B(X)^q \equiv B(X) \mod P(X)$, *alors :*

$$P(X) = \prod_{\lambda \in \mathbb{F}_q} \operatorname{pgcd}(P(X), B(X) - \lambda)\,.$$

Démonstration : Soit $\phi : A \mapsto A^q - A \mod P$. D'après le Lemme 4.11, l'élément $B(\alpha) \in \ker \phi$ s'écrit $(\beta_1, \ldots, \beta_k)$ dans $\mathbb{F}_q[X]/(P_1) \times \cdots \times \mathbb{F}_q[X]/(P_k)$ avec $\beta_i \in \mathbb{F}_q$ pour tout $i \in \{1, \ldots, k\}$.
Fixons $i \in \{1, \ldots, k\}$. D'après ce qui précède, le polynôme $P_i(X)$ divise $B(X) - \beta_i$ dans $\mathbb{F}_q[X]$. Fixons maintenant $\lambda \in \mathbb{F}_q$ et notons $D_\lambda(X) \coloneqq \operatorname{pgcd}(P(X), B(X) - \lambda)$. On remarque alors que

$$P_i \mid D_\lambda \iff \beta_i = \lambda\,.$$

En effet :

- Si $\lambda = \beta_i$, alors $P_i(X)$ divise $B(X) - \beta_i = B(X) - \lambda$, donc $P_i(X)$ divise aussi $D_\lambda(X)$.
- Si $P_i(X)$ divise $D_\lambda(X)$, alors $P_i(X)$ divise $B(X) - \lambda$, ce qui signifie que β_i, défini comme $B(X) \mod P_i(X)$, vaut λ.

On en déduit :

$$D_\lambda(X) = \prod_{i \mid \beta_i = \lambda} P_i(X) \qquad \text{puis} \qquad P(X) = \prod_{\lambda \in \mathbb{F}_q} D_\lambda(X)\,.$$

■

Analysons les conséquences pratiques du Théorème de Berlekamp.

D'abord, notons que l'on peut efficacement calculer un polynôme $B(X) \in \mathbb{F}_q[X]$ de degré $1 \leq \deg B < n$ tel que $B(\alpha) \in \ker \phi$. Pour cela, il suffit de calculer un élément du noyau de la matrice de ϕ dans la base $\{1, \alpha, \ldots, \alpha^{n-1}\}$.

Puis, comme $\deg B < n$, deux au moins des $D_\lambda = \operatorname{pgcd}(P, B - \lambda)$ sont des facteurs propres de P. Lorsque le cardinal de $\mathbb{F}_q$ est « petit », une idée est donc d'énumérer $\mathbb{F}_q$ (ou d'y tirer aléatoirement des éléments) pour obtenir des diviseurs propres de P. On obtient alors l'Algorithme 15.

Lemme 4.14

L'Algorithme 15 produit un facteur propre du polynôme sans carré P *en* $O(n^3 + q\mathsf{M}(n)\log n)$ *opérations sur* $\mathbb{F}_q$ *en pire cas.*

Démonstration : Une grande partie de la preuve de validité a été faite avant la présentation de l'algorithme. Justifions simplement le choix de la forme spécifique pour le vecteur $\boldsymbol{b}$. L'idée est qu'il faut prendre $B(X)$ de degré ≥ 1, car sinon les $D_\lambda = \operatorname{pgcd}(P, B - \lambda)$ seront constants, sauf pour $B = \lambda$ auquel cas $D_\lambda = P$.

Algorithme 15 : Algorithme de Berlekamp en petite cardinalité

Entrée : un polynôme $P = \prod_{i=1}^{k} P_i$ sans facteur carré
Sortie : au moins un diviseur propre Q de P

Calculer la matrice $\boldsymbol{M}$ de ϕ dans la base polynomiale $(1, \alpha, \ldots, \alpha^{n-1})$.
Calculer un élément non-nul du noyau de $\boldsymbol{M}$ de la forme $(b_0 = 0, \ldots, b_{n-1})$.
Construire $B(X) = \sum_{j=0}^{n-1} b_j X^j$.
Faire
  Choisir un nouveau $\lambda \in \mathbb{F}_q$ — de façon déterministe (énumération) ou probabiliste.
  Calculer $Q(X) = \mathrm{pgcd}(P(X), B(X) - \lambda)$.
tant que $Q(X) = 1$;
Retourner Q.

Pour la complexité, le calcul de la matrice $\boldsymbol{M}$ et d'un élément de son noyau requiert $O(n^3)$ opérations sur $\mathbb{F}_q$. Une fois le vecteur $\boldsymbol{b}$ obtenu, le calcul d'un pgcd nécessite $O(\mathsf{M}(n) \log n)$ opérations sur $\mathbb{F}_q$, et doit être effectué au plus $q - 1$ fois. ■

Exemple 4.15

On reprend l'Exemple 4.9, c'est-à-dire $q = 3$, et $P = X^{12} + 2 \cdot X^{11} + X^{10} + 2 \cdot X^8 + X^7 + 2 \cdot X^5 + X^4 + 2 \cdot X^2 + X + 2$, où l'on avait trouvé la partie sans carré $Q = X^4 + 2$.
Comme prescit par l'Algorithme 15, on calcule d'abord la matrice des $X^{3i} - X^i \bmod Q$ dans la base $(1, X, X^2, X^3)$:

$$\boldsymbol{M} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \end{pmatrix}$$

On calcule ensuite un $B \in \ker \boldsymbol{M}$ non-constant aléatoire, disons $B(X) = X^2 + 2$. On obtient alors :

λ	$D_\lambda = \mathrm{pgcd}(Q, B - \lambda)$
0	$X^2 + 2$
1	$X^2 + 1$
2	1

On a donc trouvé deux diviseurs propres de Q : $X^2 + 2$ et $X^2 + 1$. On pourrait ensuite appeler récursivement l'algorithme sur ces deux polynômes pour obtenir la factorisation définitive...

On remarque que la complexité de l'Algorithme 15 reste exponentielle en $\log q$. Il doit donc être utilisé seulement lorsque q est petit.

Lorsque q est trop grand pour être énuméré, l'idée est de regrouper certains diviseurs de la forme $\mathrm{pgcd}(P, B - \lambda)$, en un seul calcul de pgcd. Pour cela, remarquons que si q est impair [1], alors pour tout $\beta \in \mathbb{F}_q$:

1. ou bien $\beta = 0$,
2. ou bien β est un carré non-nul dans $\mathbb{F}_q$, auquel cas $\beta^{(q-1)/2} - 1 = 0$,
3. ou bien β est un non-carré dans $\mathbb{F}_q$, auquel cas $\beta^{(q-1)/2} + 1 = 0$.

En considérant $\beta = B \mod P_i$, on peut donc considérer trois sous-ensembles d'indices :

1. $I_1 := \{i \in \{1, \ldots, k\} \mid B \bmod P_i = 0\}$,
2. $I_2 := \{i \in \{1, \ldots, k\} \mid B^{(q-1)/2} - 1 \bmod P_i = 0\}$,

1. si q est pair, il faut utiliser une méthode similaire, voir en TD

3. $I_3 := \{i \in \{1,\dots,k\} \mid B^{(q-1)/2} + 1 \bmod P_i = 0\}$.

Ensuite, on peut démontrer qu'avec une probabilité $\geq 1/2$ sur le choix de $B(X)$, deux des ensembles I_1, I_2 et I_3 sont non-vides. Dans ce cas (favorable), on obtient donc une factorisation non-triviale

$$P = \mathrm{pgcd}(P, B) \times \mathrm{pgcd}(P, B^{(q-1)/2} - 1) \times \mathrm{pgcd}(P, B^{(q-1)/2} + 1)\,.$$

car 2 de ces facteurs sont des facteurs propres de P.

Algorithme 16 : Algorithme de Berlekamp en grande cardinalité, avec $\mathrm{car}(\mathbb{F}_q) \neq 2$.

Entrée : un polynôme $P = \prod_{i=1}^{k} P_i$ sans facteur carré
Sortie : au moins deux diviseurs propres de P

Calculer la matrice $\boldsymbol{M}$ de ϕ dans la base polynomiale $(1, \alpha, \dots, \alpha^{n-1})$.
Faire
  Calculer un aléatoirement un élément non-nul $\boldsymbol{b}$ du noyau de $\boldsymbol{M}$ de la forme $\boldsymbol{b} = (b_0 = 0, b_1, \dots, b_{n-1})$.
  Construire $B(X) = \sum_{j=0}^{n-1} b_j X^j$.
  Calculer $B^{(q-1)/2} - 1 \bmod P$ et $B^{(q-1)/2} + 1 \bmod P$.
  Calculer $A_0 = \mathrm{pgcd}(P, B)$, $A_1 = \mathrm{pgcd}(P, B^{(q-1)/2} - 1)$ et $A_2 = \mathrm{pgcd}(P, B^{(q-1)/2} + 1)$.
tant que *l'un des A_i vaut P*;
Retourner les A_i différents de 1.

Lemme 4.16

L'Algorithme 16 produit, de manière probabiliste, un facteur propre du polynôme sans carré P en $O(n^3 + \mathsf{M}(n) \log n \log q)$ opérations sur $\mathbb{F}_q$ en moyenne.

4.2 Factorisation de polynômes dans les rationnels et les entiers

Dans cette partie, on s'intéresse aux problèmes de la factorisation de polynômes sur les entiers et les rationnels. Nous allons voir que ces deux problèmes sont liés (mais non-équivalents), et que l'on peut s'aider de la factorisation de polynômes sur les corps finis pour factoriser les polynômes sur les rationnels.

4.2.1 Deux problèmes équivalents ?

Commençons par décrire formellement les problèmes.

Problème 6 (*Factorisation dans $\mathbb{Z}[X]$ ou $\mathbb{Q}[X]$*)

Soit $A = \mathbb{Z}$ ou $\mathbb{Q}$. Étant donné $F(X) \in A[X]$, décomposer (à une unité de A près) le polynôme $F(X)$ en facteurs irréductibles de $A[X]$, c'est-à-dire

$$F(X) = \prod_{i=1}^{k} F_i(X)^{m_i}, \quad m_i \geq 1\,.$$

Il est naturel de se questionner sur ce qui distingue les deux problèmes : si $F(X) \in \mathbb{Z}[X]$, est-ce équivalent de factoriser $F(X)$ dans $\mathbb{Z}[X]$ et dans $\mathbb{Q}[X]$?

On peut d'abord répondre à cette question d'un point de vue théorique. On observe facilement que les unités et les irréductibles de $\mathbb{Z}[X]$ et $\mathbb{Q}[X]$ ne sont pas les mêmes. En effet, les unités de $\mathbb{Z}[X]$ sont 1 et -1, tandis que celles de $\mathbb{Q}[X]$ sont les éléments non-nuls de $\mathbb{Q}$. Ainsi, si par exemple $10(2X+1)(3X-1)$ est une factorisation de $60X^2+10X-10$ dans $\mathbb{Q}[X]$, elle n'en est pas une dans $\mathbb{Z}[X]$ puisque 10 n'y est pas irréductible : il se factorise comme 2×5. Par conséquent, généralement, une factorisation de $F(X) \in \mathbb{Z}[X]$ dans $\mathbb{Q}[X]$ en polynômes à coefficients entiers **n'est pas nécessairement** une factorisation de $F(X)$ dans $\mathbb{Z}[X]$.

En revanche, si l'on connaît une factorisation de $F(X) \in \mathbb{Z}[X]$ dans $\mathbb{Z}[X]$, alors en multipliant entre eux les termes constants, on obtient une factorisation de $F(X)$ dans $\mathbb{Q}[X]$. Ce résultat n'est pas trivial, car un irréductible de $\mathbb{Z}[X]$ pourrait se factoriser dans $\mathbb{Q}[X]$. Le lemme de Gauss, que nous donnerons plus tard, nous assure que c'est impossible.

Avant cela, précisons un peu de terminologie.

Définition 4.17

Soit $F(X) = \sum_i f_i X^i \in \mathbb{Z}[X]$.

- *Le* contenu *de F, noté* $\mathrm{cont}(F) \in \mathbb{Z}$, *est le* pgcd *des coefficients de F.*
- *Le polynôme F est* primitif *si* $\mathrm{cont}(F) = 1$. *La* partie primitive *de F est $F/\mathrm{cont}(F)$.*
- *Le* coefficient de tête *est noté* $\mathrm{lc}(F)$.
- *On note $\|F\|_\infty = \max\{|f_i|\}$ la plus grande valeur absolue d'un coefficient de $F(X)$. On peut vérifier que $\|\cdot\|_\infty$ est bien une norme sur $\mathbb{Z}[X]$.*

Exemple 4.18

Soit $F(X) = 6X^2 - 15$. Le contenu de F est $\mathrm{cont}(F) = 3$, *sa partie primitive est $F^*(X) = 2X^2 - 5$, son coefficient de tête est 6 et on a $\|F\|_\infty = 15$.*

Lemme 4.19 (*Gauss*)

Soit $F \in \mathbb{Z}[X]$. Alors, F est irréductible dans $\mathbb{Z}[X]$ si et seulement si F est primitif et irréductible dans $\mathbb{Q}[X]$.

Remarque 4.20

Le lemme de Gauss se démontre dans le cadre plus général où $F \in A[X]$ avec A un anneau factoriel, et $K = \mathrm{Frac}(A)$ le corps des fractions de A.

Par conséquent, on peut décrire les irréductibles de $\mathbb{Z}[X]$ comme l'ensemble des

1. des irréductibles de $\mathbb{Z}$ (les nombres premiers),
2. et des éléments primitifs de $\mathbb{Z}[X]$, irréductibles dans $\mathbb{Q}[X]$.

Cela implique le résultat suivant concernant notre problème algorithmique de factorisation.

Proposition 4.21

On a l'équivalence suivante entre les problèmes de factorisation :

$$\text{factoriser dans } \mathbb{Z}[X] \iff \begin{cases} \text{factoriser dans } \mathbb{Z} \\ \text{et} \\ \text{factoriser dans } \mathbb{Q}[X]\,. \end{cases}$$

$$\iff \begin{cases} \text{factoriser dans } \mathbb{Z} \\ \text{et} \\ \text{factoriser les polynômes primitfs de } \mathbb{Z}[X]\,. \end{cases}$$

Dans un chapitre ultérieur, nous traiterons la factorisation des entiers. Pour factoriser des polynômes à coefficients entiers, on peut maintenant se restreindre au cas où le polynôme est primitif.

4.2.2 Présentation de la stratégie de factorisation

Notre nouvel objectif est le suivant : étant donné un polynôme primitif $F(X) \in \mathbb{Z}[X]$, factoriser $F(X)$. Pour résoudre ce problème, la stratégie est la suivante :

> STRATÉGIE DE FACTORISATION DANS $\mathbb{Z}[X]$
>
> Soit $F(X) = \prod_i F_i(X)^{m_i} \in \mathbb{Z}[X]$ le polynôme à factoriser.
>
> 1. Éliminer les facteurs carrés dans F, notamment en calculant $\text{pgcd}(F, F')$. Soit $F^* = \prod_i F_i(X)$ le polynôme obtenu.
> 2. Choisir un « grand entier » N, respectant certaines conditions de divisibilité, et obtenir une factorisation de $F^* \mod N$.
> 3. Regrouper certains facteurs irréductibles modulo N, afin d'obtenir la factorisation de F^* dans $\mathbb{Z}[X]$.
> 4. Par tests de divisibilité successifs, retrouver la multiplicité m_i des facteurs $F_i(X)$ dans $F(X)$.

Pour résumer, l'idée est de tirer parti des algorithmes de factorisation efficaces décrits dans la Section 4.1. Intuitivement, on choisit un nombre premier p de taille importante (par rapport aux entiers impliqués dans la factorisation) et on espère que la factorisation de $F(X)$ dans $\mathbb{F}_p[X]$ « donne aisément » une factorisation $F(X)$ dans $\mathbb{Z}[X]$. Nous allons voir par la suite que, si l'idée est prometteuse, certains obstacles rendent la tâche non-triviale. Nous donnerons dans les Sections 4.2.3 et 4.2.4 les algorithmes qui permettent d'éliminer ces obstacles.

Élimination des facteurs carrés. Traitons dès à présent la première étape, car elle est plus simple que dans le cas de $\mathbb{F}_p[X]$. Si $F(X) = \prod_{i=1}^{k} F_i(X)^{m_i} \in \mathbb{Z}[X]$ est primitif, alors

$$F'(X) = \sum_{i=1}^{k} m_i F_i'(X) \frac{F(X)}{F_i(X)}$$

puis

$$\text{pgcd}(F(X), F'(X)) = \prod_{i=1}^{k} F_i(X)^{m_i - 1}$$

On obtient donc la partie sans carrée de $F(X)$ en calculant

$$F^*(X) = \frac{F(X)}{\text{pgcd}(F(X), F'(X))}\,.$$

Remarquons que l'itération de cette méthode permet d'obtenir la *factorisation en parties sans carré* de $F(X)$, c'est-à-dire d'obtenir la liste de polynômes $A_1(X), A_2(X), \ldots, A_d(X) \in \mathbb{Z}[X]$ telle que

$$F(X) = A_1(X)A_2(X)^2 \cdots A_d(X)^d\,,$$

et chacun des $A_i(X)$ est sans carré. L'Algorithme 17 qu'on en déduit est connu sous le nom d'algorithme de Yun.

Algorithme 17 : Algorithme de Yun

Entrée : Un polynôme primitif $F(X) = A_1(X)A_2(X)^2 \cdots A_d(X)^d \in \mathbb{Z}[X]$, où les $A_i(X) \in \mathbb{Z}[X]$ sont sans carré.
Sortie : La factorisation sans carré $(A_1(X), \ldots, A_d(X))$ de $F(X)$

```
A ← []
P ← F
Tant que deg(P) ≠ 0 faire
    D ← pgcd(P, P')
    S ← P/D
    A ← S/pgcd(S,D)
    P ← D
Retourner A.
```

Exemple 4.22

Donnons ici un exemple d'exécution de l'algorithme de Yun sur le polynôme

$$P(X) = (2X+1)^4X^3(X^2+1)(X+8)$$

dont la forme développée, en entrée de l'algorithme, est :

$$P(X) = 16X^{10} + 160X^9 + 296X^8 + 360X^7 + 345X^6 + 208X^5 + 65X^4 + 8X^3\,.$$

On obtient successivement les valeurs suivantes :

P	D	S	A
$8X^5 + 12X^4$ $+6X^3 + X^2$	$8X^5 + 12X^4$ $+6X^3 + X^2$	$2X^5 + 17X^4 + 10X^3$ $+17X^2 + 8X$	$\left[X^3 + 8X^2 + X + 8\right]$
$4X^3 + 4X^2 + X$	$4X^3 + 4X^2 + X$	$2X^2 + X$	$\left[X^3 + 8X^2 + X + 8,\ 1\right]$
$2X + 1$	$2X + 1$	$2X^2 + X$	$\left[X^3 + 8X^2 + X + 8,\ 1,\ X\right]$
1	1	$2X + 1$	$\left[X^3 + 8X^2 + X + 8,\ 1,\ X,\ 2X + 1\right]$

4.2.3 Bonne réduction pour la factorisation

Dans cette section, nous supposons que le polynôme $F(X)$ à factoriser est sans carré et primitif, et que ses facteurs irréductibles sont $F_1(X), \ldots, F_d(X)$. On note $n = \deg(F)$. Rappelons

que notre stratégie consiste d'abord à factoriser le polynôme $F(X)$ modulo N (c'est-à-dire dans $(\mathbb{Z}/N\mathbb{Z})[X]$), puis d'espérer en déduire une factorisation dans $\mathbb{Z}[X]$. Pour le moment, choisissons $N = p$ un nombre premier pour simplifier le propos.

Néanmoins, le choix de la valeur de p ne peut pas être quelconque. Nous allons voir dans les points suivants une succession d'exemples qui illustrent les contraintes sur ce choix de p. Les résultats théoriques seront donnés plus tard.

• Commençons par donner un exemple où la stratégie mentionnée dans la partie précédente se déroule idéalement.

Exemple 4.23

Soit $F(X) = X^3 + 3X^2 + 2X \in \mathbb{Z}[X]$. Choisissons comme nombre premier $p = 11$. Alors l'algorithme de factorisation des polynômes dans $\mathbb{F}_p[X]$ nous renvoie la factorisation :

$$\overline{F}(X) = X(X + \overline{1})(X + \overline{2})$$

où $\bar{\cdot}$ souligne le fait que ces élements sont pris modulo p. Si l'on associe à ces trois polynômes de $\mathbb{F}_p[X]$ les polynômes X, $X + 1$ et $X + 2$ de $\mathbb{Z}[X]$, alors le produit $X(X + 1)(X + 2)$ vaut bien $F(X)$ dans $\mathbb{Z}[X]$. Comme ces trois polynômes sont de degré 1, ils sont irréductibles, et on a donc correctement factorisé $F(X)$ dans $\mathbb{Z}[X]$.

• Malheureusement, ce cas est loin d'être général. Un premier problème provient du fait qu'en associant à $\overline{a} \in \mathbb{F}_p$ l'entier $a \in \{0, \dots, p-1\}$ correspondant, on n'arrive pas à gérer les polynômes à coefficients négatifs. Pour résoudre ce problème, pour tout $a \in \mathbb{Z}/p\mathbb{Z}$, on choisit comme image de a dans $\mathbb{Z}$ l'unique représentant de a appartenant à l'intervalle

$$I_p := \left\{ -\left\lfloor \frac{p-1}{2} \right\rfloor, \dots, \left\lfloor \frac{p}{2} \right\rfloor \right\}.$$

Exemple 4.24

Soit $F(X) = X^3 + X^2 - 2X \in \mathbb{Z}[X]$. Avec le même choix de $p = 11$ que dans l'exemple précédent, l'algorithme de factorisation des polynômes dans $\mathbb{F}_p[X]$ nous renvoie la factorisation :

$$\overline{F}(X) = X(X + \overline{10})(X + \overline{2})$$

où $\bar{\cdot}$ souligne encore une fois le fait que ces élements sont pris modulo p. Si l'on associe à ces trois polynômes de $\mathbb{F}_p[X]$ les polynômes X, $X + 10$ et $X + 2$ de $\mathbb{Z}[X]$, alors le produit $X(X + 10)(X + 2) = X^3 + 12X^2 + 20X$ est différent de $F(X)$ dans $\mathbb{Z}[X]$.
Pour obtenir la bonne factorisation, il faut associer à $X + \overline{10}$ le polynôme $X - 1$, c'est-à-dire que l'entier $10 \in \mathbb{F}_p$ doit être représenté par $-1 \in \{-\lfloor \frac{p-1}{2} \rfloor, \dots, \lfloor \frac{p}{2} \rfloor\} = \{-5, \dots, 5\}$.

• Un autre problème majeur concerne la taille de l'entier p à choisir.

Exemple 4.25

Soit $F(X) = X^3 + 12X^2 + 20X \in \mathbb{Z}[X]$. Avec le même choix de $p = 11$ que dans l'exemple précédent, l'algorithme de factorisation des polynômes dans $\mathbb{F}_p[X]$ nous renvoie la factorisation :

$$\overline{F}(X) = X(X + \overline{10})(X + \overline{2}).$$

Comme on associe maintenant à ces trois polynômes de $\mathbb{F}_p[X]$ les polynômes X, $X - 1$ et $X + 2$ de $\mathbb{Z}[X]$, le produit $X(X - 1)(X + 2) = X^3 + X^2 - 2X$ est une nouvelle fois différent de $F(X)$ dans $\mathbb{Z}[X]$.

Pour obtenir la bonne factorisation, il faudrait prendre une valeur de p bien plus grande. Par exemple, pour $p = 41$, l'élément $\overline{10}$ de $\mathbb{F}_p$ est associé à $10 \in \mathbb{Z}$, et on obtient alors la bonne factorisation $X(X+10)(X+2) = F(X)$.

Observons que l'on a réglé le problème de l'exemple précédent en choisissant p de sorte que tous les coefficients de $F(X)$ soient « bien réduits » modulo p, c'est-à-dire de sorte que $a \in I_p$ pour tout coefficient a de $F(X)$. Autrement dit, on doit choisir

$$p > 2\,\|F\|_\infty\,.$$

• Hélas, une nouvelle fois, ce choix de p n'est pas suffisant. Il ne faut pas seulement que les coefficients de $F(X)$ soient bien réduits, mais également que les coefficients de **tous ses facteurs** le soient. L'exemple suivant en atteste.

Exemple 4.26

Soit $F(X) = X^8 + X^6 + X^5 - X^3 + X^2 - X - 1 \in \mathbb{Z}[X]$. Notons dès maintenant que $F(X)$ se factorise comme

$$F(X) = (X^2 - X + 1)(X^6 + X^5 + X^4 + X^3 - 2X - 1) \quad \textit{dans } \mathbb{Z}[X]\,.$$

La borne précédente nous permettrait de prendre $p = 3$. Malheureusement, pour une telle de valeur de p, l'algorithme de factorisation dans $\mathbb{F}_p[X]$ retourne comme second facteur $\overline{F_2}(X) = X^6 + X^5 + X^4 + X^3 + X + \overline{2}$, qui se transcrit comme $X^6 + X^5 + X^4 + X^3 + X - 1$ dans $\mathbb{Z}[X]$.

On doit donc choisir

$$p > 2\max\{\|F_i\|_\infty, 1 \le i \le d\}\,.$$

Remarque 4.27

On pourrait penser que, si $G(X)$ divise $F(X)$ dans $\mathbb{Z}[X]$, alors $\|G(X)\|_\infty \le \|F(X)\|_\infty$. Ce n'est malheureusment pas le cas, et cela peut en être assez loin. Prenons par exemple

$$F(X) := 5X^4 + 7X^3 + X^2 + 4X + 8 = (X^2 - X + 1) \times (5X^2 + 12X + 8) \in \mathbb{Z}[X]$$

Le second facteur de $F(X)$ a un coefficient égal à 12, qui est de valeur absolue sensiblement plus grande que $\|F\|_\infty = 8$.

Néanmoins, il existe des bornes reliant la taille des coefficients de F et de ses diviseurs. La borne de Landau–Mignotte en est un exemple célèbre.

Theorème 4.28 (*Borne de Landau–Mignotte*)

Soit $G \in \mathbb{Z}[X]$ un polynôme qui divise $F(X)$. Si $m := \deg(G)$, alors

$$\|G\|_\infty \le \binom{m}{\lceil m/2 \rceil} \|F\|_2$$

où

$$\|F\|_2 := \sqrt{\sum_{i=0}^{n} |f_i|^2}\,.$$

Démonstration : Todo: à venir ■

Remarque 4.29

Pour $m = 1$, la borne de Landau–Mignotte produit une majoration de la valeur absolue des racines entières d'un polynôme. En effet, $a \in \mathbb{Z}$ est racine de $F \in \mathbb{Z}[X]$ si et seulement si $X - a$ divise $F(X)$. La borne de Landau–Mignotte nous assure alors que

$$|a| \leq \|F\|_2 .$$

Observons que les normes $\|\cdot\|_2$ et $\|\cdot\|_\infty$ sont équivalentes. En particulier, on a :

$$\|F\|_2 = \sqrt{\sum_{i=0}^{n} |f_i|^2} \leq \sqrt{\sum_{i=0}^{n} \|F\|_\infty^2} \leq \sqrt{n+1}\,\|F\|_\infty .$$

Par conséquent, on obtient le corollaire suivant.

Corollaire 4.30

Soient $G \in \mathbb{Z}[X]$ un polynôme qui divise $F(X)$. Si $m := \deg(G)$, alors

$$\|G\|_\infty \leq 2^m \sqrt{n+1}\,\|F\|_\infty .$$

Démonstration : Il nous reste simplement à démontrer que $\binom{m}{\lceil m/2 \rceil} \leq 2^m$. Si m est pair, alors on pose $m = 2k$ et on obtient

$$\binom{m}{\lceil m/2 \rceil} = \frac{(2k)!}{k!k!} = \frac{2k \times (2k-2) \times \ldots 2}{k \times (k-1) \times \cdots \times 1} \times \frac{(2k-1) \times (2k-3) \times \ldots 1}{k \times (k-1) \times \cdots \times 1} \leq 2^k \times 2^k = 2^m$$

en réordonnant les produits.
Si $m = 2k+1$ est impair, le calcul est similaire :

$$\binom{m}{\lceil m/2 \rceil} = \frac{(2k+1)!}{k!(k+1)!} = \frac{2k \times (2k-2) \times \ldots 2}{k \times (k-1) \times \cdots \times 1} \times \frac{(2k+1) \times (2k-3) \times \ldots 1}{(k+1) \times k \times \cdots \times 1} \leq 2^k \times 2^{k+1} = 2^m .$$

■

La borne de Landau–Mignotte donne une valeur minimale de p pour laquelle on est certain que les facteurs $F_i(X)$ seront bien réduits modulo p. Cependant, cette borne est exponentielle en le degré de F et mène à choisir des valeurs de p beaucoup plus grandes que nécessaires. Cela a un impact en terme de complexité, car, par exemple, le temps d'exécution des algorithmes de factorisation de polynômes sur les corps finis dépend fortement de la taille du corps considéré.

Pour accélérer les calculs, on aimerait donc avoir un algorithme itératif pour lequel la réduction des polynômes modulo N est réalisée avec des entiers N de taille croissante, mais non nécessairement premiers. L'idée est qu'en partant avec un premier module N_1 très petit, même si les premiers choix de modules $N_1 < \cdots < N_{k-1}$ ne donnent pas la factorisation escomptée, on espère toutefois l'obtenir avec un module N_k bien plus petit que la borne de Landau–Mignotte.

Pour rendre cette méthode efficace, il faut éviter d'avoir à recalculer entièrement les factorisations modulo N_i à chaque étape. Comme pour le calcul de racines carrées modulo N (voir Section 3.3.2), on va utiliser une méthode par *relèvement* de solutions. En définissant $N_i := p^i$ pour $i \geq 1$, on va pouvoir déduire la factorisation de F modulo N_i de celle modulo N_{i-1}.

Lemme 4.31 (*Lemme de Hensel, revisité pour la factorisation de polynômes*)

Soit $F \in \mathbb{Z}[X]$ primitif et $p \in \mathbb{Z}$ un nombre premier qui ne divise pas $\mathrm{lc}(F)$. *Supposons que $F \bmod p$ puisse s'écrire*

$$F \equiv \lambda_1 A_1 B_1 \mod p$$

où $A_1, B_1 \in \mathbb{F}_p[X]$ sont premiers entre eux.
Alors, pour tout $i \geq 1$, il existe un unique couple de polynômes $A_i, B_i \in (\mathbb{Z}/p^i\mathbb{Z})[X]$ unitaires et premiers entre eux, tels que

$$\begin{cases} F & \equiv A_i B_i & \mod p^i \\ A_i & \equiv A_1 & \mod p \\ B_i & \equiv B_1 & \mod p \end{cases}$$

Démonstration : La preuve se fait par récurrence et ressemble dans sa structure à celle de le remontée de Hensel pour le calcul de racines carrées.

Le cas $i = 1$ est clair. Soit $i \geq 1$ et supposons donc le résultat pour i. On a alors $p^i \mid F - \lambda_i A_i B_i$, par conséquent il existe $S(X) \in (\mathbb{Z}/p\mathbb{Z})[X]$ tel que

$$F - A_i B_i \equiv p^i S \mod p^{i+1}.$$

Cherchons des candidats pour A_{i+1} et B_{i+1}. S'ils existent, alors sont de la forme

$$A_{i+1} \equiv A_i + p^i Q \mod p^{i+1} \quad \text{et} \quad B_{i+1} \equiv B_i + p^i R \mod p^{i+1}$$

pour Q et R deux polynômes à coefficients dans $\mathbb{Z}/p\mathbb{Z}$. On aurait alors :

$$A_{i+1}B_{i+1} \equiv A_i B_i + p^i(QB_i + RA_i) \equiv A_i B_i + p^i(QB_1 + RA_1) \mod p^{i+1}$$

car par hypothèse de récurrence $A_i \equiv A_1 \mod p$ et $B_i \equiv B_1 \mod p$. Maintenant, si l'on suppose que $F \equiv A_{i+1}B_{i+1} \mod p^{i+1}$, alors on obtient :

$$A_i B_i + p^i S \equiv A_{i+1} B_{i+1} \mod p^{i+1}$$

ce qui induit $p^i(QB_1 + RA_1 - S) \equiv 0 \mod p^{i+1}$ puis :

$$S \equiv QB_1 + RA_1 \mod p.$$

Considérons maitenant U et V les polynômes de Bezout associés à A_1 et B_1 dnas $(\mathbb{Z}/p\mathbb{Z})[X]$; autrement dit $UA_1 + VB_1 \equiv 1 \mod p$. Alors on a :

$$\begin{cases} S \equiv QB_1 & \mod (p, A_1) \\ S \equiv RA_1 & \mod (p, B_1) \end{cases} \implies \begin{cases} Q \equiv SV & \mod (p, A_1) \\ R \equiv SU & \mod (p, B_1) \end{cases}$$

On a donc montré que, **si** A_{i+1} et B_{i+1} vérifient les conditions du résultat à démontrer, **alors** ils sont de la forme

$$\begin{cases} A_{i+1} = A_i + p^i(SV & \mod (p, A_1)) \\ B_{i+1} = B_i + p^i(SU & \mod (p, B_1)) \end{cases} \quad \text{où} \quad S = \frac{F - A_i B_i}{p^i} \mod p$$

On vérifie aisément que ces polynômes, définis de la sorte, conviennent bien. ∎

Le résultat précédent étant constructif, il mène à un algorithme itéartif pour « remonter » la factorisation de $(\mathbb{Z}/p\mathbb{Z})[X]$ à $(\mathbb{Z}/p^\ell\mathbb{Z})[X]$ pour ℓ suffisamment grand pour dépasser la borne de Landau–Mignotte. La méthode est présentée dans l'Algorithme 18.

Remarque 4.32

L'Algorithme 18 s'étend à un k-uplets de polynômes en entrée (au lieu de 2). Pour cela, il faut calculer les polynômes de Bezout généralisés.

Algorithme 18 : Remontée de Hensel pour la factorisation de polynômes.

Entrée : $F(X) \in \mathbb{Z}[X]$, p un nombre premier et $A_1(X), B_1(X) \in (\mathbb{Z}/p\mathbb{Z})[X]$ premiers entre eux, tels que $F(X) = A_1(X)B_1(X) \mod p$

Sortie : $A_\ell(X)$ et $B_\ell(X)$ premiers entre eux dans $(\mathbb{Z}/p^\ell\mathbb{Z})[X]$ tels que $F(X) = A_1(X)B_1(X) \mod p^\ell$

1 Calculer $U(X), V(X) \in (\mathbb{Z}/p\mathbb{Z})[X]$ les polynomes de Bezout associés à $A_1(X)$ et $B_1(X)$.
2 Calculer ℓ tel que p^ℓ dépasse la borne de Landau–Mignotte pour $F(X)$.
3 **Pour tout** $i = 1 \dots \ell - 1$ **faire**
4 Calculer $S = \frac{F - A_i B_i}{p^i} \mod p$
5 Calculer $A_{i+1} = A_i + p^i(SV \mod (p, A_1))$
6 Calculer $B_{i+1} = B_i + p^i(SU \mod (p, B_1))$
7 Retourner A_ℓ, B_ℓ.

Remarque 4.33

Pour p de taille constante, l'Algorithme 18 exécutera donc $\ell - 1 \in O(\log(\|F\|_\infty) + \deg(F))$ tours de boucle. Ajoutons qu'il existe une remontée de Hensel « quadratique », c'est-à-dire telle que l'on obtient successivement les facteurs de $F(X)$ modulo $p, p^2, p^4, p^8, \dots, p^{2^\ell}$, etc. Le nombre d'étapes devient alors $\ell \in O(\log(\log(\|F\|_\infty) + \deg(F)))$.

Rassemblons tout ce qui précède et essayons maintenant de factoriser un polynôme dans $\mathbb{Z}[X]$.

Exemple 4.34

Soit $F(X) = X^5 + 5X + 19X^3 + 86X^2 + 39X + 2 \in \mathbb{Z}[X]$.

1. *Élimination des facteurs carrés : $F'(X) = 5X^4 + 20X^3 + 57X^2 + 172X + 39$ mène à $\mathrm{pgcd}(F, F') = 1$. Il n'y a pas de facteur carré dans $F(X)$.*
2. *Choix de p : on prend $p = 5$ et on réduit $F(X)$ modulo p. On obtient*

$$\overline{F}(X) = X^5 + 4X^3 + X^2 + 4X + 2\,.$$

3. *On exécute l'algorithem de factorisationde Berlekamp sur $\overline{F}(X) \in \mathbb{F}_5[X]$. On obtient*

$$\overline{F} = \overline{A_1 B_1} \text{ avec } \overline{A_1}(X) = X^3 + 2X + 1 \text{ et } \overline{B_1}(X) = X^2 + 2X + 2\,.$$

4. *On calcule la borne de Landau–Mignotte. Prenons la borne précise :*

$$\ell = \left\lceil \log_5 \binom{5}{2} \|F\|_2 \right\rceil = 6$$

5. *On effectue la remontée de Hensel :*
   - *(a)* $A_2(X) = X^2 - 20X + 2 \mod 25$ *et* $B_2(X) = X^3 + 17X + 1 \mod 25$
   - *(b)* $A_3(X) = X^2 + 5X + 2 \mod 125$ *et* $B_2(X) = X^3 + 17X + 1 \mod 125$

   [!!] *Ici, on peut s'arrêter car $F(X) = A_3(X)B_3(X)$ dans $\mathbb{Z}[X]$. On a factorisé $F(X)$!*

Dans l'exemple précédent, on remarque que la remontée de Hensel nous donne directement des facteurs propres de $F(X)$ dans $\mathbb{Z}[X]$, et pas uniquement dans $(\mathbb{Z}/p^\ell\mathbb{Z})[X]$. Est-ce un résulta général ? Hélas, comme l'atteste le point suivant, la réponse est non...

• Terminons donc par une dernière conséquence négative du passage par la factorisation modulo p. Si $F(X) = F_1(X)F_2(X)\cdots F_d(X)$ où les $F_i(X)$ sont irréductibles dans $\mathbb{Z}[X]$, rien ne nous assure que ces $F_i(X)$ soient également irréductibles dans $\mathbb{F}_p[X]$ (ou dans les $(\mathbb{Z}/p^\ell\mathbb{Z})[X]$). Prenons l'exemple du polynôme X^2+1 : il est irréductible dans $\mathbb{Z}[X]$ mais se factorise comme $(X+2)(X+3)$ dans $\mathbb{F}_5[X]$. On pourrait se dire que c'est simplement un mauvais choix de p, mais malheureusement, il existe des polynômes irréductibles sur les entiers, qui ont des facteurs propres modulo n'importe quel nombre premier.

Proposition 4.35

Le polynôme $G(X) = X^4+1$ est irréductible dans $\mathbb{Z}[X]$, mais ne l'est dans aucun des $\mathbb{F}_p[X]$.

Démonstration : En exercice. ■

Remarque 4.36

En revanche, si $A(X)$ est irréductible dans $\mathbb{F}_p[X]$, alors il existe un polynôme $B(X) \in \mathbb{Z}[X]$ irréductible qui lui est congru modulo p. Par ailleurs, la taille des coefficients de $B(X)$ est majorée par la borne de Landau–Mignotte.

Cette dernière remarque nous mène à l'observation suivante. Si $F(X) = F_1(X)F_2(X)\cdots F_d(X)$ où les $F_i(X)$ sont irréductibles dans $\mathbb{Z}[X]$, alors la factorisation de $F(X)$ modulo p nous donne

$$\overline{F}(X) = \overline{G_1}(X)\overline{G_2}(X)\ldots\overline{G_k}(X)$$

où $k \geq d$, où les $\overline{G_j}(X) \in \mathbb{F}_p[X]$ sont irréductibles, et où chaque $\overline{F_i}(X)$ s'exprime comme le produit de certains $\overline{G_j}(X)$. L'ultime tâche de notre algorithme de factorisation consiste donc à chercher quel sous-ensemble des $\overline{G_j}(X)$ on doit sélectionner pour obtenir un facteur propre de $F(X)$.

4.2.4 Regroupement des facteurs

Todo: à faire

Chapitre 5

Factorisation des entiers

5.1 Introduction

5.1.1 Présentation du problème

La factorisation d'entiers est un problème ancien qui a été longuement étudié. Les mathématiciens grecs (par exemple Ératosthène) se sont questionnés sur la possibilité de réduire les nombres en un produits d'éléments simples : leurs facteurs premiers. En 1659, le mathématicien suisse Johann Rahn produisit une table des facteurs premiers d'entiers allant jusqu'à 24 000. La décennie suivante, l'anglais John Pell étendit cette table à 100 000 entrées. Dorénavant, avec l'emploi de l'ordinateurs et d'algorithmes sophistiqués, on sait factoriser des entiers de plusieurs centaines de chiffres décimaux.

Le problème de la factorisation repose sur le théorème fondamental de l'arithmétique, connu dès l'Antiquité et donné dans sa formalisation moderne par Gauss (*Disquisitiones Arithmeticae*, 1801). Il peut être énoncé de la sorte.

Problème 7 (*Factorisation d'entiers*)

Étant donné un entier $N \geq 2$, déterminer sa décomposition en facteurs premiers (unique à l'ordre près)

$$N = \prod_{i=1}^{k} p_i^{e_i}$$

où les p_i sont premiers et distincts, et les $e_i \geq 1$ sont des entiers.

Dans la suite, on choisit d'ordonner les facteurs premiers de N de la manière suivante : $p_1 > p_2 > \cdots > p_k$.

5.1.2 Complexité algorithmique et théorique

Complexité algorithmique sur les entiers. La complexité des algorithmes sera évaluée en fonction de la taille $\log_2(N)$ de l'entier à factoriser, et comptée en nombre d'opérations binaires. Par conséquent, un algorithme de factorisation est dit « polynomial » si sa complexité binaire est de la forme $O(\log(N)^c)$ avec c une constante.

Parfois, il sera intéressant de mesurer la complexité d'algorithmes dans des cas spécifiques (cas moyen, pire cas, avec présence de facteurs typiques), et en fonction de la taille $\log_2(p_i)$ de

certains facteurs (typiquement, p_1, p_2 ou p_k) ou de certains paramètres annexes (par exemple, une borne de friabilité).

On note $\mathsf{M}(N)$ le coût algorithmique de la multiplication de deux entiers $\leq N$. On rappelle dans ce contexte que $\mathsf{M}(N) = O(\log(N)^2)$ avec la méthode élémentaire de multiplication, $\mathsf{M}(N) = O(\log(N)^{1,59})$ avec la méthode de Karatsuba et $\mathsf{M}(N) = \widetilde{O}(\log(N))$ avec les meilleurs algorithmes (exemple : Schönage–Strassen).

Dans ce contexte, la division euclidienne de deux entiers $\leq N$ coûte $O(\mathsf{M}(N))$ opérations binaires et le calcul du pgcd et les coefficients de Bezout de ces entiers s'exécute en $O(\mathsf{M}(N) \log N)$ opérations binaires.

Quelle classe de complexité ? D'un point de vue **théorique**, le problème appartient à la classe de complexité NP, et il est largement conjecturé qu'il n'est pas dans P. Autrement dit, on conjecture qu'il n'existe pas d'algorithme déterministe de complexité binaire polynomiale en $\log(N)$, permettant de factoriser N.

Néanmoins, cela ne signifie pas que l'on suppose que tout algorithme factorisant N a une complexité exponentielle en $\log(N)$. En effet, il existe de nombreuses fonctions au comportement asymptotique « intermédiaire » entre $\log^d(N)$ et $2^{c \log N}$ pour tous d et c. Pour observer cela, notons $n = \log(N)$ et définissons, pour $\alpha \in [0,1]$ et $\beta \in \mathbb{R}^+$:

$$L_n[\alpha, \beta] = 2^{(\beta + o(1))\, n^\alpha \log(n)^{1-\alpha}}.$$

On remarque alors que toute fonction dans $L_n[0, \beta]$ est polynomiale en n (et β représente le degré du polynôme), et toute fonction dans $L_n[1, \beta]$ est exponentielle en n (et 2^β représente la base de l'exponentielle). Le cas où $\alpha \in]0,1[$ correspond à des fonctions dites *sous-exponentielles* ou *super-polynomiales*. La Figure **??** suivante illustre le comportement de certaines de ces fonctions.

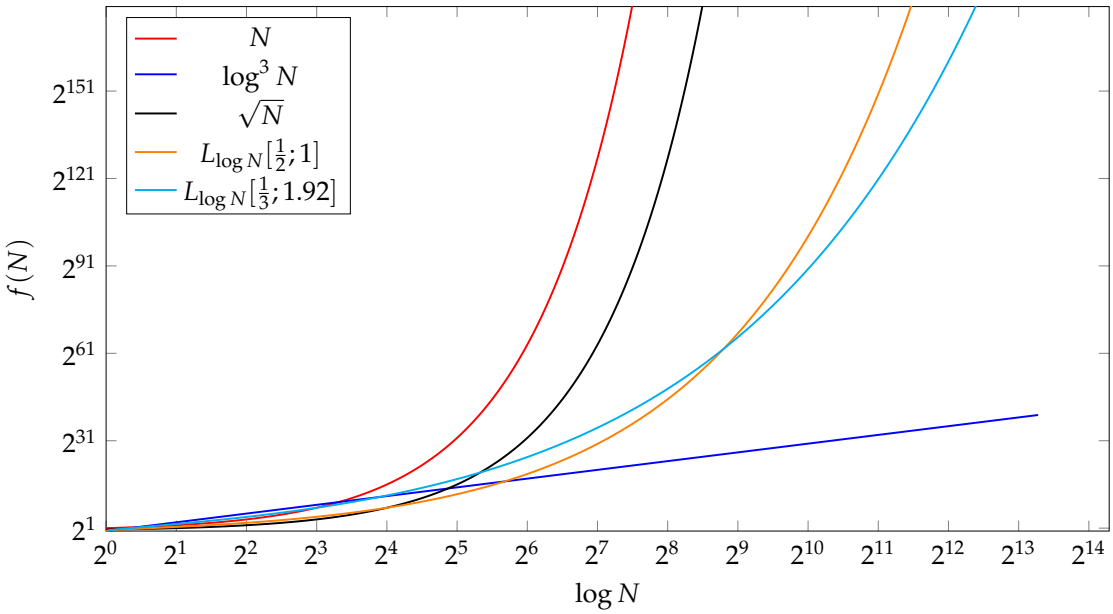


FIGURE 5.1 – Illustration du comportement de fonctions exponentielles, sous-exponentielles et polylogarithmiques en $\log(N)$.

Dans le cadre de la factorisation, une complexité en $O(\sqrt{N})$ (comme on le verra pour la méthode naïve des divisions successives) est donc dans $L_{\log N}[1, \frac{1}{2}]$: c'est une complexité expo-

nentielle. En revanche, la complexité des meilleurs algorithmes de factorisation (par exemple le crible algébrique) est sous-exponentielle : typiquement de l'ordre de $L_{\log N}[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}]$.

5.1.3 Éléments de théorie des nombres

Définition 5.1

Soit $B \geq 2$ et N un entier.

1. *On dit que N est B-friable si tout nombre premier p qui divise N vérifie $p \leq N$.*
2. *On dit que N est B-superfriable si tout élément de la forme p^e avec p premier et $e \in \mathbb{N}$ qui divise N vérifie $p^e \leq N$.*

Exemple 5.2

L'entier $720 = 2^4 \times 3^2 \times 5$ est 5-friable mais n'est pas 5-superfriable, car 2^4 et 3^2 sont supérieurs à 5. En revanche, 720 est 16-friable.

5.2 Premières méthodes

Dans toute la suite, on suppose que l'on dispose

1. d'un algorithme efficace [1] permettant de tester la primalité d'un entier,
2. d'un algorithme efficace `next_prime(a)` qui renvoie le plus petit nombre premier strictement supérieur à a.

5.2.1 Divisions successives

La méthode la plus élémentaire pour factoriser un entier consiste à tester sa divisibilité par des nombres premiers de valeur croissante (potentiellement jusqu'à $\sqrt{N}$). L'Algorithme 19 en donne une description.

Proposition 5.3

La méthode de factorisation par divisions successives :

- *trouve le plus petit facteur premier p de N en temps $\widetilde{O}(p)$;*
- *factorise complètement N en temps $\widetilde{O}(\sqrt{N})$.*

Démonstration : La méthode présentée teste la divisibilité de N par l'ensemble les nombres premiers en ordre croissant, donc tous les p_i seront détectés. Puis,
Pour chaque nombre premier p calculé dans l'algorithme, on effectue :

— un test de primalité de coût $O(\log^3(N))$,
— un test de divisibilité de coût $O(\mathsf{M}(N))$.

La complexité de la méthode dépend donc de la quantité de nombres premiers p énumérés par l'algorithme. On désigne par p' le dernier de ces nombre premier ; on a $p' = p_2$ si $e_1 = 1$ et $p' = p_1$ sinon. dans tous les cas, $p' \in O(\sqrt{N})$. Alors, on effectue $O(\pi(p'))$ tests de divisibilité, où $\pi(p')$ est le nombre

1. Par exemple, le test probabiliste de Miller–Rabin permet de tester la primalité d'un entier de n bits en temps $O(kn^3)$ avec probabilité d'échec $\leq 2^{-k}$. On sait également que l'algorithme AKS permet d'effectuer ce test de manière déterministe en temps $O(n^{12})$, qui pourrait s'améliorer en $O(n^3)$ si certaines conjectures de théorie des nombres (comme la conjecture d'Agrawal) s'avèrent correctes.

Algorithme 19 : Factorisation par divisions successives

Entrée : un entier $N \geq 2$
Sortie : la factorisation $[(p_1, e_1), \ldots, (p_k, e_k)]$

```
Initialiser n ← N, p ← 2, et L ← [ ]
Tant que n n'est pas premier faire
    Si p divise n
        ajouter (p, 0) à L
    Tant que p divise n faire
        Remplacer (p, i) par (p, i + 1) dans L
        Diviser n par p
    p ← next_prime(p)
Si n ≠ 1
    Ajouter (n, 1) à L
Retourner L.
```

de nombres premiers $\leq p'$. Comme $\pi(p') \in O(p'/\log(p'))$, pour cette partie de l'algorithme on obtient une complexité binaire en

$$O\left((\log^3(N) + \mathsf{M}(N))\frac{p'}{\log(p')}\right) \subseteq O\left(\log^3(N)\mathsf{M}(N)\frac{\sqrt{N}}{\log(\sqrt{N})}\right) \subseteq \tilde{O}\left(\sqrt{N}\right).$$

Pour les premiers p_i présents dans la factorisation de N, il faut ajouter à cela e_i divisions entières de coût binaire $O(\mathsf{M}(N))$. Comme $\sum_i e_i \leq \log(N)$, on a donc un coût binaire supplémentaire en $O(\mathsf{M}(N)\log(N))$, ce qui ne modifie pas l'ordre de grandeur de complexité de l'algorithme. ■

Remarque 5.4

Notons que l'on s'épargne la recherche du dernier facteur par un test de primalité. Ce test peut s'effectuer en temps polynomial en la taille de l'entier, de manière déterministe par le test AKS (célèbre article de Agrawal, Kayal et Saxena [AKS04]) ou plus efficacement de manière probabiliste [donner des références ici].

Pour la factorisation par divisions successives, le cas typique le plus difficile est un entier N de la forme $N = pq$ où p et q sont deux nombres premiers proches de $\sqrt{N}$. On pourrait se demander si un entier « moyen » a presque toujours deux grands facteurs premiers, auquel cas la méthode par divisions successives aurait une complexité moyenne en $\tilde{O}(\sqrt{N})$.

5.2.2 Algorithme de Fermat

On suppose ici que N est impair. Si ce n'est pas le cas, on divise N par 2 jusqu'à ce qu'il le soit. La méthode de Fermat (Algorithme 20) permet de trouver efficacement un diviseur de N très proche de $\sqrt{N}$.

L'algorithme repose sur l'observation suivante : si N s'écrit comme différence de deux carrés $N = t^2 - s^2$ (avec $t > s + 1$), alors $t - s$ et $t + s$ fournissent deux diviseurs propres de N. Réciproquement, si d_1 et d_2 sont deux diviseurs propres de $N = d_1 d_2$, alors on a

$$N = \left(\frac{d_1 + d_2}{2}\right)^2 - \left(\frac{d_1 - d_2}{2}\right)^2.$$

L'idée de Fermat est donc, à partir de $t = \lceil\sqrt{N}\rceil$ et en faisant croître t, de chercher à écrire $t^2 - N$ comme un carré.

Algorithme 20 : Factorisation par la méthode de Fermat

Entrée : un entier impair $N \geq 3$
Sortie : un diviseur propre d de N

```
Calculer t = ⌈√N⌉ et c = t² − N.
Tant que c n'est pas un carré dans ℕ faire
    calculer c ← c + 2t + 1
    incrémenter t ← t + 1
Calculer une racine carrée s de c.
Retourner t + s.
```

Proposition 5.5

Soit d le plus petit diviseur de N qui est plus grand que $\sqrt{N}$, et définissons α le réel tel que $d = (1+\alpha)\sqrt{N}$. Alors l'algorithme de Fermat (Algorithme 20) retourne la valeur de d en

$$O\left(1 + \frac{\alpha^2}{1+\alpha}\sqrt{N}\right)$$

tours de boucle.

Démonstration : Si $d \times e = N$, alors on a $e = \frac{N}{d} = \frac{1}{1+\alpha}\sqrt{N}$. Observons que l'algorithme s'arrête lorsque $t = \frac{d+e}{2}$ (voir commentaires ci-dessus). Par conséquent, comme t est incrémenté de 1 en 1 en partant de $\lceil\sqrt{N}\rceil$, il y aura approximativement (à une valeur entière près)

$$1 + \left(\frac{d+e}{2} - \sqrt{N}\right) = 1 + \left(\frac{1+\alpha}{2} + \frac{1}{2(1+\alpha)} - 1\right)\sqrt{N} = 1 + \frac{\alpha^2}{2(1+\alpha)}\sqrt{N}$$

tours de boucle. ■

Conséquence sur la complexité. En conséquent, la méthode de Fermat est très efficace dès lors que $\alpha \in O(N^{-1/4})$: dans ce cas il y a un nombre constant d'étapes ! Notons que $\alpha \in O(N^{-1/4})$ signifie que $d = N^{1/2} + O(N^{1/4})$, autrement dit les diviseurs d et e ont en commun la moitié deurs premiers chiffres : ils sont donc très proches (en comparaison relative).

En revanche, si d est de l'ordre de $N^{1/2+\beta}$ avec $\beta > 0$ (ce qui est le cas le plus souvent), cela signifie que $\alpha = N^{\beta}$, donc que l'algorithme de Fermat exécutera $O(N^{1/2+\beta})$ tours de boucle : c'est pire que la méthode par divisions successives. En particulier, pour $N = 3d$ et d premier, la méthode de Fermat effectue $O(N)$ tours de boucle...

Remarque 5.6

Il est possible de coupler astucieusement les méthodes par divisions successives (pour éliminer les petits facteurs) et de Fermat (pour éliminer les facteurs proches de $\sqrt{N}$) pour aboutir à une méthode de factorisation en temps $O(N^{1/3})$.

5.2.3 Méthode ρ de Pollard

5.2.3.1 Contexte mathématique

La méthode ρ de Pollard repose sur des propriétés de collisions de suites récursives à valeurs finies. Dans un cadre général, considérons E un ensemble fini de cardinal M, et f une

application $E \to E$. Étant donné $x_0 \in E$, on définit une suite récurrente $(x_t)_{t\geq 0}$ par

$$x_{t+1} = f(x_t) \quad \text{pour } t \geq 0\,.$$

Alors on sait que $(x_t)_{t\geq 0}$ est ultimement périodique :

$$\exists \tau \geq 1, \exists T \geq \tau, \forall t \geq T,\ x_t = x_{t-\tau}\,. \tag{5.1}$$

En effet, parmi les $M+1$ premiers termes de la suite, au moins deux d'entre eux sont identiques car $|E| = M$. Appelons-les x_i et x_j. Alors on a $x_{i+1} = f(x_i) = f(x_j) = x_{j+1}$ et par induction, $x_{i+t} = x_{j+t}$ pour tout $t \geq 0$.

Définition 5.7

Soit $\boldsymbol{x} = (x_t)_{t\geq 0} \in E^{\mathbb{N}}$ une suite récurrente sur E. Le plus petit $\tau \geq 1$ vérifiant (5.1) *est appelé la* période *de $\boldsymbol{x}$. Le plus petit $T \geq \tau$ tel que $x_{T-\tau} = x_T$ est appelé la* prépériode *de $\boldsymbol{x}$.*

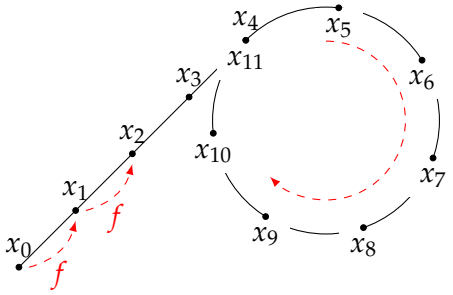


FIGURE 5.2 – Illustration du « ρ » obtenu par l'application successive de $f : E \to E$ avec E fini. Sur cet exemple, la période est 7 et la prépériode est 11.

On peut se questionner sur la taille typique de τ et T lorsque f et x_0 sont aléatoires.

Lemme 5.8 (*Paradoxe des anniversaires*)

Soit X une variable aléatoire de loi uniforme sur E de cardinal M. Pour $m \geq 1$, on note $S_m = \{x_0, \ldots, x_{m-1}\}$, où chaque x_i est tiré indépendamment selon X.
Alors,

$$\mathbb{P}(|S_{m+1}| = m+1) = \prod_{i=1}^{m} \left(1 - \frac{i}{M}\right), \quad \forall m \in \mathbb{N}\,.$$

En particulier, pour m fixé et $M \to \infty$, on a l'équivalent

$$\mathbb{P}(|S_{m+1}| = m+1) \sim e^{-m(m+1)/2M}\,.$$

Démonstration : Par indépendance des tirages, on a :

$$\begin{aligned}\mathbb{P}(|S_{m+1}| = m+1) &= \mathbb{P}(x_1 \notin \{x_0\}) \times \cdots \times \mathbb{P}(x_{m-1} \notin \{x_0, \ldots, x_{m-2}\}) \\ &= \left(1 - \frac{1}{M}\right) \times \cdots \times \left(1 - \frac{m}{M}\right) = \prod_{i=1}^{m} \left(1 - \frac{i}{M}\right)\end{aligned}$$

Pour l'équivalent asymptotique, on observe que $e^{-i/M} = 1 - \frac{i}{M} + O(\frac{1}{M^2})$. Il s'ensuit (produit fini d'équivalents) que

$$\prod_{i=1}^{m} \left(1 - \frac{i}{M}\right) \sim \prod_{i=1}^{m} e^{-i/M} \sim e^{-m(m+1)/2M}\,.$$

■

La conséquence majeure du Lemme 5.8 la probabilité qu'une suite récurrente aléatoire ait une prépériode $T \gg \sqrt{M}$ est exponentiellement petite. En effet, si l'on suppose que f a un « comportement aléatoire » sur E et si on pose $x_1 = f(x_0), \ldots, x_{n-1} = f(x_{n-2})$, alors $\mathbb{P}(T > m) = \mathbb{P}(|S_m| = m)$.

Remarque 5.9

Flajolet et Odlyzko ont par ailleurs démontré par des méthodes analytiques que la taille moyenne de $\tau \sim \sqrt{\pi M/8}$ et celle de $T \sim \sqrt{\pi M/2}$. Voir [FO89] pour les détails.

5.2.3.2 Application à la factorisation

Pollard propose d'utiliser les résultats précédents afin de déterminer le plus petit facteur premier de N (ou, *a minima*, l'un de ses petits facteurs premiers). Notons donc $p \ll N$ un diviseur premier de N, que nous ne connaissons pas *a priori*.

Avec les notatins de la section précédente, l'idée est alors de poser $E = \mathbb{Z}/N\mathbb{Z}$, et de construire une suite récurrente aléatoire $\boldsymbol{x}$ sur E. Alors, la suite « $\boldsymbol{x} \mod p$ » définie comme $(x_t \mod p)_{t\geq 0}$, est également récurrente et aléatoire. Comme elle est définie sur $\mathbb{Z}/p\mathbb{Z}$ de cardinal p, sa prépériode est donc en $O(\sqrt{p})$.

Par ailleurs, **sans connaître** p, nous pouvons détecter une éventuelle collision de la suite $\boldsymbol{x}$ modulo p. En effet, si $x_i \equiv x_j \mod p$, alors p divise à la fois $x_i - x_j$ et N, donc p divise $\text{pgcd}(x_i - x_j, N)$. Le calcul de ce pgcd s'effectue efficacement, avec la connaissance seule de N et de $x_i, x_j \in \mathbb{Z}/N\mathbb{Z}$.

On espère donc que $\text{pgcd}(x_i - x_j, N)$ soit différent de N, afin d'obtenir un multiple de p (éventuellement lui-même) qui soit un diviseur propre de N. Le fait que ce diviseur soit propre se produit avec une probabilité très forte. En effet, si $\text{pgcd}(x_i - x_j, N) = N$, cela signifie que $x_i \equiv x_j \mod N$. Autrement dit, on a choisi x_0 tel que la première collision de la suite (x_t) se réalise en même temps modulo N et modulo p.

Pour des raisons d'efficacité on choisit pour $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$ une application *déterministe* et calculatoirement peu coûteuse. Typiquement, f est une fonction polynomiale de bas degré.. L'aléa de la suite récurrente aléatoire $\boldsymbol{x}$ repose donc uniquement sur le choix de $x_0 \in \mathbb{Z}/N\mathbb{Z}$. Heuristiquement, l'application $f : z \mapsto z^2 + 1$ est souvent choisie car elle produit des suites avec un bon comportement pseudo-aléatoire [2].

À des fins pédagogiques, écrivons une première version de l'algorithme.

Calculons la complexité de cet algorithme. Notons A le nombre de passage dans la boucle `while`. Notons qu'en entrée de boucle, la liste L est de taille A. Si $d = 1$ en sortie de boucle, on a donc effectué A calculs de pgcd a effectuer. D'après l'argumentation qui précède, on va effectuer en moyenne $O(\sqrt{p})$ fois cette boucle, où p et le plus petit diviseur premier de N. Donc, en l'état, la complexité de la méthode ρ est en $O(p)$; ce n'est pas mieux que l'algorithme par divsions susscessives.

Le problème est que l'on cherche des collisions en parcourant exhaustivement l'ensemble des paires (x_i, x_j) calculées. Ne peut-on pas trouver les collisions plus efficacement en restreignant l'ensemble de recherche ? Une réponse positive est apportée un résultat de Floyd.

Lemme 5.10 (*Lemme de Floyd*)

Si $\boldsymbol{x} = (x_t)_{t\geq 0}$ est une suite ultimement périodosie de période τ et de prépériode T, alors il existe $i \leq T$ tel que $x_i = x_{2i}$.

2. pour davantage de détails, voir les genéarateurs de nombres pseudo-aléatoires Blum–Blum–Shub

Algorithme 21 : Version **naïve** de la méthode ρ de Pollard

Entrée : un entier N composé
Donnée externe : une fonction $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$, typiquement $f(z) = z^2 + 1 \mod N$
Sortie : un diviseur propre d de N

```
1  Tirer x uniformément dans {0, ..., N − 1}
2  Initialiser L ← [x]
3  Initialiser d ← 1
4  Tant que d = 1 faire
5      Calculer x ← f(x)
6      Pour tout y ∈ L faire
7          Calculer d = pgcd(x − y, N)
8          Si d ≠ 1
9              Sortir de la boucle prématurément.
10     Ajouter x à la liste L.
11 Si d = N
12     Revenir à l'étape 1.
13 Retourner d.
```

Démonstration : On oberse simplement que la différence d'indices $2i - i = i$ sera divisible par τ pour un certain $i \geq T - \tau$. ∎

Exemple 5.11

Voici une illusatration du lemme de Floyd, avec $N = 11 \times 13 = 143$, *donc* $\sqrt{N} \lesssim 12$ *et* $\sqrt{p} = \sqrt{11} \lesssim 4$.

Avec le choix $x_0 = 133$, *correspondant à une période* 2 *et une prépériode* 4 *:*

i	x_i	$y_i = x_{2i}$
0	133	133
1	101	49
2	49	127
3	114	127
4	127	127
5	114	127
6	127	127
7	114	127
8	127	127

Avec le choix $x_0 = 34$, *correspondant à une période* 4 *et une prépériode* 4

i	x_i	$y_i = x_{2i}$
0	34	34
1	13	27
2	27	83
3	15	105
4	83	83
5	26	105
6	105	83
7	15	105
8	83	83

Par conséquent, on peut modifier l'Algorithme 21 pour accélérer la recherche de collision, en ne les cherchant que parmi les couples de la forme (x_i, x_{2i}). Pour cela, on construit la suite $y_i = x_{2i}$ en appliquant deux fois la fonction f à chaque tour de boucle.

Proposition 5.12

La méthode ρ *de Pollard calcule un diviseur propre d'un entier* N *en temps moyen* $O(\sqrt{p}\log^2 N)$, *où* p *est le plus petit facteur premier de* N.

Algorithme 22 : Méthode ρ de Pollard

Entrée : un entier N composé
Donnée externe : une fonction $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{Z}/N\mathbb{Z}$, typiquement $f(z) = z^2 + 1 \mod N$
Sortie : un diviseur propre d de N

1 Tirer x uniformément dans $\{0, \dots, N-1\}$
2 Initialiser $y \leftarrow x$
3 Initialiser $d \leftarrow 1$
4 **Tant que** $d = 1$ **faire**
5 Calculer $x \leftarrow f(x)$
6 Calculer $y \leftarrow f(f(y))$
7 Calculer $d = \text{pgcd}(x - y, N)$
8 **Si** $d = N$
9 Revenir à l'étape 1.
10 **Retourner** d.

Démonstration : La validité de l'algorithme découle des discussions précédentes. En ce qui concerne sa complexité, on n'a maintenant plus qu'un pgcd à calculer par tour de boucle, c'est-à-dire $O(\sqrt{p})$ pgcd au total, en moyenne. Chaque calcul de pgcd a un coût polynomial en $\log N$. ■

Remarque 5.13

La complexité de la méthode ρ de Pollard reste exponentielle en la taille de l'entrée, dans le pire cas où $N = pq$ et $p, q \sim \sqrt{N}$, car elle est en $O(N^{1/4})$. Néanmoins, cette méthode permet de trouver rapidement et simplement les petits diviseurs de N (s'il en admet), et est donc utilisée en pratique pour éliminer ces diviseurs.

Exemple 5.14

Illustrons maintenant la méthode ρ de Pollard dans son ensemble, avec $N = 4307$.

Choix : $x_0 = 2747$

x_i	$y_i = x_{2i}$	$\text{pgcd}(x_i - y_i, N)$
146	4089	1
4089	370	1
148	3451	1
370	2027	1
3384	370	1
3451	3451	4307

Pas de chance...

Choix : $x_0 = 2748$

x_i	$y_i = x_{2i}$	$\text{pgcd}(x_i - y_i, N)$
1334	766	1
766	2188	1
1005	1267	1
2188	620	1
2268	3502	1
1267	2435	73

C'est bon : $N = 73 \times 59$

Enfin nous n'avons pas commenté la probabilité de « mal tirer » le premier élément x_0 de la suite itérée. Todo: à faire (un peu long ?)

5.3 Méthodes spéciales

Dans cette section, nous allons présenter des méthodes de factorisation qui utilisent la forme particulière de certains facteurs premiers de N. Les deux premières méthodes ($p-1$ de Pollard et $p+1$ de Williams) sont assez proches dans leur nature et permettent de trouver des facteurs

très particuliers de N, tandis qu'ECM (*Elliptic Curve Method*) propose une extension à des entiers N plus génériques.

5.3.1 Méthode $p-1$ de Pollard

En 1974, John Pollard [Pol74] propose une méthode pour calculer les facteurs premiers p d'un entier N tels que $p-1$ est superfriable. Pour cela, il s'appuie sur le petit théorème de Fermat, dont on rappelle l'énoncé ci-dessous.

Theorème 5.15 (*Petit théorème de Fermat*)

Soit p un nombre premier. Alors, pour tout a non-multiple de p, on a

$$a^{p-1} \equiv 1 \mod p\,.$$

En conséquence du petit théorème de Fermat, si M est un multiple de $p-1$ et si p ne divise pas a, alors p divise a^M-1. Autrement dit, on peut espérer obtenir p (ou un de ses multiples) en calculant $\text{pgcd}(a^M-1, N)$ pour un M mutliple de $p-1$. Mais comment obtenir un multiple de $p-1$ si on ne connaît pas p ?

Supposons donc que p soit un facteur premier de N de taille assez grande, mais tel que $p-1$ est B-superfriable (rappel : toutes les puissances de premiers q^e qui divisent $p-1$ sont plus petites de B). L'idée est alors de choisir l'entier M comme le produit de toutes puissances de premiers $q^e \leq B$. Ce nombre M peut être de taille immense, mais il faut retenir que construire un grand nombre M (puis calculer $a^M \mod N$) est une opération que l'on peu réaliser très efficacement.

On peut maintenant résumer la méthode du « $p-1$ de Pollard » dans l'Algorithme 23.

Algorithme 23 : Méthode du $p-1$ de Pollard.

Entrée : un entier N à factoriser, une borne $B \geq 2$
Hypothèse : N admet un facteur premier p tel que $p-1$ est B-superfriable
Sortie : un facteur propre de N

```
Calculer M ← ppcm(2, ... B)
d ← N
Tant que d = N faire
    Tirer aléatoirement a ∈ {2, ..., N − 1}
    Calculer b ← pgcd{a, N}
    Si b ≠ 1
        Retourner b
    Calculer efficacement d ← pgcd(a^M − 1, N)
Retourner d.
```

Onservons que dans l'Algorithme 23, on a écrit une boucle `while` pour laquelle on teste si N ne divise pas la valeur de $d = \text{pgcd}(a^M-1, N)$ calculés. En effet, dans ce cas on n'obtiendrait pas de facteur propre de N.

Ce cas pathologique se produit lorsque l'ordre de a modulo N (et pas seulement modulo p) est divisible par M. Autrement dit, lorsque l'ordre de a modulo N est B-superfriable. Deux cas peuvent se présenter :

1. Ce phénomène se produit pour tout a. Alors, cela signifie que **tous** les facteurs premiers p de N sont B-superfriables. Solution : relancer l'algorithme avec un B plus petit.

2. Ce prénomène ne se produit pas pour les les a. Solution : on tire un nouveau a aléatoirement.

Complexité. Remarquons d'abord que

$$M = \mathrm{ppcm}(2, \ldots, B) = \prod_{\text{premier} p \leq B} p^{\lfloor \log_p(B) \rfloor}$$

Cela permet un calcul efficace (en $O(B \log B)$ opérations sur les entiers) de M lorsqu'on connaît la liste des nombres premiers inférieurs à B. On peut ensuite démontrer[3] que $M = \mathrm{ppcm}(B) \sim e^B$. Cela signifie donc que le calcul de $a^M - 1 \mod N$ se fait en $O(B)$ opérations dans $\mathbb{Z}/N\mathbb{Z}$, donc (en supposant un nombre constant de tirages de a) que la complexité totale de l'algorithme est donc grossièrement en $O(B \log B \log^2 N)$.

Remarque 5.16

Écrivons

$$M = \prod_{\text{premier} p \leq B} p^{\lfloor \log_p(B) \rfloor} = Q_1 \times \cdots \times Q_\ell \quad \text{où } Q_\ell := p_\ell^{\lfloor \log_{p_\ell}(B) \rfloor}.$$

On peut alors accélérer sensiblement la méthode $p-1$ en calculant itérativement des $\mathrm{pgcd}(a^{M_i} - 1, N)$ *pour $M_i = Q_1 \times \cdots \times Q_i$, voir exemple suivant.*

Exemple 5.17

Pour $N = 108\,147\,037$, on va exécuter la variante avec les pgcd intermédiaires. Pour l'exemple, on choisit $B = 14$ (on pourrait aller plus loin). On a donc $M = Q_1 \times \cdots \times Q_\ell = 2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$ et on note $M_i = Q_1 \times \cdots \times Q_i$ pour les calculs intermédiaires.

1. *Exemple avec le choix $a = 2$:*

Q_i	$a^{M_i} - 1 \mod N$	$\mathrm{pgcd}(a^{M_i} - 1, N)$
8	255	1
9	77888826	1
5	14060764	1
7	66102662	1
11	53395803	1
13	92398868	36037

   On obtient le facteur $p = 36037$, et $p - 1 = 2^2 \cdot 3^2 \cdot 7 \cdot 11 \cdot 13$.

2. *Exemple avec le choix $a = 20$*

Q_i	$a^{M_i} - 1 \mod N$	$\mathrm{pgcd}(a^{M_i} - 1, N)$
8	77299267	1
9	45988448	1
5	97367445	3001
7	×	×
11	×	×
13	×	×

   On obtient rapidement le facteur $p = 3001$, ce qui permet d'économiser des calculs.

3. c'est assez technique, voir par exemple les équivalents de la seconde fonction de Tchebychev

5.3.2 Méthode $p+1$ de Williams

Prenons un peu de hauteur sur la dernière section. Pour la méthode $p-1$ de Pollard, on cherche à obtenir un diviseur propre de N en calculant $\text{pgcd}(a^M - 1, N)$ pour un certain $a \in (\mathbb{Z}/N\mathbb{Z})^\times$, et utilisant le fait que

$$p-1 \text{ divise } M \quad \Longleftrightarrow \quad a^M \equiv 1 \mod p\,.$$

Lorsque $p-1$ est superfriable, un choix de M « pas trop grand » permet alors de remplir la condition de gauche sans connaître p. Pour résumer, on s'est placé dans le groupe $(\mathbb{Z}/N\mathbb{Z})^\times$, qui admet un sous-groupe dont l'ordre $p-1$ est superfriable.

En 1982, Williams [Wil82] propose une extension de cette méthode dans le cas où ce n'est pas $p-1$ mais $p+1$ qui est superfriable. L'idée est de se placer dans un autre groupe (constructible avec la connaissance seule de N) qui admet un sous-groupe d'ordre $p+1$. Pour cela, Williams propose de fixer $D \in \mathbb{Z}$ et de s'intéresser à :

$$\mathcal{A} = \{(x,y) \in (\mathbb{Z}/N\mathbb{Z})^2 \mid x^2 - Dy^2 \equiv 1 \mod N\} \subseteq (\mathbb{Z}/N\mathbb{Z})^2$$

Proposition 5.18

Muni du produit

$$(x,y)\cdot(u,v) = (xu + Dyv, xv + yu)$$

l'ensemble $\mathcal{A}$ forme un groupe commutatif de neutre $(1,0)$. L'inverse de (x,y) est alors $(x,-y)$.

Démonstration : Il est clair que $(1,0) \in \mathcal{A}$. Par ailleurs, si $(x,y) \in \mathcal{A}$, on a aussi $(x,-y) \in \mathcal{A}$ car $x^2 - D(-y)^2 = x^2 - Dy^2 \equiv 1 \mod N$.

— Vérifions que $\cdot$ est bien une loi de coposition interne. Si (x,y) et (u,v) sont dans $\mathcal{A}$, alors leur produit $(xu+Dyv, xv+yu)$ vérifie

$$\begin{aligned}(xu+Dyv)^2 - D(xv+yu)^2 &\equiv (xu)^2 + 2Dxyuv + D^2(yv)^2 - D(xv)^2 - 2Dxyuv - D(yu)^2 \mod N\\ &\equiv (1+Dy^2)(1+Dv^2) + D^2(yv)^2 - D(1+Dy^2)v^2 - Dy^2(1+Dv^2) \mod N\\ &\equiv 1 \mod N\end{aligned}$$

— Cette loi est clairement commutative.

— La loi est associative :

$$\begin{aligned}\Big((x,y)\cdot(u,v)\Big)\cdot(a,b) &= (xu+Dyv, xv+yu)\cdot(a,b)\\ &= (xua + Dyva + Dxvb + Dyub, xub + Dyvb + xva + yua)\\ &= \Big(x(ua+Dvb) + Dy(va+ub), x(va+ub) + y(ua+Dvb)\Big) \quad = (x,y)\cdot\Big((u,v)\cdot(a,b)\Big)\end{aligned}$$

— On vérifie enfin que $(x,y)\cdot(x,-y) = (x^2 - Dy^2, xy - yx) = (1,0)$. ■

La projection de $\mathcal{A}$ modulo p est notée $\mathcal{A}_p = \{(x \mod p, y \mod p) \mid (x,y) \in \mathcal{A}\}$. Alors, on a le résultat suivant.

Proposition 5.19

Si D n'est pas un carré modulo p, alors $\mathcal{A}_p$ est isomorphe au sous-groupe $\mathcal{U}_p = \{u \in \mathbb{F}_{p^2} \mid u^{p+1} = 1\}$ de $\mathbb{F}_{p^2}^\times$, où $\mathbb{F}_{p^2}$ est construit comme $\mathbb{F}_p[X]/(X^2 - D) = \mathbb{F}_p[\sqrt{D}]$. Plus précisément, l'isomorphisme est donné par :

$$(x,y) \in \mathcal{A}_p \longleftrightarrow x + \sqrt{D}y\,.$$

Par ailleurs, l'ordre de $\mathcal{U}_p$ est $p+1$.

Démonstration : Si D n'est pas un carré modulo p, on peut bien s'intéresser au corps $\mathbb{F}_{p^2} = \mathbb{F}_p[\sqrt{D}]$. L'ensemble $\mathcal{U}_p$ n'est rien d'autre que l'ensemble des éléments de norme 1 ; rappelons que c'est bien un sous-groupe de $\mathbb{F}_{p^2}^\times$ d'ordre $p+1$. Il suffit maintenant de montrer que pour tout $(x,y) \in \mathcal{A}_p$, on a bien $(x+\sqrt{D}y)^{p+1} = 1$. Pour cela, rappelons que $\sqrt{D}^p = -\sqrt{D}$ (ce sont les racines conjuguées du polynôme $X^2 - D$ qui définit le corps $\mathbb{F}_{p^2}$), ce qui permet d'obtenir :

$$(x+\sqrt{D}y)^{p+1} = (x+\sqrt{D}y)^p(x+\sqrt{D}y) = (x-\sqrt{D}y)(x+\sqrt{D}y) = x^2 - Dy^2 = 1\,.$$

■

Maintenant, on peut imiter la méthode $p-1$, en considérant $u_0 = x_0 + y_0\sqrt{D} \in \mathcal{A}$, en calculant $u_M = u^M$ et en espérant que $\gcd(x_M - 1, N)$ ou que $\gcd(y_M, N)$ donne un facteur propre de N.

Focalisons-nous sur le calcul de $\gcd(x_M - 1, N)$. On aimerait que son exécution soit sensiblement aussi rapide que le calcul de $a^M \mod N$ dans la méthode $p-1$. Ceci veut dire que l'on va adapter la méthode d'exponentiation binaire dans le groupe $\mathcal{A}$. Détaillons un peu les calculs.

Pour effectuer l'exponentiation binaire, on doit exprimer x_{2n} et x_{2n+1} en fonction de précédents x_i. Si (x_n,y_n) est défini par $u^n = x_n + y_n\sqrt{D}$, alors :

- On a $u^{2n} = (x_n + y_n\sqrt{D})^2 = (x_n^2 + Dy_n^2) + 2x_ny_n\sqrt{D} = (2x_n^2 - 1) + 2x_ny_n\sqrt{D}$. Donc,

  $$x_{2n} = 2x_n^2 - 1$$

  et $y_{2n} = 2x_ny_n$.
- D'une part, $u^{2n+1} = u^{2n}u$ donne $x_{2n+1} = x_{2n}x_1 + y_{2n}y_1D = (2x_n^2 - 1)x_1 + 2x_ny_ny_1D$. D'autre part, $u^{n+1} = u^nu$ donne $x_{n+1} = x_nx_1 + Dy_ny_1$.
- Donc,

  $$x_{2n+1} = (2x_n^2 - 1)x_1 + 2x_n(x_{n+1} - x_nx_1) = 2x_nx_{n+1} - x_1\,.$$

  et bien sûr

  $$x_{2n+2} = 2x_{n+1}^2 - 1$$

Donc, la connaissance du couple (x_n, x_{n+1}) permet de déduire les couples (x_{2n}, x_{2n+1}) et (x_{2n+1}, x_{2n+2}). Au cours du calcul, on va donc devoir garder en mémoire, non pas la valeur de x_n seule, mais des couples de la forme (x_n, x_{n+1}).

On peut maintenant parfaitement adapter la méthode $p-1$ de Pollard (observez les similiudes !), pour obtenir la méthode $p+1$ de Williams dans l'Algorithme 24.

Remarque 5.20

1. *Dans l'Algorithme 24, en initialisant un x_1 aléatoire et en imposant implicitement qu'il correspond à un élément de $\mathcal{A}$, on détermine implicitement une valeur de D. Avec bonne probabilité sur le tirage, D sera un non-résidu quadratique modulo p.*
2. *Des accélérations calculatoires sont possibles en calculant la suite $v_n = 2x_n$ à la place de la suite x_n.*

5.3.3 Méthode ECM

Todo: à faire

Algorithme 24 : Méthode $p+1$ de Williams.

Entrée : un entier N à factoriser, une borne $B \geq 2$
Hypothèse : N admet un facteur premier p tel que $p+1$ est B-superfriable
Sortie : un facteur propre de N

Calculer $M \leftarrow \text{ppcm}(2, \dots B)$
$d \leftarrow N$
Tant que $d = N$ **faire**
  Tirer aléatoirement $x_1 \in \{2, \dots, N-1\}$
  Calculer $b \leftarrow \text{pgcd}\{x_1, N\}$
  Si $b \neq 1$
    Retourner b
  Calculer efficacement $d \leftarrow \text{pgcd}(x_M - 1, N)$, en utilisant les relations de récurrences données précédemment.
Retourner d.

5.4 Méthodes génériques

5.4.1 Crible quadratique

5.4.1.1 Stratégie de factorisation

5.4.1.2 Construction de la base de facteurs

5.4.1.3 Effritement

5.4.1.4 Étape d'algèbre linéaire

5.4.1.5 Complexité et résultats pratiques

5.4.2 Crible algébrique

Chapitre 6

Calcul de logarithme discret

6.1

6.2 Résumé, chronologie et perspectives

Bibliographie

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2) :781–793, 2004.

[Atk92] A. O. L. Atkin. Probabilistic primality testing. *Inria Research Report 1779*, pages 159–163, 1992. summary by F. Morain.

[BCG+17] Alin Bostan, Frédéric Chyzak, Marc Giusti, Romain Lebreton, Grégoire Lecerf, Bruno Salvy, and Éric Schost. *Algorithmes Efficaces en Calcul Formel*. Frédéric Chyzak (auto-édit.), 2017.

[Ber67] Elwyn R. Berlekamp. Factoring polynomials over finite fields. *The Bell System Technical Journal*, 46(8) :1853–1859, 1967.

[Ber70] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111) :713–735, 1970.

[FO89] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.

[KCY+06] F. Kong, Z. Cai, J. Yu, , and D. Li. Improved generalized atkin algorithm for computing square roots in finite fields. *Information Processing Letters*, 98 :1–5, 2006.

[LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their Applications, 2nd edition*. Cambridge University Press, 1994.

[Pol74] John M. Pollard. Theorems of factorization and primality testing. In *Proceedings of the Cambridge Philosophical Society*, volume 76, pages 521–528, 1974.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.

[Wie86] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1) :54–62, 1986.

[Wil82] H.C. Williams. A $p + 1$ method of factoring. *Mathematics of Computation*, 39(159), 1982.