Cours de Calcul Formel : Résumé

Eric Sanlaville

17 juin 2009

Table des matières

Chapitre 1

Introduction

1.1 Présentation

On voit souvent un ordinateur comme un calculateur, c'est à dire une machine calculant les résultats d'opérations sur des nombres. Mais un ordinateur peut également manipuler des symboles. Il peut même manipuler des formules, des équations,... sur ces symboles. Ces manipulations sont importantes pour des applications très variées. Le premier intérêt est de pouvoir obtenir des résultats exacts (c'est à dire sans approximation). Le second est d'obtenir des résultats impossibles par un calcul numérique (par exemple la simplification d'une expression ou d'une fraction, l'expression d'une primitive) mais très utiles pour des applications en physique, mécanique,...

Définition 1 (Essai de première définition) *Le calcul formel est la discipline de la manipulation des symboles mathématiques, en vue d'obtenir les résultats exacts de calculs complexes, essentiellement sur des nombres ou des polynômes.*

Terminologies employées : calcul formel, calcul symbolique, algèbre computationnelle. Le troisième terme (computer algébra) met bien l'accent sur le domaine mathématique le plus concerné par le calcul formel, l'algèbre.

1.2 Liens et prérequis

Fondements mathématiques utilisés :
- Arithmétique (théorie des nombres, opérations élémentaires) ;
- Algèbre (structures : anneau et corps, équations, polynômes) ;
- Algèbre Linéaire (systèmes d'équations, matrices et déterminants)

Algorithmique :
- correction des algorithmes ;
- complexité algorithmique en temps et espace ;
- algorithmique exacte : structure de données, calculs exacts.

Domaines connexes :

- Calcul numérique (ou calcul en virgule flottante) : effectue des calculs complexes en privilégiant la vitesse à la précision. On travaille avec à chaque étape de calcul une précision fixée. Les algos sont souvent de type itératif (basés sur des résultats de convergence théorique de suites vers les résultats cherchés). Il est intéressant de comparer les 2 approches.
  Exemple : Newton, optimisation non linéaire.
- Recherche Opérationnelle (Optimisation Combinatoire) : Il s'agit moins ici d'effectuer un calcul que de résoudre un problème de grande taille, de type optimisation. Les contraintes du problème définissent en général un domaine fini mais de grande taille. La difficulté consiste ici à trouver la meilleure solution dans ce domaine. Si le problème est bien structuré, on dispose de méthodes rapides (Gloutonne, Descente). Dans le cas contraire, on doit effectuer une énumération implicite.
  Exemple : Programmation Linéaire, Flots de coût minimum dans un réseau, Programmation Linéaire en Nombres Entiers.
  Dans certaines applications, il est souhaitable (et possible) de résoudre ces problèmes en arithmétique exacte.

1.3 Historique

- 1953 : premier programme de dérivation formelle.
- 1958 : langage LISP.
- Années 60 : programmes de manipulation de polynômes, d'intégration. Logiciel reduce (LISP), MATLAB.
- Années 70 : logiciel Macsyma (LISP). Scratchpad (IBM)
- Années 80 : MAPLE (Waterloo, C)
- Années 90 : Mathematica (système "grand public", essort en milieu industriel). MUPAD (Paderborn).

1.4 Bibliographie

Le cours s'appuie essentiellement sur les deux livres ci-dessous, mais de nombreuses autres sources existent. Il est également simple de trouver de nombreux tutoriels mupad sur internet. Celui de Damien Olivier nous servira de support.

Philippe Saux-Picart : Cours de calcul formel. Algorithmes fondamentaux. Ellipses, 1999

Joachim von zur Gathen and Jürgen Gerhard : Modern Computer Algebra, seconde édition, Cambridge University Press, 2003.

Damien Olivier : Introduction à la programmation scientifique. MUPAD, http ://litis.univ-lehavre.fr/˜ olivier/enseignement/l1 /cours/MuPAD/support/Programmation_scientifique_polyp.pdf

Chapitre 2

Les bases du calcul formel

2.1 Questions de représentation

2.1.1 Représentation des grands nombres entiers

Dans les langages de programmation classiques, les nombres sont classés par types (entiers, réels,...). Chaque élément d'un type donné occupe une place fixée en mémoire. Par exemple un entier sera souvent stocké dans 2 ou 4 octets. En conséquence, le type entier ne peut contenir que 2^{32} éléments distincts, et le plus grand entier utilisable sera $2^{31} - 1$, soit environ deux milliards ($2 \cdot 10^9$). Une des raisons à cela est que les registres de la machine sont de taille fixe. Une opération élémentaire comme une addition entre deux entiers ne peut se faire en un temps constant (un top de l'horloge), que si ces nombres sont de taille inférieure à la taille des registres. Mais il se trouve que dans de nombreuses applications modernes comme la cryptographie, il est nécessaire de manipuler des entiers arbitrairement grands. De même une suite d'opérations arithmétiques simples, mais effectuées de manière exacte, peut entraîner une grande augmentation du nombre de chiffres des nombres rationnels manipulés (exemple : suite de divisions). Tout système qui attribuera aux entiers ou aux rationnels une taille maximale sera donc inadéquat. La solution est de stocker les nombres (qu'ils soient entiers, réels ou complexes) sous forme de listes.

La solution évidente est de stocker un nombre comme la liste de ses chiffres dans la base classique, la base 10. Mais on aboutit à des listes très longues. On peut tout à fait stocker dans chaque place de la liste un nombre plus grand, du moment que sa taille n'excède pas la taille des nombres stockables dans un registre. C'est à dire que l'on écrira ce nombre en base B, $B >> 10$. Par exemple $B = 10^{4~}$; ceci permet de diviser par 4 la taille de la liste, tout en permettant des calculs rapides sur chaque élément de la liste (les chiffres en base B).

On notera généralement $(x_k, x_{k-1}, ..., x_o)_B$ l'écriture du nombre entier x en base B. Dans le cas de nombres contenant un grand nombre de chiffres nuls (nombre *creux*) il pourra être avantageux de stocker seulement la liste des couples (exposant de B, coefficient non nul).

2.1.2 Représentation d'autres nombres

A partir de la représentation des nombres entiers, on peut facilement représenter d'autres ensembles de nombres. Pour les entiers relatifs, il suffit de rajouter un champ *signe* à la liste des chiffres. Un nombre rationnel sera représenté par sa fraction (éventuellement irréductible), donc par 2 listes pour le numérateur et le dénominateur.

Un nombre irrationnel pourra souvent être représenté par un symbole (π) ou par une expression ($\sqrt{2}$, $3 \cdot \pi$,...) qui lui est caractéristique.

2.1.3 Représentation des polynômes

Un polynôme monovarié de degré n est une fonction de $\mathbb{R}$ dans $\mathbb{R}$ en général, qui s'écrit sous forme :

$$a_n X^n + a_{n-1} X^{n-1} + \ldots + a_1 X + a_0, \quad a_n, a_{n-1}, \ldots, a_1, a_0 \in \mathbb{R}.$$

Quand $a_n = 1$, on dit que P est *unitaire*. On note $\mathbb{R}[X]$ l'ensemble de ces polynômes sur $\mathbb{R}$. De manière générale on notera $F[X]$ l'ensemble des polynômes monovariés définis sur un ensemble F, où F est un corps ou un anneau. On voit immédiatement qu'un polynôme peut être représenté de manière similaire à un grand nombre, par la liste de ses coefficients. La seule limite est que ces coefficients doivent pouvoir être stockés avec une taille finie. Mais cette limite peut être levée si ces coefficients eux-mêmes sont stockés sous forme de liste suivant une base B.

Comme pour les grands nombres, il est maladroit de conserver tous les coefficients nuls d'un polynôme creux. Il sera alors plus efficace d'utiliser une liste dont les éléments sont des couples (exposant de X,coefficient non nul).

Une conséquence de cette similarité de représentation est que de nombreux algorithmes pourront s'appliquer aussi bien à des nombres qu'à des polynômes. Ceci est encore renforcé par les liens entre les structures algébriques des ensembles F et $F[X]$ (voir section 2.2).

Si $\geq$ est une relation d'ordre total sur $\mathcal{A}$, Il existe une relation d'ordre total sur $\mathcal{A}[\mathcal{X}]$: soient P et P' 2 polynômes de degrés d et d', on a que $P \geq P'$ si et seulement si :

$$d > d', \text{ ou bien : } d = d' \text{ et } a_d \geq a'_d.$$

Enfin, on peut étendre la définition des polynômes à des fonctions à plusieurs variables, on parlera alors de polynômes multivariés. On notera par exemple $\mathbb{R}[X_1, X_2, ..., X_n]$ l'ensemble des polynômes à n variables sur $\mathbb{R}$. Le stockage de tels polynômes peut se faire de manière récursive : on stocke la liste des coefficients associés à X_1. Un tel coefficient est un polynôme à $n-1$ variables, que l'on va donc stocker comme une liste associée à la variable X_2, et ainsi de suite.

2.2 Rappels d'algèbre : anneaux et corps

Définition 2 $(\mathcal{A}, +, \cdot)$ *est un anneau si et seulement si :*

– $(\mathcal{A}, +)$ *est un groupe commutatif d'élément neutre* 0 *(+ est appelée addition)*
– $\cdot$ *est associative, possède un élément neutre noté* 1 *et est distributive par rapport à +* ($\cdot$ *est appelé multiplication).*

Les anneaux généralement utilisés en calcul formel sont commutatifs ($\cdot$ est commutatif) et cette hypothèse est implicite dans la suite.

Un anneau est dit intègre quand : le produit de deux éléments est égal à 0 implique que l'un au moins des éléments est nul. Par exemple $(\mathbb{Z}, +, \cdot)$ est un anneau intègre. Mais l'ensemble des matrices $n \times n$ à coefficients dans $\mathbb{Z}$, muni de l'addition et de la multiplication classiques sur les matrices, est un anneau non intègre.

2.2.1 Divisibilité et Anneaux Factoriels

Un élément non nul est dit *inversible* (ou unité) s'il possède un inverse pour la multiplication. L'ensemble des éléments inversibles de $\mathcal{A}$, noté $U(\mathcal{A})$ est un groupe commutatif. On voit immédiatement qu'un corps est un anneau où tout élément non nul est inversible. Par contre les seuls inversibles de $\mathbb{Z}$ sont 1 et -1.

Deux éléments sont *associés* si et seulement si l'un est obtenu à partir du premier par multiplication par un inversible. L'association est une relation d'équivalence et on confondra généralement deux éléments de la même classe.

On dit que a *divise* b, noté $a|b$, si et seulement si il existe c tel que $b = a \cdot c$. Tout élément divise un inversible. Un élément non nul a, non inversible, qui n'admet aucun diviseur autre qu'un inversible et ses associés, est appelé *premier ou irréductible*. Cette définition sur un anneau $\mathcal{A}$ généralise bien sûr la notion de primarité classique sur $\mathbb{N}$ (on vérifiera que 1 n'est pas premier dans $\mathbb{N}$ ou dans $\mathbb{Z}$). Deux éléments sont premiers entre eux s'ils n'ont d'autre diviseur commun que des inversibles.

Un *anneau factoriel* est un anneau intègre dans lequel tout élément non nul peut s'écrire de manière unique (à une association près) comme le produit d'éléments premiers. Les anneaux factoriels ont des propriétés importantes concernant la divisibilité, en particulier l'existence d'un PGCD (voir chapitre 4).

2.2.2 Anneaux Euclidiens

Informellement, un anneau est euclidien quand il permet la division euclidienne. Les propriétés d'un anneau euclidien sont donc celles de $\mathbb{Z}$.

Définition 3 *Un anneau $\mathcal{A}$ est euclidien si et seulement si il existe une application v de $\mathcal{A}^*$ sur $\mathbb{N}$ vérifiant :*

1. $\forall a, b \in \mathcal{A}^*, a|b \Rightarrow v(a) \leq v(b)$ *(v est appelée une valuation de $\mathcal{A}$).*
2. $\forall (a, b) \in \mathcal{A} \times \mathcal{A}^*, \exists (q, r) \in \mathcal{A} \times \mathcal{A} / a = b \cdot q + r$ *(avec $r = 0$ ou $v(r) < v(b)$) (v valuation euclidienne).*

Si v est une valuation euclidienne, alors l'inégalité du premier point est stricte dès que b n'est pas associé à a. Il faut noter que (q, r) n'est pas unique en général.

On peut démontrer que tout anneau euclidien est factoriel. Pour cela on commence par montrer que tout élément non inversible et non irréductible possède un diviseur irréductible. Ensuite on construit grâce à une récurrence sur $v(x)$ une décomposition de x en éléments irréductibles, et enfin on mentre que cette décomposition est unique.

2.2.3 Exemples

Pour $\mathbb{Z}$, on peut prendre comme valuation $v(x) = |x|$. Alors $\mathbb{Z}$ est euclidien, mais les couples (q, r) ne sont pas uniques quand $r \neq 0$ (il existe un couple avec $r < 0$, et un couple avec $r > 0$).

L'ensemble des entiers de Gauss est l'ensemble $\{x + i \cdot y / x, y \in \mathbb{Z}\}$, muni de l'addition et de la multiplication des nombres complexes. Si l'on prend la valuation $v(z) = z\bar{z}$ (module de z) on obtient un anneau euclidien.

$\mathcal{A}[X]$, ensemble des polynômes à coefficient dans l'anneau $\mathcal{A}$, est un anneau factoriel dès que $\mathcal{A}$ lui-même est factoriel. Ce résultat qui reste vrai pour des polynômes à plusieurs variables, permet d'envisager la factorisation de ces polynômes. Mais $\mathcal{A}[X]$ est un anneau euclidien si et seulement si $\mathcal{A}$ est un corps (quand on choisit comme valuation $v(P) = deg(P)$). On peut dans ce cas introduire la division euclidienne entre polynômes. Celle-ci a alors la propriété d'unicité : le couple (Q, R) obtenu est unique.

Chapitre 3

Arithmétique

On s'intéresse ici à la manipulation des polynômes sur un anneau $\mathcal{A}$ (souvent un corps), en général $\mathbb{R}$ (les algorithmes et leurs complexités sont identiques sur $\mathbb{C}$), et des grands nombres. On définira les polynômes P et Q de $\mathcal{A}[X]$ par $\sum_{i=0}^{d} a_i X^i$ et $\sum_{j=0}^{d'} a_j X^j$. Un grand nombre est supposé écrit en base B, où B est tel qu'une opération arithmétique sur des nombres plus petits que B s'effectue en un temps constant.

3.1 Opérations arithmétiques de base, algorithmes naïfs

3.1.1 Evaluation de polynômes monovalués

Première opération sur les polynômes : leur évaluation en fonction d'une valeur affectée à la variable. On suppose ici que les coefficients du polynôme sont de taille maximale fixée. En conséquence, les opérations arithmétiques de base s'effectueront en un temps constant.

Algorithme 1 *Entrée : P, val*
pour $i = 0$ *jusqu'à* d *faire*
    $sum2 \leftarrow a_i$
    pour $j = 1$ *jusqu'à* i *faire*
        $sum2 \leftarrow sum2 \times val$
    $sum \leftarrow sum + sum2$

Complexité : cet algorithme naïf effectue d additions et $\frac{(d+1)(d+2)}{2}$ multiplications.

3.1.2 Addition

Addition de 2 polynômes P et Q : il suffit ici d'ajouter 2 à 2 les coefficients des deux polynômes, ce qui se fait en exactement d additions simples, si d est le degré maximum des polynômes.

Addition de 2 nombres A et B écrits en base B : Le principe est le même, il faut simplement y ajouter un mécanisme de propagation de la retenue. Si k est le nombre maximum de chiffres en base B des opérandes, on a donc encore k additions (de 3 nombres de taille inférieure à B). On peut aussi dire que le nombre d'addition est de l'ordre de $\log_B A$ (en supposant $A \geq B$).

3.1.3 Multiplication

Multiplication de 2 polynômes : Si on suppose $d \geq d'$, on a naturellement

$$P \times Q(X) = \sum_{j=0}^{d'} \sum_{i=0}^{d} a_i b_j X^{i+j}.$$

Le calcul, basé sur la distributivité de la multiplication sur l'addition, s'effectue donc en $d \times d'$ multiplications et $d + d'$ additions.

La multiplication de 2 grands nombres en base B s'effectue de la même façon, à ceci près que les additions là encore doivent tenir compte de la propagation des retenues. Il faut aussi prendre garde que la retenue est obtenue comme le quotient d'une division par B. Cependant on montre que ce quotient est inférieur à B. On doit donc effectuer des divisions par B de nombres s'écrivant avec au plus 2 chiffres en base B. Ces divisions seront considérées comme des opérations élémentaires. Le nombre global d'opérations élémentaires reste alors de l'ordre de $d \times d'$.

3.1.4 Division euclidienne

On suppose ici que la division est unique sur $\mathcal{A}[X]$: Quels que soient P et P' dans $\mathcal{A}[X]$, il existe 2 polynômes uniques Q et R tels que :

$$P(X) = Q(X) \times P'(X) + R(X), \text{ où } R \text{ vérifie : } deg(R) < deg(P')$$

Ceci est vrai si $\mathcal{A}$ est un corps, et reste vrai si $\mathcal{A}$ est un anneau factoriel et P' est unitaire.

L'algorithme de la division suit le schéma de la division entière apprise en primaire. On adapte simplement cet algorithme dans le cas de chiffres en base B. Il faut également faire attention, à chaque étape, au calcul du bon chiffre du dénominateur. En ce qui concerne les polynômes, on applique exactement le même principe.

L'algorithme est plus simple à appliquer si le polynôme diviseur est unitaire. En particulier dans ce cas les coefficients restent entiers. C'est pourquoi on peut diviser dans, par exemple, $\mathbb{Z}[X]$ par un polynôme unitaire.

De manière un peu surprenante, cet algorithme de division (s'il est écrit soigneusement) n'effectue par plus d'opérations élémentaires que celui de la multiplication naïve : de l'ordre de $d \times d'$ additions, multiplications et divisions élémentaires.

3.2 Algorithmes plus performants

3.2.1 Horner pour l'évaluation

L'algorithme de Horner est bien connu en informatique, car il permet de calculer rapidement un nombre à partir de la suite de ses chiffres, même si ceux-ci sont donnés un par un et non stockés en mémoire. Il est ici adapté à l'évaluation d'un polynôme. On a en effet :

$$P(X) = a_0 + X(a_1 + X(a_2 + X(.... + X(a_{n-1} + Xa_n)...)$$

La transcription directe de cette formule nous donne l'algorithme d'évaluation de polynôme de Horner. Cet algorithme effectue d additions mais seulement d multiplications, ce qui est bien meilleur que l'algorithme naïf.

3.2.2 Karatsuba pour la multiplication

L'algorithme de Karatsuba est décrit pour la multiplication des polynômes, mais il s'applique aussi, naturellement, à la multiplication d'entiers.

C'est un algorithme de type *diviser pour règner*. On commence par écrire chaque opérande à l'aide de deux polynômes deux fois plus petits (en degrés). Le produit est ensuite écrit uniquement à l'aide de produits de ces polynômes plus petits. Ce principe est bien sûr réitéré pour construire un algorithme récursif dont on montre que le nombre de multiplication dans $\mathcal{A}$ est largement inférieur à celui de la première méthode.

Soient 2 polynômes P et Q de degré $2d$. On peut écrire $P = P_b + X^d \times P_h$, et $Q = Q_b + X^d \times Q_h$. Il vient alors :

$$P \times Q = P_b \times Q_b + (P_b \times Q_h + P_h \times Q_b) \times X^d + P_h \times Q_h \times X^{2d}.$$

Le produit de 2 polynômes de degré $2d$ est donc effectué grâce à 4 produits de 2 polynômes de degré d. Mais on peut encore faire mieux en posant $R_1 = P_b Q_b$, $R_2 = P_h Q_h$ et $R_3 = (P_b + P_h)(Q_b + Q_h)$. En effet on écrit alors le produit sous la forme :

$$P \times Q = R_1 + (R_3 - R_1 - R_2) \times X^d + R_2 \times X^{2d}$$

et trois multiplications suffisent.

L'algorithme récursif utilisant la formule ci-dessous effectue un nombre de multiplications dans $\mathcal{A}$ $nbmut(d) \leq 3^{k+2}$, où d est le degré des opérandes, et $k = \lfloor \log_2 d \rfloor$. On en déduit que la complexité de l'algorithme de Karatsuba est $\mathcal{O}(d^{\log_2 3})$ ($\log_2 3 \approx 1,585$).

Chapitre 4

Division euclidienne et calcul modulaire

Dans ce chapitre, on travaille sur un anneau factoriel A ; la plupart du temps, A sera euclidien (voir le chapitre 2).

4.1 Algorithme d'Euclide et Extensions

4.1.1 PGCD

Définition 4 *le pgcd de 2 éléments $a, b \in A$ est l'élément $d \in A$ divisant a et b et tel que tout diviseur de a et de b est également diviseur de d. d est unique (à une association près). On note $d = pgcd(a, b) = a \wedge b$.*

L'algorithme d'Euclide permet de calculer le *pgcd* de deux éléments d'un anneau Euclidien. Cette hypothèse est fondamentale : si l'anneau n'est pas euclidien, on n'a pas de division euclidienne !
Principe : soient a, b dans A^*
on définit la suite :
$r_0 := a$; $r_1 := b$; r_i est le reste de la division euclidienne de r_{i-1} par r_{i-2}
cette suite est finie et il existe $n \in I\!N$ tel que $r_n = a \wedge b$ et $r_{n+1} = 0$.

Le *pgcd* n'est pas unique en général. Mais si $\mathcal{A}$ est un corps, alors le *pgcd* est unique.

4.1.2 Relation de Bezout

Théorème 1 *Soient a et b 2 éléments de $\mathcal{A}$, alors il existe deux éléments u, v de $\mathcal{A}$ tels que $au + bv = a \wedge b$.*

L'algorithme d'Euclide étendu permet de calculer u et v. Mais en général ces deux nombres ne sont pas uniques.

Principe de l'algorithme d'Euclide étendu : soient a, b dans $\mathcal{A}^*$
on définit 4 suites q, r, u, v dans $\mathcal{A}$:

$$\begin{array}{lll} r_0 := a; & r_1 := b; & r_i = r_{i-2} - q_i \cdot r_{i-1} (\text{div. euclidienne } r_{i-2}/r_{i-1}) \\ u_0 = 1; & u_1 := 0; & u_i = u_{i-2} - q_i \cdot u_{i-1} \\ v_0 = 0; & v_1 := 1; & v_i = v_{i-2} - q_i \cdot v_{i-1} \end{array}$$

On reconnait la suite (r_i) de l'algorithme d'euclide simple. On sait donc qu'il existe n tel que $r_n = a \wedge b$ et $r_{n+1} = 0$. On peut montrer par récurrence : $\forall i \leq n, r_i = a \cdot u_i + b \cdot v_i$, ce qui au rang n nous donne le résultat cherché. A chaque étape, le calcul des u_i et v_i se fait très simplement (dès lors que q_i a déjà été calculé en même temps que r_i).

On peut bien sûr étendre les définitions de *pgcd* et de la relation de Bezout à un nombre quelconque d'éléments de A.

4.1.3 Complexité Dans $\mathbb{Z}$

Théorème 2 *L'algorithme d'Euclide dans $\mathbb{Z}$ a une complexité $\mathcal{O}(\log a \times \log b)$.*

Ce résultat est prouvé en 2 temps. On majore d'abord le nombre d'étapes n de l'algorithme : il est borné par $2\lfloor \log_2 a \rfloor + 1$. D'autre part, à chaque étape la complexité de la division de x par y est $\mathcal{O}(\log y \times \log q)$, où q est le quotient. On peut vérifier que $\prod_1^n q_i \leq a$, ce qui donne le résultat.

4.1.4 recherche du *pgcd* dans un anneau de polynômes $\mathcal{A}[X]$

Si $\mathcal{A}$ est un corps, tout va bien : $\mathcal{A}[X]$ est un anneau euclidien, et les 2 algorithmes ci-dessus s'appliquent. Leur complexité dépend de la complexité des opérations arithmétiques dans $\mathcal{A}$.

Mais si $\mathcal{A}$ n'est pas un corps ? Il n'existe pas dans $\mathcal{A}[X]$ de division euclidienne ! Ainsi, on si l'on veut diviser un polynôme à coefficients entiers par un autre, on risque d'obtenir un polynôme dont les coefficients seront fractionnaires. Pourtant, le *pgcd* existe. Comment le calculer ?

Première méthode : On travaille dans un corps contenant $\mathcal{A}$. Ainsi, si l'on recherche le *pgcd* de deux polynômes de $\mathbb{Z}[X]$, on peut utiliser l'algorithme d'Euclide sur $\mathbb{Q}[X]$. Le résultat sera un polynôme à coefficients entiers (à un facteur multiplicatif rationnel près). Mais il n'est pas très agréable (ni efficace) d'effectuer les divisions euclidiennes successives dans $\mathbb{Q}[X]$ (les coefficients ont tendance à devenir des nombres fractionnels de grande taille).

Deuxième méthode : On introduit une *pseudo-division euclidienne* dans $\mathbb{Z}[X]$. On voit facilement que le problème vient du coefficient de plus haut degré du dénominateur. Notons $cd(P)$ ce coefficient pour un polynôme quelconque P. La pseudo-division est définie ainsi :

Définition 5 *Soient P et P' 2 polynômes de $\mathcal{A}[X]$ où $\mathcal{A}$ est un anneau intègre, avec $deg(P) \leq deg(P')$. Alors il existe Q et R uniques, $R = 0$ ou $deg(R) < deg(P')$, tels que :*

$$cd(P')^{deg(P)-deg(P')+1}P = P' \times Q + R$$

On peut maintenant montrer le résultat suivant : $P \wedge P' = P' \wedge R$, où $deg(P) \leq deg(P')$ et R est le pseudo reste de la division de P par P'. On effectue donc une suite de pseudo-divisions pour obtenir le $PGCD$, ce qui ne coûte pas plus cher que des divisions euclidiennes classiques. Malheureusement, les coefficients des pseudo-restes, même s'ils sont dans $\mathcal{A}$, ont tendance à rapidement grossir. On peut pallier à ce fait.

Soit $c(P)$ le $PGCD$ des coefficients de P. On l'appelle le contenu de P. Un polynôme de contenu 1 est dit **primitif**, et le polynôme obtenu en divisant P par son contenu est appelé sa partie primitive, notée $pp(P)$. On montre comme pour le résultat ci dessus que : $PGCD(P, P') = PGCD(P', pp(R))$, où P etP' sont primitifs avec $deg(P) \leq deg(P')$ et R est le pseudo reste de la division de P par P'. Comment se ramener à des polynômes primitifs ?

Théorème 3 *On a $c(P \wedge P') = c(P) \wedge c(P')$, et $pp(P \wedge P') = pp(P) \wedge pp(P')$.*

On peut donc calculer séparément les contenu et partie primitive de $P \wedge P'$; ce qui nous donne un algorithme en pratique beaucoup plus efficace que la première méthode.

4.2 calcul modulaire

Dans cette section nous nous plaçons dans $\mathbb{Z}$ (même si des extensions sont possibles à des anneaux de polynômes). Nous avons donc des algorithmes pour calculer le reste d'une division euclidienne. Ces algorithmes sont d'autant plus efficaces que le dénominateur est petit par rapport au numérateur. Que peut-on faire avec cela, notamment pour le *pgcd* ?

On écrit $a \equiv b[p]$ (a est congruent à b modulo p) pour dire que a et b ont le même reste par p. On remarque que $\equiv$ est conservée pour l'addition, la soustraction, la multiplication (mais pas par la division euclidienne !) :

$$a \equiv b[p] \Rightarrow a * c \equiv b * c[p], \forall c \in \mathbb{Z}, * = +, -, .$$

Attention aux simplifications excessives : $0\dot{2} \equiv 2\dot{2}[4]$, mais $0 \not\equiv 2[4]$! Par contre, une conséquence importante est la suivante : considérons une fonction $f(x_1, x_2, \ldots, x_k)$. Si f est calculable à partir des opérations élémentaires ci-dessus, alors

$$f(x_1, x_2, \ldots, x_k) \equiv \tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_k),$$

où $\tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_k)$ est le résultat du calcul de f dans $\mathbb{Z}/p\mathbb{Z}$. On obtient donc le reste du résultat cherché modulo p. Comment reconstruire f revient à résoudre un système d'équations modulaires.

Rappelons enfin que si p est un entier, $\mathbb{Z}/p\mathbb{Z}$ est un anneau pour l'addition et la multiplication. Mais les éléments inversibles sont tous les éléments de $\mathbb{Z}/p\mathbb{Z}$ premiers avec p. On en déduit que $\mathbb{Z}/p\mathbb{Z}$ est un corps si et seulement si p est premier. Il est donc aisé de calculer dans cet ensemble, de taille réduite si p est un nombre premier "pas trop grand". C'est l'objectif du calcul modulaire, dont la méthodologie est présentée à la fin de cette section.

4.2.1 Systèmes d'équations modulaires

Une façon de coder un nombre est de ne conserver que le reste de ses divisions par k nombres plus petits. Mais ceci soulève quelques problèmes : que doit valoir k ? Comment choisir ces nombres pour minimiser k ? Inversement, peut-on calculer facilement un nombre dont on connaît les restes (problème de reconstruction) ?

Cas de deux équations

Problème posé ; trouver un entier x tel que :

$$\begin{aligned} x &\equiv a1[m1] \\ x &\equiv a2[m2] \end{aligned}$$

.

Théorème 4 *Soient $m1$ et $m2$ premiers entre eux. Soient u et v solutions de la relation de Bezout pour $m1$ et $m2$. Alors $x = a2 \cdot u \cdot m1 + a1 \cdot v \cdot m2$ est solution du système et toute solution est congruente à x modulo $m1m2$.*

Cas général

On considère n équations modulaires, avec n nombres premiers entre eux : $m_1, m_2, ..., m_n$. Algorithme récursif : on résoud pour les 2 premières. On obtient un nombre α. On résoud ensuite :

$$\begin{aligned} x &\equiv \alpha[m1 \cdot m_2] \\ x &\equiv a3[m_3] \end{aligned}$$

et ainsi de suite jusqu'à obtenir un nombre de $[0, m_1 \cdot \ldots m_n]$.

Conséquence : si les m_i sont premiers entre eux, on a unicité de l'écriture d'un nombre de $[0, m_1 \cdot \ldots m_n]$ sous forme de ses restes. C'est le fameux *théorème du reste chinois*, connu depuis l'antiquité.

Inconvénient de l'algo : explosion de la taille de x (il faut ensuite refaire une division par $m_1 \cdot \ldots m_n$ pour obtenir le nombre cherché).

Base mixte

On peut écrire tout nombre $x \in [0, m_1 \cdot \cdots m_n]$ comme $x = b_1 + b_2 m_1 + b_3 m_1 \cdot m_2 + \cdot + b_n \cdot m_1 \cdot \ldots m_{n-1}$, avec $b_i \in [0, m_i[$ (et on a bijection).

L'algorithme de Garner consiste à calculer les b_i pour un x solution du système de n équations modulaires. On remarque d'abord que $a_1 = b_1$. Ensuite supposons $b_1, b_2, \ldots, b_{k-1}$ connus, $k \leq n$.
On a $a_k \equiv (b_1 + \cdots + b_k \cdot m_1 \cdot \cdot m_{k-1})[m_k]$. Or $m_1 \cdot \cdot m_{k-1}$ est premier avec m_k et donc inversible dans $\mathbb{Z}/m_k \cdot \mathbb{Z}$. D'où :

$$b_k \equiv (m_1 \cdot \cdot m_{k-1})^{-1} \cdot (a_k - b_1 - b_2 \cdot m_1 - \ldots - b_{k-1} \cdot m_1 \cdot \cdot m_{k-2})[m_k].$$

L'avantage de cet algorithme est l'absence d'explosion combinatoire. Les seuls calculs effectués par l'algorithme sont les calculs d'inverses par l'algorithme d'Euclide étendu, d'où une complexité totale $\mathcal{O}(n \cdot \log(\Pi m_i))$

4.2.2 méthode générale de calcul modulaire

Soit un problème $\mathcal{P}$ à résoudre dans un anneau euclidien $\mathcal{A}$. L'idée est de résoudre $\mathcal{P}$ modulo un ou plusieurs entiers. Une solution dans $\mathcal{A}$ est ensuite reconstruite. 3 variantes sont utilisées en pratique. Dans la première, un seul entier (qui est supposé premier) est choisi. Mais la reconstruction d'une solution de $\mathcal{P}$ n'est pas toujours possible . Un cas simple est le calcul d'un déterminant. Dans la seconde, on choisit n nombres premiers. Si $\mathcal{A} = \mathbb{Z}$, on résoud donc $\mathcal{P}$ dans n corps de petite dimension. Le choix des moduli est le point crucial. Une condition nécessaire est que le résultat soit majoré par le produit des moduli (de manière à garantir l'unicité). Un exemple est le calcul du *pgcd*dans $\mathbb{Z}[X]$. La troisième méthode considère comme moduli les puissances successives d'un nombre premier p.

L'efficacité du calcul modulaire (en particulier dans les 2 dernières variantes) tient aux avantages suivants :

1. il évite l'accroissement de taille des données lors des calculs intermédiaires
2. Si les moduli sont bien choisis (assez petits) les données tiennent dans les registres de la machine cible, ce qui permet des calculs très rapides pour les opérations de base.
3. L'algorithme initial de résolution de $\mathcal{P}$ est en général de complexité au moins linéaire. La décomposition en n problèmes plus petit accélère alors la résolution, sauf si la phase de reconstruction est trop longue (voir la complexité de l'algorithme de Garner).
4. la résolution est grandement accélérée si le calcul est distribué sur un système informatique parallèle (cluster de stations, grille, machine massivement parallèle). En effet les calculs suivant les différents moduli sont indépendants.

Chapitre 5

Calculs sur les polynômes : Factorisation, Interpolation

Dans ce chapitre, on travaille sur un anneau de polynômes à coefficients dans $\mathcal{K}$ (corps) ou $\mathcal{A}$ (anneau euclidien). Après avoir abordé l'interpolation, nous nous concentrons sur la factorisation dans $\mathcal{K}[X]$.

Définition 6 *La factorisation de $P \in \mathcal{K}[X]$ est l'écriture de P sous la forme :*

$$P = cd(P) f_1^{e_1} \cdots f_r^{e_r},$$

où les f_i, irréductibles et unitaires, sont appelés les facteurs irréductibles (ou premiers) de P.

5.1 Algorithme d'interpolation

Rappelons le problème : soient les valeurs dans $\mathcal{K}$ $\alpha_1, \ldots, \alpha_n$ et $\beta_1, \ldots, \beta_n$. Trouver un polynôme P de degré strictement inférieur à n tel que $P(\alpha_i) = \beta_i$, $i = 1, \ldots, n$. Il existe des algorithmes très rapides basés sur l'analyse numérique. La méthode proposée donne les coefficients exacts de P et se généralise à d'autres interpolations.

Remarquons que la donnée des $P(\alpha_i)$ n'est rien d'autre qu'une nouvelle manière de représenter P, puisque l'on sait que P est alors unique. Le problème d'interpolation est donc celui de la réécriture de P dans un autre système (base x), tout comme un entier de taille bornée pouvait être défini par ses restes modulo un ensemble de nombres premiers. Ainsi, une extension de l'algorithme de Garner résoud le problème !

Définition 7 *Les interpolants de Lagrange d'un polynôme P sont les $l_i = \prod_j \frac{x-\alpha_j}{\alpha_i-\alpha_j}$. On notera $m_i = x - \alpha_i$.*

Utilisant le fait que $l_i(\alpha_j)$ est nul si $j \neq i$, on peut écrire

$$P = \sum_i \beta_i \prod_{j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}.$$

En conséquence, P est la somme de polynômes qui sont tous divisibles par m_i,$i = 1 \ldots, n$, sauf $\beta_i \cdot l_i$, pour lequel le reste de la division est une constante de $\mathcal{K}$ qui ne peut être que β_i. Dès lors on peut écrire que P est l'unique solution du système :

$$\begin{array}{rcl} P & \equiv & \beta_1[m_1] \\ \vdots & & \vdots \\ P & \equiv & \beta_n[m_n] \end{array}$$

Les m_i sont premiers entre eux, et la généralisation de Garner permet d'obtenir P en $\mathcal{O}(n^2)$ opérations dans $\mathcal{K}$.

Note : la démonstration est similaire. On utilise simplement comme valuation le degré des polynômes. Toute autre solution que P est congrue à P modulo le produit des m_i, et a donc forcément un degré supérieur à n.

5.2 Algorithme de factorisation par interpolation

Cett méthode historique de factorisation dans $\mathbb{Z}[X]$, dûe à Newton et Kronecker, est restée longtemps la seule pour factoriser à la main des polynômes. Elle est basée sur les idées suivantes : si P est le produit de 2 polynômes Q et U, ses facteurs premiers sont l'union des facteurs de ses diviseurs. D'autre part, étant donné un nombre α, $Q(\alpha)$ divise $P(\alpha)$ dans $\mathbb{Z}$.

Soient donc n nombres distincts $\alpha_1, \ldots, \alpha_n$, avec $n \leq d/2$ car on sait que P a un diviseur de degré $n-1$ au plus. On recherche alors $\beta_1, \ldots, \beta_n$ tels que β_i divise $P(\alpha_i)$, $i = 1, \ldots, n$. Ensuite on recherche le polynôme d'interpolation Q associé. S'il est à coefficients dans $\mathbb{Z}$, on a trouvé un candidat, et s'il divise P c'est un facteur irréductible.

L'interpolation étant une opération assez rapide (même au $XVIII^e$ siècle), cette méthode serait efficace si Q avait beaucoup de chances d'être dans $\mathbb{Z}$, ce qui n'est malheureusement pas le cas. De plus, le nombre de n-uplets candidats peut être très élevé. Cette méthode n'est donc plus guère utilisée aujourd'hui.

5.3 Algorithme de factorisation dans $\mathbb{Z}_p[X]$

Dans cette section, nous présentons la méthode en trois phases pour la factorisation dans $\mathbb{Z}_p[X]$. Nous introduisons ensuite brièvement l'algorithme

d'élévation de Hensel (Lifting algorithm) qui permet par un passage aux puissances de p de reconstruire une solution dans $\mathbb{Z}[X]$. La méthode s'applique plus généralement à $K_p[X]$, où K_p est un corps fini à p éléments.

Principe de la méthode en 3 phases : On écarte d'abord les facteurs multiples (appelés facteurs carrés). On effectue ensuite une partition des facteurs en groupant ensemble les facteurs de même degré. Enfin, on résoud les sous-problèmes ainsi obtenus en calculant les facteurs de même degré.

5.3.1 Etape 1 : élimination des facteurs multiples

En théorie, cette étape est nécessaire avant la phase de partition. En pratique, la seconde phase peut se passer de ce prétraitement (voir ci-dessous). Il est donc plus simple d'attendre que tous les facteurs aient été calculés. Il suffit ensuite de vérifier leur multiplicité par une suite de division P/f (on continue tant que la division par le facteur f est sans reste).

Une alternative consiste bien sûr à se débarrasser de ces multiplicités immédiatement, ce qui diminue la taille des entrées des algorithmes des étapes 2 et 3. Il se trouve que cela est possible (voir références) simplement en divisant P par le *pgcd* de P et de sa dérivée P' ! Mais le gain pratique de ce prétraitement ne semble pas très important.

5.3.2 Etape 2 : factorisation partielle avec degrés distincts

Cette étape est celle qui demande, de loin, le plus de temps de calcul. Elle prend en entrée le polynôme P et sépare les facteurs suivant leurs degrés, c'est-à-dire qu'elle calcule $g_1, g_2, \ldots, g_s$, où g_i est le produit des facteurs de degré i. La méthode proposée utilise le résultat suivant :

Proposition 1 *Quel que soit $d \geq 1$, $x^{q^d} - x \in \mathcal{K}_q[x]$ est le produit de tous les polynômes irréductibles unitaires de $\mathcal{K}_q[x]$ dont le degré divise d.*

Quand $d = 1$, cette proposition est le petit théorème de Fermat. Un facteur irréductible de P est donc aussi un diviseur de $x^{q^d} - x$. Et si l'on calcule le *pgcd* de ces deux polynômes, on obtient le produit des facteurs de f dont le degré divise d.

On appelle décomposition en degrés distincts de $P \in \mathcal{K}_q[x]$ la séquence $g_1, g_2, \ldots, g_s$, où g_i est le produit de tous les polynômes irréductibles unitaires de $\mathcal{K}_q[x]$ divisant P.

Algorithme 2 *(factorisation en degrés distincts)*
Entrée : $P \in \mathcal{K}_q[x]$, de degré $n > 0$, P unitaire et sans carré.
init : $h_0 = x$, $P_0 = P$, $i = 0$.
tant que $P_i \neq 1$ faire
$\quad i := i + 1$
$\quad h_i := h_{i-1}^q \ mod\ P$
$\quad g_i :=$*pgcd*$(h_i - x, P_{i-1}), \quad f_i := \frac{f_{i-1}}{g_i}$
Retourner $(g_1, \ldots, g_i)$

On peut vérifier qu'après la première boucle, g_1 contient bien, d'après la proposition ci-dessus, les irréductibles unitaires de degré 1 divisant P. On les retire ensuite de P. On a de même g_2 et g_3, produits des facteurs de degrés respectifs 2 et 3. Pour g_4, on a les facteurs dont le degré divise 4. Mais ceux de degré 2 ont déjà été retirés de P (par hypothèse, P est sans carré). g_4 est donc bien le produit des facteurs irréductibles de degré 4. Et ainsi de suite... La complexité de l'algorithme est $\mathcal{O}(s \cdot M(n) \cdot \log(nq))$, où $M(n)$ est la complexité d'une opération élémentaire dans $\mathcal{K}$. Cet algorithme est donc polynômial en n et $\log(q)$.

5.3.3 Etape 3 : factorisation avec degrés égaux des facteurs

Soit q un nombre premier, ou bien la puissance impaire d'un nombre premier (l'algorithme proposé est correct pour cette hypothèse, mais peut être modifié si elle n'est pas vérifiée).
L'algorithme présenté est dû à Cantor et Zassenhauss. Il considère en entrée un polynôme P de $\mathcal{K}_q$ de degré n, sans carré, dont tous les facteurs sont de même degré d. P compte donc r facteurs irréductibles f_i avec $r \cdot d = n$. C'est un algorithme probabiliste : il teste un polynôme a généré aléatoirement. Si a n'est pas premier avec P, on dit que c'est un polynôme *séparateur*. Dans ce cas, le *pgcd* de P et de a est un diviseur propre de P. Sinon on construit un nouveau candidat à partir de a, en utilisant la proposition ci-dessous. S'il n'est pas non plus un séparateur, on recommence.

L'algorithme suivant retourne un facteur irréductible de P (ou un diviseur propre de P) ou ECHEC.

Algorithme 3 *(Cantor-Zassenhauss)*
Entrée : $P \in \mathcal{K}_q[x]$, de degré $n > 0$, P unitaire et sans carré.
init : choix aléatoire d'un polyn. $a \in \mathcal{K}_q[x]$, $0 < deg(a) < n$
$g :=$*pgcd*(a, P)
si $g \neq 1$: retourner g.
$b := a^{\frac{q^d-1}{2}} \mod P$
$g :=$*pgcd*$(b-1, P)$
si $g \neq 1$ et $g \neq P$ alors retourner g sinon retourner ECHEC

Cet algorithme dont le principe est très simple est efficace si la probabilité que $b-1$ soit un séparateur est suffisamment élevée. Toute l'astuce (assez technique) consiste à le construire pour cela. On s'appuie sur la proposition suivante :

Proposition 2 *Soit S l'ensemble des carrés non nuls de $\mathcal{K}_q$. Alors S est de cardinalité $\frac{q-1}{2}$, et $a \in S \Rightarrow a^{\frac{q-1}{2}} = 1$, sinon $a^{\frac{q-1}{2}} = -1$.*

Considérons un polynôme quelconque α, et le reste de sa division par P $\chi = \alpha \mod P$. Par une généralisation du théorème du reste chinois, on montre qu'il existe un isomorphisme entre l'ensemble des polynômes χ et le sous - ensemble $R_1 \times \cdots \times R_r$ de K_q^r contenant les vecteurs $\chi(\alpha)$, avec $\chi_i(\alpha) = \alpha$

mod $f_i, i = 1, \ldots, r$. $\chi_i(\alpha) = 0$ si et seulement si α est multiple de f_i. Les R_i sont des corps à q^d éléments que l'on peut "identifier" avec K_{q^d}. SI l'on applique la proposition 2 à R_i (avec q remplacé par q^d), on voit que si $\beta \in R_i$, et $e = \frac{q^d-1}{2}$, alors β^e vaut 1 ou -1 avec la même probabilité (car a tel que $\beta = a \mod f_i$ a été choisi au hasard). On a donc une chance sur deux pour que $\chi_i(a^e) - 1 = \chi_i(a^e - 1) = 0$, donc une chance sur 2^{r-1} pour que les χ_i soient, soit tous nuls (on a P entier), soit tous non nuls (le *pgcd* est 1). Sinon, $\chi(a^e - 1)$ est un séparateur.

Même si $r = 2$, on a encore une chance sur deux de tomber juste dès le premier essai. La probabilité de trouver effectivement un diviseur propre croît très rapidement avec le nombre de tirages aléatoires. ! La complexité d'une exécution de l'algorithme est donnée par celle du calcul du *pgcd*, soit $\mathcal{O}(M(n) \cdot \log(n))$ où $M(n)$ est la complexité d'une opération élémentaire dans K_q, et celle du calcul de b, qui est de $\mathcal{O}(d \cdot \log(q))$ opérations. Pour obtenir tous les facteurs de P, on exécute récursivement l'algorithme sur les diviseurs propres, d'où une complexité totale $\mathcal{O}(M(n) \cdot (d \log(q) + \log(n)) \cdot \log(r))$, polynômiale en n, en $\log(q)$ et en $\log(d)$.

Considérons enfin le problème complet de factorisation. On commence par diviser P par son coefficient directeur pour le rendre unitaire. On peut ensuite appliquer l'algorithme de l'étape 2. A chaque fois qu'un g_i est calculé, l'algorithme de l'étape 3 est appelé pour le décomposer en les facteurs irréductibles de degré i. Puis on détermine la multiplicité des facteurs trouvés par division. On divise ensuite le polynôme courant par les facteurs de degré i trouvés (en tenant compte de leur multiplicité) et on recommence. La seule chose à vérifier, c'est que g_i est exactement égal au produit des facteurs *distincts* de degré i, donc sans carré. Mais g_i est un diviseur de $x^{q^i} - x$, qui est le produit de tous les irréductibles unitaires de degré i distincts, donc sans carré. g_i est donc lui-même unitaire et sans carré, ce qui permet d'appliquer l'algorithme de Cantor et Zassenhauss. On a bien un algorithme polynômial pour la factorisation dans $\mathcal{K}_q[x]$, et en particulier dans $relatifrelatif_q[x]$.

5.3.4 Principe de l'algorithme d'élévation de Hensel

Cet algorithme n'est pas étudié en cours. Il est cependant utilisé dans l'algorithme de factorisation basé sur l'algorithme LLL.

Parmi les méthodes de calcul modulo, deux peuvent être utilisées pour la factorisation dans $\mathbb{Z}[x]$: utilisation directe d'un "grand" nombre premier p et travail dans $\mathbb{Z}_p$, ou utilisation d'un premier plus petit p ET d'un entier l tel que p^l soit assez grand. L'algorithme d'Hensel (Hensel lifting) appartient à la seconde catégorie et est plus efficace en pratique. Une difficulté importante ici est qu'un polynôme irréductible pour un corps fini comme $\mathbb{Z}_p$ n'est pas forcément fini dans $\mathbb{Z}[x]$ (ou pour un autre corps fini).

Le principe de l'algorithme est le suivant : supposons que P s'écrive comme le produit de deux polynômes dans $\mathbb{Z}_p[x]$, f et g, g unitaire. On peut (sous certaines conditions) construire deux nouveaux polynômes $\bar{f}$ et $\bar{g}$ à l'aide de

l'algorithme d'Euclide étendu pour f et g dans $\mathbb{Z}_p[x]$, tels que P s'écrive comme le produit de $\bar{f}$ et $\bar{g}$ dans $\mathbb{Z}_{p^2}[x]$. Il suffit ensuite de recommencer jusqu'à une puissance de p "assez grande". Le mode de calcul de $\bar{f}$ et $\bar{g}$ n'est pas détaillé dans ce résumé.

5.3.5 Utilisation pour la factorisation dans $\mathbb{Z}[X]$

Si P est primitif à coefficients dans $\mathbb{Z}$, factoriser P dans $\mathbb{Z}[x]$ ou dans $\mathbb{Q}[x]$, c'est la même chose (par propriété des contenus, les facteurs irréductibles seront également primitifs). Par contre, si P n'est pas primitif, sa factorisation nécessite celle de son contenu. Notons que dans $\mathbb{Z}[x]$, la factorisation de P primitif est unique, à une multiplication des facteurs par -1 près. Ce n'est plus le cas si P n'est pas primitif.

L'algorithme de Hensel peut être directement utilisé. Mais il n'est pas polynomial dans le cas général. Une méthode moins directe, mais plus rapide en pratique, est présentée dans la section suivante.

5.4 Aperçu de géométrie algorithmique : algorithme LLL pour la factorisation dans $\mathbb{Z}[X]$

5.4.1 Principe de l'algorithme de factorisation

On dispose de plusieurs algorithmes efficaces pour la factorisation dans $\mathbb{Z}_m[X]$, où m est premier ou une puissance d'un premier. Soit u un des facteurs irréductibles trouvés. Si u est un facteur dans $\mathbb{Z}[X]$, tant mieux. Sinon, on se place dans l'ensemble $\mathcal{E}$ des polynômes multiples de u modulo m (et de degré borné). L'idée est de chercher dans cet ensemble LE facteur irréductible g de f multiple de u modulo m. Ensuite on réitère le processus pour les trouver tous. Comme on peut s'y attendre, la complexité globale de l'algorithme sera assez élevée. Mais cette complexité reste polynomiale en n. La clé est l'algorithme de recherche dans $\mathcal{E}$.

Il se trouve que $\mathcal{E}$ est un ensemble très particulier. Rappelons d'abord qu'un polynôme de $\mathbb{Z}[X]$ de degré inférieur à d est évidemment identifiable à un vecteur de $\mathbb{Z}^{d+1}$. Plus généralement dans $\mathbb{R}^n$, on peut définir un treillis (ou un $\mathbb{Z}$-module) L comme étant l'ensemble des vecteurs engendrés par une famille de vecteurs $(f_1, \ldots, f_n)$ de $\mathbb{R}^n$:

$$L = \{x = \sigma_i r_i \cdot f_i, r_i \in \mathbb{R}, i = 1, \ldots, n\}$$

Si les f_i sont entiers, le treillis est donc une partie de $\mathbb{Z}^n$. Or on peut choisir les f_i à partir de u et de m pour que le treillis contienne exactement les multiples de u modulo m, de degré inférieur ou égal à un k fixé : $L = \mathcal{E}$. Comment trouver g dans L ? On se sert du résultat (admis) suivant :

Théorème 5 *Soient f et g dans $\mathbb{Z}[X]$ tels que :*

1. $deg(f) = n$, $deg(g) = k$
2. $\exists u \in \mathbb{Z}[X]$, $deg(u) > 0$
3. $\exists m \in \mathbb{N}$, $m > ||f||^k \cdot ||g||^n$
4. *u divise f et g modulo m.*

Alors f et g ne sont pas premiers entre eux dans $\mathbb{Z}[X]$.

Ce résultat est utilisé ainsi : f est le polynôme à factoriser, g est un candidat pour le facteur irréductible cherché (g sera trouvé par l'algorithme de Hensel). Si la condition sur la norme de g et m est vérifiée, et si g est irréductible, alors g doit diviser f. f et m étant fixés, il nous reste à trouver un candidat g dans L. On est donc ramené à la recherche d'un élément de L de "petite" norme, ce qui est le résultat de l'algorithme LLL présenté ci-après.

5.4.2 Algorithme LLL

L'algorithme dû à Lenstra, Lenstra et Lovacz date de 1982, et a beaucoup d'autres applications, en particulier en crypto-analyse.

On considère le $\mathbb{Z}$-module L engendré par la famille $(f_1, \ldots, f_n)$ de $\mathbb{R}^n$, les vecteurs étant supposés linéairement indépendants. On utilise la norme euclidienne sur $\mathbb{R}^n$. La norme de L est la valeur absolue du déterminant de la matrice des f_{ij} (on montre facilement que $|L|$ est indépendant du choix de la base).

orthogonalisation

On rappelle l'algorithme d'orthogonalisation de Gram-Schmidt :

Algorithme 4 *(Gram-Schmidt)*
Entrée : $(f_1, \ldots, f_n)$ *une base de* $\mathbb{R}^n$.
$f_1* = f_1$
pour $i = 2$ *à n faire*
$f_i* := f_i - \sum_{j<i} \mu_{ij} \cdot f_i*$, *avec* $\mu_{ij} = \frac{<f_i, f_j*>}{||f_j||^2}$

Cet algorithme est de complexité cubique si les coefficients sont rationnels.
Les f_i* vérifient les propriétés suivantes :

1. f_k* est la projection de f_k sur l'espace engendré par $(f_1*, \ldots, f_{k-1}*)$ (ce qui entraîne que $||f_k * || \leq ||f_k||$)
2. $< f_i*, f_j* >= 0 \forall i \neq j$
3. $det(f_{ij}) = det(f_{ij}*$

Une conséquence importante de ces résultats est l'inégalité d'Hadamar :

$$f_{ij} \leq B \Rightarrow |det(f_{ij}| \leq n^{n/2} \cdot B^n$$

De plus, si l'on prend un vecteur non nul quelconque de L, sa norme est supérieure à celle de la plus petite norme des f_i*.

Cas d'une famille génératrice entière

Considérons maintenant le cas où les f_i sont des vecteurs entiers. Les f_i* fournissent un vecteur de norme minimale, mais ils ne sont pas entiers ! Notre problème, c'est que pour l'instant nous n'avons pas de lien entre la norme des f_i et celle des f_i*. L'algorithme LLL construit une nouvelle base pour laquelle ce lien existe.

Algorithme 5 LLL
Entrée : $f_1, \ldots, f_n$ *de* $\mathbb{Z}^n$
pour $i = 1$ *à* n *faire* $g_i \leftarrow f_i$
calculer les $g*$ *et* μ *par l'algorithme de Gram-Schmidt*
$i \leftarrow 2$
tant que $i \leq n$ *faire*
    pour $j = i-1, i-2, \ldots, 1$ *faire*
        $g_i \leftarrow g_i - \lceil \mu_{ij} \rfloor \cdot g_j$ # $\lceil x \rfloor$ *dénote l'entier le plus proche de* x
    si $i > 1$ *et* $\|g *_{i-1}\|^2 > 2\|g *_i\|^2$
    alors échanger g_{i-1} *et* g_i., *mettre à jour* $g*$ *et* μ, $i \leftarrow i-1$
    sinon $i \leftarrow i+1$

Sortie : $g_1, \ldots, g_n$

Cet algorithme calcule une base entière de L, et cette base est réduite, c'est à dire que la base de Gram-Schmidt associée $g*$ vérifie :

$$\|g_i *\|^2 \leq norm g_{i+1} *^2 .$$

Théorème 6 *Soit une base* $g_1, \ldots, g_n$ *de* L *réduite. Alors quel que soit le vecteur* f *non nul de* L, $\|g_1\|^2 \leq 2^{n-1/2} \cdot \|f\|$

Ce résultat ne paraît pas très fort. Pourtant il suffit pour le problème de factorisation. On remarquera de plus qu'en pratique, g_1 est souvent un vecteur de taille minimale ou presque. Et la base des g_i est quasi-orthogonale.