Algorithmes en Calcul Formel et en Automatique
En cours de rédaction

Notes du cours 2-22 du MPRI, année 2006–2007
Version du 5 décembre 2006

Alin Bostan

Frédéric Chyzak

Marc Giusti

Bruno Salvy

Éric Schost

Table des matières

Les sections indiquées par ⋆ peuvent être omises en première lecture. Elles ne sont généralement pas traitées dans le cours oral.

COURS 1

Introduction

Le calcul formel calcule des objets mathématiques exacts. Ce cours « Algorithmes efficaces en calcul formel » explore deux directions : la calculabilité et l'efficacité. En particulier, de nombreuses questions portant sur les objets fondamentaux que sont les entiers, les polynômes et les séries admettent une réponse en complexité quasi-optimale.

Le choix des sujets couverts par le cours est guidé par un objectif simple : montrer comment des calculs d'intégrales ou de sommes de fonctions ou de suites spéciales de la combinatoire ou de la physique mathématique peuvent être abordés algorithmiquement. Les techniques reposent sur des calculs d'élimination et nous dédions de ce fait une grande place à l'étude effective des systèmes polynomiaux (bases standard, résolution géométrique) où ces questions sont classiques. Le souhait d'aboutir à une bonne complexité nous amène à traiter dans une première partie du cours l'algorithmique de base du calcul formel du point de vue de l'efficacité. Ce chapitre introductif présente rapidement le calcul formel et les notions de complexité, tels qu'ils seront développés dans l'ensemble du cours.

1. Décider, calculer

1.1. Fondements logiques. D'une certaine manière, le calcul formel est fondé sur une contrainte d'origine logique.

THÉORÈME 1 (Richardson). *Dans la classe des expressions obtenues à partir de* $\mathbb{Q}(x)$, π, $\log 2$ *par les opérations* $+, -, \times$ *et la composition avec* $\exp$, $\sin$ *et* $|\cdot|$, *le test d'équivalence à 0 est indécidable.*

Autrement dit, il n'existe pas d'algorithme permettant pour toute expression de cette classe de déterminer en temps fini si elle vaut 0 ou non. Plus généralement tout test d'égalité peut bien entendu se ramener à tester l'égalité à zéro dès que la soustraction existe. Cette limitation de nature théorique explique la difficulté et parfois la frustration que rencontrent les utilisateurs débutants des systèmes de calcul formel face à des fonctions de « simplification », qui ne peuvent être qu'heuristiques.

Pour effectuer un calcul, il est pourtant souvent crucial de déterminer si des expressions représentent 0 ou non, en particulier pour évaluer une fonction qui possède des singularités (comme la division). L'approche du calculateur formel expérimenté consiste à se ramener autant que faire se peut à des opérations d'un domaine dans lequel le test à zéro est décidable. Le calcul formel repose ainsi de manière naturelle sur des constructions algébriques qui préservent la décidabilité du test à 0. En particulier, les opérations courantes sur les vecteurs, matrices, polynômes, fractions rationnelles, ne nécessitent pas d'autre test à 0 que celui des coefficients.

1.2. Structures de base. Les objets les plus fondamentaux sont assez faciles à représenter en machine de manière exacte. Nous considérons tour à tour les plus importants d'entre eux, en commençant par les plus basiques.

Entiers machine. Les entiers fournis par les processeurs sont des entiers modulo une puissance de 2 (le nombre de bits d'un mot machine, typiquement 32 ou 64). Ils sont appelés des *entiers machine*. Les opérations rendues disponibles par le processeur sont l'addition, la soustraction, la multiplication et parfois la division. La norme ANSI du langage C fournit au programmeur la division et le modulo pour ces entiers, c'est-à-dire que le compilateur implante ces opérations si le processeur ne le fait pas.

Entiers. Pour manipuler des entiers dont la taille dépasse celle d'un mot machine, il est commode de les considérer comme écrits dans une base B assez grande :

$$N = a_0 + a_1 B + \cdots + a_k B^k.$$

L'écriture est unique si l'on impose $0 \leq a_i < B$. (Le signe est stocké séparément.) Ces nombres peuvent être stockés dans des tableaux d'entiers machine. Les objets obtenus sont des entiers de taille arbitraire appelés parfois *bignums*.

L'addition et le produit peuvent alors être réduits à des opérations sur des entiers inférieurs à B^2, au prix de quelques opérations de propagation de retenue. Le choix de B dépend un peu du processeur. Si le processeur dispose d'une instruction effectuant le produit de deux entiers de taille égale à celle d'un mot machine, renvoyant le résultat dans deux mots machines, alors B pourra être pris aussi grand que le plus grand entier tenant dans un mot machine. Sinon, c'est la racine carré de ce nombre qui sera utilisée pour B.

Entiers modulaires. Les calculs avec des polynômes, des fractions rationnelles ou des matrices à coefficients entiers souffrent souvent d'une maladie propre au calcul formel : la croissance des expressions intermédiaires. Les entiers produits comme coefficients des expressions intervenant lors du calcul sont de taille disproportionnée par rapport à ceux qui figurent dans l'entrée et dans la sortie.

Exemple 1. Voici le déroulement typique du calcul du plus grand diviseur commun (pgcd) de deux polynômes à coefficients entiers par l'algorithme d'Euclide :

$$P_0 = 7x^5 - 22x^4 + 55x^3 + 94x^2 - 87x + 56,$$
$$P_1 = 62x^4 - 97x^3 + 73x^2 + 4x + 83,$$
$$P_2 = \operatorname{rem}(P_0, P_1) = \frac{113293}{3844}x^3 + \frac{409605}{3844}x^2 - \frac{183855}{1922}x + \frac{272119}{3844},$$
$$P_3 = \operatorname{rem}(P_1, P_2) = \frac{18423282923092}{12835303849}x^2 - \frac{15239170790368}{12835303849}x + \frac{10966361258256}{12835303849},$$
$$P_4 = \operatorname{rem}(P_2, P_3) = -\frac{216132274653792395448637}{44148979404824831944178}x - \frac{631179956389122192280133}{88297958809649663888356},$$
$$P_5 = \operatorname{rem}(P_3, P_4) = \frac{20556791167692068695002336923491296504125}{3639427682941980248860941972667354081}.$$

Chaque étape calcule le reste (noté rem pour *remainder*) de la division euclidienne des deux polynômes précédents. Les coefficients de ces polynômes intermédiaires

font intervenir des entiers qui croissent de manière exponentielle, alors que le résultat recherché est 1.

Les entiers modulaires remédient à ce problème de deux manières. D'une part, pour un calcul de décision, de dimension, ou de degré, l'exécution de l'algorithme sur la réduction de l'entrée modulo un nombre premier donne un algorithme probabiliste répondant à la question. Cette technique peut aussi servir de base à un algorithme déterministe lorsque les nombres premiers pour lesquels la réponse est fausse peuvent être maîtrisés. C'est le cas du pgcd : en évitant les premiers qui divisent les coefficients de tête des deux polynômes, le degré du pgcd modulaire est le même que le degré du pgcd exact.

D'autre part, les entiers modulaires sont utilisés dans les algorithmes reposant sur le théorème des restes chinois. Ce théorème indique qu'un entier inférieur au produit de nombres premiers $p_1 \cdots p_k$ peut être reconstruit à partir de ses réductions modulo $p_1, \ldots, p_k$. Lorsqu'une borne sur la taille du résultat est disponible, il suffit d'effectuer le calcul modulo suffisamment de nombres premiers (choisis assez grands pour que leur nombre soit faible et assez petits pour que les opérations tiennent dans un mot machine), pour ensuite reconstruire le résultat, court-circuitant de la sorte toute croissance intermédiaire.

Rationnels. Les rationnels peuvent être stockés comme des paires où numérateur et dénominateur sont des entiers de taille arbitraire. Les opérations d'addition et de multiplication se réduisent aux opérations analogues sur les entiers et le test d'égalité à zéro se réduit au test d'égalité à 0 sur le numérateur. L'implantation d'un calcul de plus grand dénominateur commun (pgcd) permet aussi de réduire ces fractions et donne une forme normale où le test d'égalité est facile.

Vecteurs et matrices. Une fois donnée une représentation exacte pour des coefficients, il est facile de construire des vecteurs ou matrices comme des tableaux, ou plus souvent comme des tableaux de pointeurs sur les coefficients. Les opérations de produit par un scalaire, de produit de matrices ou de produit d'une matrice par un vecteur se réduisent aux opérations d'addition et de multiplication sur les coefficients. Il en va de même de la recherche de noyau ou d'inverse de matrices.

Polynômes et fractions rationnelles. Les polynômes peuvent être stockés de plusieurs manières, et la meilleure représentation dépend des opérations que l'on souhaite effectuer. Pour un polynôme en une variable, les choix principaux sont :

- la représentation dense : comme pour les entiers, le polynôme est représenté comme un tableau de (pointeurs sur les) coefficients ;
- la représentation creuse : le polynôme est représenté comme une liste de paires (coefficient, exposant) généralement triée par les exposants.

Ces représentations peuvent être utilisées récursivement pour stocker des polynômes multivariés. De même que pour les entiers, les fractions rationnelles sont représentées par des paires de polynômes et la réduction est possible dès lors qu'un pgcd est disponible. Les opérations d'addition, produit, division euclidienne, pgcd, se réduisent aux additions et multiplications sur les coefficients.

Ces constructions sont possibles dès que les coefficients sont disponibles. Il est donc possible par exemple de manipuler des polynômes dont les coefficients sont des rationnels, des entiers modulaires, ou des matrices.

Séries tronquées. Les séries tronquées

$$\sum_{k=0}^{N} a_k x^k + O(x^{N+1})$$

se représentent pratiquement comme des polynômes. La différence principale apparaît lors du produit : les coefficients des termes d'exposant au moins $N+1$ n'ont pas besoin d'être calculés, ni stockés. Cette structure de données joue un rôle très important non seulement pour des calculs d'approximations, mais aussi, comme on le verra dans le cours, comme une représentation *exacte*. En voici trois exemples importants qui seront abordés dans le cours :

1. Une fraction rationnelle dont les numérateurs et dénominateurs ont degré borné par d peut être reconstruite à partir d'un développement en série à l'ordre $2d+1$. Cette représentation joue ainsi un rôle clé dans le calcul efficace de la division euclidienne de polynômes (Cours 3) ; de suites récurrentes linéaires, comme le calcul rapide du $10\,000^{\text{e}}$ nombre de Fibonacci (Cours 4) ; le calcul du polynôme minimal d'une matrice creuse (Cours 26).
2. Un polynôme en deux variables peut être reconstruit à partir du développement en série d'une solution. L'efficacité de la résolution de systèmes polynomiaux par la méthode de la *résolution géométrique* abordée au cours 20 repose de manière cruciale sur cette opération, qui doit être effectuée rapidement.
3. Il est possible de reconstruire une équation différentielle linéaire à coefficients polynomiaux à partir du développement en série d'une solution et de bornes sur l'ordre et le degré des coefficients. De façon analogue, il est possible de reconstruire une récurrence linéaire à coefficients polynomiaux à partir des premiers termes d'une de ses solutions.

1.3. Équations comme structures de données. Une fois construits les objets de base que sont les polynômes, les séries ou les matrices, il est possible d'aborder des objets mathématiques construits *implicitement*. Ainsi, il est bien connu qu'il n'est pas possible de représenter toutes les solutions de polynômes de haut degré par radicaux, mais de nombreuses opérations sur ces solutions sont aisées en prenant le polynôme lui-même comme structure de données. Ce point de vue permet d'étendre le domaine d'application du calcul formel pourvu que des algorithmes soient disponibles pour effectuer les opérations souhaitées (typiquement addition, multiplication, multiplication par un scalaire, test d'égalité) par manipulation des équations elles-mêmes.

Nombres algébriques. C'est ainsi que l'on nomme les solutions de polynômes univariés. Les opérations d'addition et de multiplication peuvent être effectuées à l'aide de résultants (Cours 10). Ceux-ci peuvent être calculés efficacement à l'aide de séries (Cours 3). La division s'obtient par l'algorithme d'Euclide (Cours 10), et le test à

zéro se déduit du pgcd. Par exemple, il est possible de prouver assez facilement une identité comme

$$\frac{\sin\frac{2\pi}{7}}{\sin^2\frac{3\pi}{7}} - \frac{\sin\frac{\pi}{7}}{\sin^2\frac{2\pi}{7}} + \frac{\sin\frac{3\pi}{7}}{\sin^2\frac{\pi}{7}} = 2\sqrt{7}$$

une fois que l'on reconnaît qu'il s'agit d'une égalité entre nombres algébriques.

Systèmes polynomiaux. Vu l'importance des systèmes polynomiaux, une grande partie du cours leur sera consacrée (Cours 13 à 21). En considérant un système de polynômes, les questions naturelles qui peuvent être résolues sont l'existence de solutions, la dimension de l'espace des solutions (qui indique s'il s'agit d'une surface, d'une courbe, ou de points isolés), le degré, ou le calcul d'une paramétrisation de l'ensemble des solutions.

Il est également possible d'éliminer une ou des variables entre des polynômes. Cette opération peut s'interpréter géométriquement comme une projection. Dans le cas le plus simple, elle permet de calculer un polynôme s'annulant sur les abscisses des intersections de deux courbes. Une autre application est l'implicitisation, qui permet par exemple de calculer une équation pour une courbe donnée sous forme paramétrée.

Équations différentielles linéaires. Cette structure de données permet de représenter de nombreuses fonctions usuelles (exponentielle, fonctions trigonométriques et trigonométriques hyperboliques, leurs réciproques) ainsi que de nombreuses fonctions spéciales de la physique mathématique (fonctions de Bessel, de Struve, d'Anger, . . ., fonctions hypergéométriques et hypergéométriques généralisées), ainsi bien sûr que de multiples fonctions auxquelles n'est pas attaché un nom classique. Les opérations d'addition et de produit sont effectuées par des variantes noncommutatives du résultant qui se ramènent à de l'algèbre linéaire élémentaire (Cours 12). Le test à zéro se ramène à tester l'égalité d'un nombre fini de conditions initiales.

Ainsi, des identités élémentaires comme $\sin^2 x + \cos^2 x = 1$ sont non seulement facilement prouvables algorithmiquement, mais elles sont également calculables, c'est-à-dire que le membre droit se calcule à partir du membre gauche. Les relations étroites entre équations différentielles linéaires et récurrences linéaires — les séries solutions des unes ont pour coefficients les solutions des autres — amènent aux mêmes réponses algorithmiques à des questions sur des suites. Par exemple, l'identité de Cassini sur les nombres de Fibonacci

$$F_{n+2}F_n - F_{n+1}^2 = (-1)^{n+1}, \qquad n \geq 0$$

est exactement du même niveau de difficulté que $\sin^2 x + \cos^2 x = 1$.

Systèmes d'équations différentielles et de récurrences linéaires. Ces systèmes sont aux équations ce que les systèmes polynomiaux sont aux polynômes en une variable. Les mêmes opérations sont disponibles. En particulier, l'élimination s'étend dans ce cadre en introduisant des algèbres d'opérateurs adaptés (Cours 24). Une application très importante de cette élimination, la *création téléscopique*, permet de calculer

automatiquement des sommes et des intégrales définies. Ainsi,

$$\sum_{k=0}^{n}\left(\sum_{j=0}^{k}\binom{n}{j}\right)^3 = n2^{3n-1}+2^{3n}-3n2^{n-2}\binom{2n}{n},$$

$$\sum_{n=0}^{\infty} H_n(x)H_n(y)\frac{u^n}{n!} = \frac{\exp\left(\frac{4u(xy-u(x^2+y^2))}{1-4u^2}\right)}{\sqrt{1-u^2}},$$

$$\frac{1}{2}J_0(x)^2+J_1(x)^2+J_2(x)^2+\cdots=\frac{1}{2},$$

$$\int_{-1}^{+1}\frac{e^{-px}T_n(x)}{\sqrt{1-x^2}}\,dx=(-1)^n\pi I_n(p),$$

$$\int_0^{+\infty} xe^{-px^2}J_n(bx)I_n(cx)\,dx=\frac{1}{2p}\exp\left(\frac{c^2-b^2}{4p}\right)J_n\left(\frac{bc}{2p}\right),$$

$$\int_0^{+\infty} xJ_1(ax)I_1(ax)Y_0(x)K_0(x)\,dx=-\frac{\ln(1-a^4)}{2\pi a^2},$$

$$\sum_{k=0}^{n}\frac{q^{k^2}}{(q;q)_k(q;q)_{n-k}}=\sum_{k=-n}^{n}\frac{(-1)^kq^{(5k^2-k)/2}}{(q;q)_{n-k}(q;q)_{n+k}},$$

formules qui mettent en jeu diverses fonctions spéciales ou polynômes orthogonaux classiques, peuvent être prouvées automatiquement. Les algorithmes correspondants seront décrits au cours 25.

Ainsi, les exemples ci-dessus illustrent bien la manière dont le calcul formel parvient à effectuer de nombreux calculs utiles dans les applications malgré l'indécidabilité révélée par le théorème de Richardson.

2. Calculer rapidement

En pratique, la calculabilité n'indique que la faisabilité. Il faut disposer d'algorithmes efficaces et d'une bonne implantation pour pouvoir effectuer des calculs de grande taille. La première partie de ce cours (Cours 2 à 12) est consacrée aux algorithmes efficaces sur les structures de base du calcul formel. L'efficacité sera mesurée par la théorie de la complexité et nous ferons ressortir des principes récurrents dans la conception d'algorithmes efficaces.

Exemple 2. Pour donner une idée de ce que veut dire rapidement, voici ce qui peut être calculé en *une minute* avec le système Magma sur une machine de bureau d'aujourd'hui[1], en notant $\mathbb{K}$ le corps $\mathbb{Z}/p\mathbb{Z}$ à p éléments, $p = 67108879$ étant un nombre premier de 26 bits (dont le carré tient sur un mot machine) :

1. Entiers :
   - produit de deux entiers avec 200 000 000 de chiffres ;
   - factorielle de 4 000 000 (environ 25 000 000 de chiffres) ;
   - factorisation d'un entier de 45 chiffres (150 bits).
2. Polynômes dans $\mathbb{K}[x]$:

[1]Ce texte est écrit en 2006. La machine a un processeur AMD 64 à 2,2 GHz et une mémoire de 2 Go ; le système d'exploitation est linux.

   - produit de deux polynômes de degré 8 000 000 (plus d'un an avec la méthode naïve) ;
   - pgcd et résultant de deux polynômes de degré 200 000 ;
   - factorisation d'un polynôme de degré 2 000.
3. Polynômes dans $\mathbb{K}[x, y]$:
   - résultant de deux polynômes de degré total 100 (sortie de degré 10 000) ;
   - produit et somme de deux nombres algébriques de degré 450 (sortie de degré 200 000) ;
   - factorisation d'un polynôme de degré 500 en deux variables.
4. Matrices :
   - déterminant d'une matrice $3\,500 \times 3\,500$ à coefficients dans $\mathbb{K}$;
   - polynôme caractéristique d'une matrice $2\,000 \times 2\,000$ à coefficients dans $\mathbb{K}$;
   - déterminant d'une matrice 700×700 dont les coefficients sont des entiers 32 bits.

Ces exemples montrent qu'il est relativement aisé de calculer avec des objets de taille colossale (mais pas avec les algorithmes naïfs), et donnent envie d'une mesure de complexité des différents algorithmes permettant d'expliquer, voire de prédire, les différences de tailles atteintes pour ces questions.

2.1. Mesures de complexité. Pour bien définir la complexité, il faut se donner : un modèle de machine ; les opérations disponibles sur cette machine ; leur coût unitaire. La complexité en espace mesure la mémoire utilisée par l'exécution de l'algorithme, et la complexité en temps, la somme des coûts unitaires des opérations effectuées par l'algorithme. Dans ce cours, nous ne nous intéresserons pas à la complexité en espace.

Machine RAM. Le modèle que nous utiliserons est celui de la *Random Access Machine* (RAM). Dans ce modèle, un programme lit et écrit des entiers sur deux bandes différentes et utilise un nombre arbitraire de registres entiers pour ses calculs intermédiaires. Les opérations élémentaires (l'assembleur de la machine) sont la lecture, l'écriture (sur bande ou en registre), l'addition, la soustraction, le produit, la division et trois instructions de saut : saut inconditionnel, saut si un registre est nul et saut si un registre est positif. Un point technique est que le programme ne fait pas partie des données, il n'est donc pas modifiable.

Complexité binaire ou arithmétique. Nous considérerons deux mesures de complexité :

1. Dans la complexité *binaire*, les bandes d'entrée et de sortie ainsi que les registres ne peuvent stocker que des *bits* (0 ou 1). La mesure de complexité des algorithmes opérant sur une telle machine tient compte de la taille des entiers manipulés et modélise précisément le temps de calcul.
2. Dans la complexité *arithmétique*, les opérations sur les entiers ont coût unitaire. Cette mesure modélise précisément le temps de calcul pour des calculs sur des entiers modulaires ou sur les flottants machine. Nous étendrons cette mesure au cas où les objets manipulés ne sont pas des entiers, mais plus

généralement des éléments d'un corps k donné et nous mesurerons alors la complexité en nombre d'opérations arithmétique dans k.

EXEMPLE 3. Le calcul de $n!$ par la méthode naïve requiert n opérations arithmétiques et $O(n^2 \log^2 n)$ opérations binaires. Nous verrons au cours 4 qu'il est possible d'abaisser ce coût à seulement $O(n^{1/2} \log n)$ opérations arithmétiques, et $O(n \log^3 n)$ opérations binaires. Les algorithmes rapides permettant d'atteindre ces complexités fournissent le meilleur algorithme connu de factorisation déterministe d'entiers et des algorithmes très efficaces pour le calcul de millions de décimales de π, $\log 2$ et de nombreuses autres constantes.

Taille. Un algorithme et une structure de données sont généralement dotés d'une notion naturelle de taille et il s'agit d'étudier le coût de l'algorithme en fonction de cette taille. Pour simplifier, il est souvent commode de considérer le comportement asymptotique de ce coût lorsque la taille tend vers l'infini. Il est important de comprendre que la complexité d'un problème n'a de sens qu'une fois la structure de donnée fixée pour l'entrée comme pour la sortie.

Par exemple, pour les polynômes, le choix de la représentation dense mène à mesurer la complexité par rapport au degré, alors que le choix de la représentation creuse met en avant le nombre de monômes. Pour la factorisation, la complexité est polynomiale en le degré, mais exponentielle en le nombre de monômes, dans le cas le pire.

Cas le pire, cas moyen. La complexité dans le cas le pire est le maximum des complexités pour toutes les entrées d'une taille donnée. C'est celle que nous étudierons. Il est souvent utile de considérer aussi la complexité en moyenne, lorsque l'on peut mettre une mesure sur l'ensemble des entrées de taille bornée. Pour la plupart des algorithmes que nous étudierons dans la première partie de ce cours, il n'y a pas de différence importante entre les deux. Ce n'est plus le cas en revanche pour la complexité des algorithmes sur les systèmes polynomiaux (où la notion de complexité en moyenne est avantageusement remplacée par celle de complexité dans le cas générique).

Bornes inférieures. La recherche de bornes inférieures de complexité est très difficile. Par exemple, à l'heure actuelle on ne sait pas prouver que la multiplication de matrices est nécessairement plus coûteuse qu'un nombre borné d'additions. Dès qu'il est possible de montrer que tous les bits de l'entrée doivent être pris en compte, la somme de la taille de l'entrée et de la taille de la sortie est une borne inférieure sur la complexité. En effet, dans le modèle RAM, chacune des écritures et des lectures prend une opération. Si N est cette borne inférieure, l'algorithme sera dit *quasi-optimal* lorsque sa complexité sera bornée par $O(N \log^k N)$ pour un $k \geq 0$ arbitraire. L'essentiel de la première partie du cours consistera à rechercher des algorithmes quasi-optimaux pour les opérations de base sur les structures de données fondamentales.

2.2. La notation $O(\cdot)$. Nous utilisons la notation $O(\cdot)$ pour exprimer une borne sur la complexité des algorithmes. La signification précise de la notation

$$f(n) = O(g(n)), \qquad n \to \infty$$

est qu'il existe $K > 0$ et $A > 0$ tels que pour tout $n > A$, f et g soient liés par l'inégalité

$$|f(n)| \leq K|g(n)|.$$

Lorsque plusieurs paramètres interviennent dans l'analyse de la complexité, il faut absolument préciser lequel tend vers l'infini pour que cette notation ait un sens. Si plusieurs paramètres tendent vers l'infini, soit ils sont liés par des inégalités qui seront précisées, soit la définition ci-dessus s'étend avec une constante K qui ne dépend d'aucun des paramètres.

La notation $O(\cdot)$ intervient aussi dans ce cours pour représenter la troncature des séries. L'expression

$$f(x) := g(x) + O(x^N)$$

signifiera que le polynôme ou la série g est tronqué après son N^{e} terme et que le résultat, un polynôme, est stocké dans f.

2.3. Diviser pour régner. Le principe le plus important de la conception d'algorithmes efficaces est le paradigme « diviser pour régner ». Il consiste à résoudre un problème en le réduisant à un certain nombre m d'entrées de taille divisées par p (le plus souvent $p = 2$) puis à recombiner les résultats. Le coût de la recombinaison et éventuellement du découpage préliminaire est borné par une fonction f de la taille des entrées. Le coût total dépend de la croissance de f par rapport à N. Un cadre commode pour nos applications est résumé dans le lemme suivant.

LEMME 1 (« Diviser pour régner »). *S'il existe $q > 1$ tel que pour tout réel $x \geq 1$, la relation $f(x) \geq qf(x/p)$ soit vérifiée, alors pour tout $m \geq 1$ et $x \geq 1$, toute solution de l'inégalité*

$$C(x) \leq f(x) + mC(x/p) \quad \textit{pour } x > p \qquad \textit{et} \qquad C(x) \leq c \quad \textit{pour } x \leq p$$

vérifie

$$C(x) \leq cm^{\lceil \log_p x \rceil} + \begin{cases} \frac{1}{1-\frac{m}{q}} f(x) & \textit{si } q > m, \\ f(x)\lceil \log_p x \rceil & \textit{si } q = m, \\ \frac{x^{\log_p(m/q)}}{1-\frac{q}{m}} f(x) & \textit{si } q < m. \end{cases}$$

La notation $\lceil x \rceil$ désigne l'entier k tel que $k - 1 < x \leq k$, et $\log_p x$ représente le logarithme en base p de x, c'est-à-dire $\log x / \log p$.

En pratique, la fonction f représente le coût d'un algorithme et n'est souvent définie que pour des arguments entiers. Dans ce cas, la complexité de l'algorithme appliqué à une entrée de taille x sera souvent bornée par la complexité de l'algorithme lorsqu'il est appliqué à des entrées de taille supérieure, et on prendra la puissance de 2 immédiatement supérieure pour pouvoir appliquer le lemme. Comme cette puissance est à moins d'un facteur 2 de x, les estimations perdent au pire un facteur q.

Démonstration. En appliquant plusieurs fois l'inégalité, il vient

$$\begin{aligned} C(x) &\leq f(x) + mC\left(\frac{x}{p}\right), \\ &\leq f(x) + mf\left(\frac{x}{p}\right) + m^2 C\left(\frac{x^2}{p^2}\right), \\ &\leq f(x) + mf\left(\frac{x}{p}\right) + m^2 f\left(\frac{x}{p^2}\right) + \cdots + m^{\lceil \log_p x \rceil} C\left(\frac{x}{p^{\lfloor \log_p x \rfloor}}\right). \end{aligned}$$

L'inégalité vérifiée par f entraîne par récurrence

$$f\left(\frac{x}{p^k}\right) \leq q^{-k} f(x).$$

La somme considérée est donc majorée par

$$cm^{\lceil \log_p x \rceil} + f(x)\left(1 + \frac{m}{q} + \left(\frac{m}{q}\right)^2 + \cdots + \left(\frac{m}{q}\right)^{\lceil \log_p x \rceil - 1}\right).$$

Soit S la somme entre parenthèses. Si $q > m$, la série géométrique S est convergente, d'où le résultat. Si $q = m$, tous les termes de S sont égaux à un et leur nombre est $\lceil \log_p x \rceil$. Enfin, si $q < m$, on se ramène encore à une série géométrique en réécrivant S sous la forme

$$\left(\frac{m}{q}\right)^{\lceil \log_p x \rceil - 1}\left(1 + \frac{q}{m} + \left(\frac{q}{m}\right)^2 + \cdots + \left(\frac{q}{m}\right)^{\lceil \log_p x \rceil - 1}\right).$$

□

Exemple 4. Pour trier un tableau de N éléments, le tri fusion utilise un « diviser pour régner » : chaque moitié du tableau est triée récursivement, donc $p = m = 2$, et les deux moitiés sont ensuite fusionnées en au plus N comparaisons, donc $q = 2$. Si N est une puissance de 2 $\lceil \log_2 N \rceil = \log_2 N$ et la complexité est donc d'au plus $N \log N + N$ comparaisons pour trier une liste de taille N.

Exemple 5. La complexité de l'algorithme de Karatsuba du cours 2 s'obtient avec $m = 3$, $p = 2$ et f linéaire (donc $q = p$).

3. Organisation du cours

Le diagramme de la Figure 1 montre les dépendances entre les cours.(Un cours est au-dessous d'un autre lorsqu'il utilise des notions qui y ont été présentées.)

Notes

Les références générales sur les algorithmes du calcul formel sont deux livres : celui de von zur Gathen et Gerhard [10] et celui, plus élémentaire, de Geddes, Czapor et Labahn [5]. La complexité est également utilisée comme fil conducteur dans le livre plus difficile de Bürgisser, Clausen et Shokrollahi [3]. Une bonne introduction à la seconde partie du cours, sur les systèmes polynomiaux, est le livre de Cox, Little et O'Shea [4]. De premiers éléments pour aborder la partie 3, sur les équations différentielles et les récurrences linéaires, sont donnés dans le livre $A = B$ de Petkovšek, Wilf et Zeilberger [7].

Le théorème de Richardson [8] s'applique à des fonctions. Pour des constantes, l'approche la plus récente [9] réduit le test à zéro à une conjecture de théorie des

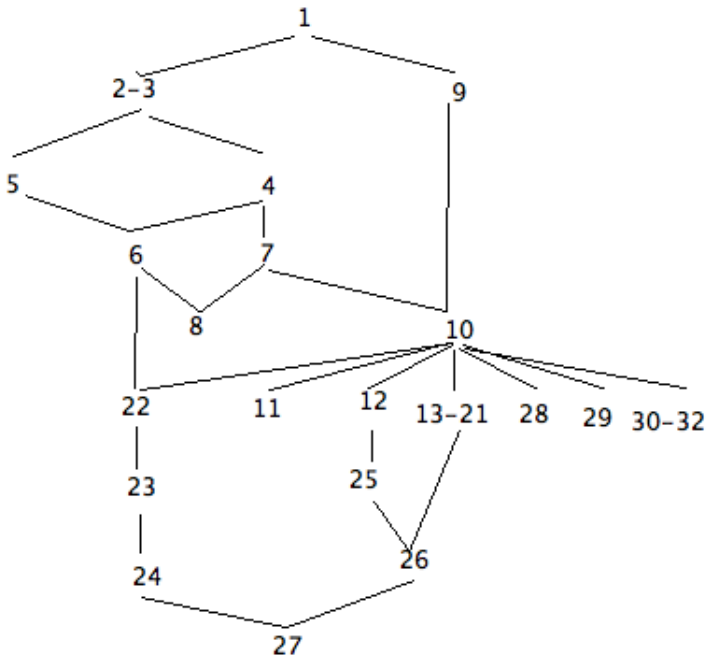


FIG. 1.

nombres due à Schanuell qui exprime que les seules relations entre exponentielles et logarithmes sont celles qui découlent des formules d'addition et de multiplication.

L'implantation d'une arithmétique efficace pour les entiers longs (bignums) est un travail très délicat. Une des meilleures arithmétiques disponibles est fournie par GMP, le *Gnu Multiprecision Package* [6]. Elle est le résultat d'un travail de nombreuses années, qui comporte une partie importante de code assembleur consacré à la multiplication sur chacun des processeurs produits dans une période récente. Les entiers de GMP sont ceux qui sont utilisés dans Maple pour les grandes tailles. D'autres entiers très efficaces sont implantés dans le système Magma.

Les différents modèles de complexité (machine RAM, machine de Turing, *straight-line program*, ...) sont bien présentés par Aho, Hopcrof et Ulmann dans [1].

Bibliographie

[1] Aho (Alfred V.), Hopcroft (John E.), and Ullman (Jeffrey D.). – *The design and analysis of computer algorithms*. – Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1974, x+470p. Addison-Wesley Series in Computer Science and Information Processing.

[2] Beck (Matthias), Berndt (Bruce C.), Chan (O-Yeat), and Zaharescu (Alexandru). – Determinations of analogues of Gauss sums and other trigonometric sums. *International Journal of Number Theory*, vol. 1, n° 3, 2005, pp. 333–356.

[3] Bürgisser (Peter), Clausen (Michael), and Shokrollahi (M. Amin). – *Algebraic complexity theory*. – Springer-Verlag, Berlin, 1997, *Grundlehren der Mathematischen Wissenschaften*, vol. 315, xxiv+618p.

[4] Cox (David), Little (John), and O'Shea (Donal). – *Ideals, varieties, and algorithms*. – Springer-Verlag, New York, 1996, second edition, xiv+536p.

[5] Geddes (Keith O.), Czapor (Stephen R.), and Labahn (George). – *Algorithms for Computer Algebra*. – Kluwer Academic Publishers, 1992.

[6] Granlund (Torbjörn). – *GNU Multiple Precision Arithmetic Library*. – http://swox.com/gmp, 2006.

[7] Petkovšek (Marko), Wilf (Herbert S.), and Zeilberger (Doron). – $A = B$. – A. K. Peters, Wellesley, MA, 1996, xii+212p.

[8] Richardson (Daniel). – Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, vol. 33, n° 4, 1968, pp. 514–520.

[9] Richardson (Daniel). – How to recognize zero. *Journal of Symbolic Computation*, vol. 24, n° 6, 1997, pp. 627–645.

[10] von zur Gathen (Joachim) and Gerhard (Jürgen). – *Modern computer algebra*. – Cambridge University Press, New York, 1999, xiv+753p.

Première partie

Algorithmes Fondamentaux

COURS 2

Multiplication rapide

Résumé

Le produit de polynômes et d'entiers est une opération élémentaire, qui intervient dans un nombre impressionnant d'algorithmes du calcul formel. L'efficacité de ces algorithmes repose donc sur celle du produit. Pour multiplier deux polynômes de degré n à coefficients dans un anneau A, la méthode classique requiert $O(n^2)$ opérations dans A. De même, l'algorithme scolaire de multiplication de deux entiers à n chiffres nécessite un nombre d'opérations binaires en $O(n^2)$. Nous présentons dans ce cours plusieurs algorithmes de multiplication rapide, dont celui de Karatsuba, de complexité $O(n^{1,59})$, ainsi que ceux utilisant la transformée de Fourier rapide, dont la complexité est essentiellement linéaire en n.

1. Introduction, résultats principaux

Les problèmes abordés dans ce cours concernent la complexité arithmétique de la multiplication des polynômes à une variable et la complexité binaire de la multiplication des entiers. Au vu de l'exemple suivant, il est facile de se convaincre de la similitude des deux questions :

Polynômes : Soient à multiplier $3X^2 + 2X + 1$ et $6X^2 + 5X + 4$ dans $\mathbb{Z}[X]$.

$$(3X^2 + 2X + 1) \times (6X^2 + 5X + 4)$$

$$= (3 \cdot 6)X^4 + (3 \cdot 5 + 2 \cdot 6)X^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)X^2 + (2 \cdot 4 + 1 \cdot 5)X + (1 \cdot 4)$$

$$= 18X^4 + 27X^3 + 28X^2 + 13X + 4.$$

Nombres entiers : Soient à multiplier 321 et 654 en base 10.

$$(3 \cdot 10^2 + 2 \cdot 10 + 1) \times (6 \cdot 10^2 + 5 \cdot 10 + 4)$$

$$= (3 \cdot 6)10^4 + (3 \cdot 5 + 2 \cdot 6)10^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)10^2 + (2 \cdot 4 + 1 \cdot 5)10 + (1 \cdot 4)$$

$$= 18 \cdot 10^4 + 27 \cdot 10^3 + 28 \cdot 10^2 + 13 \cdot 10 + 4$$

$$= 2 \cdot 10^5 + 9 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10 + 4 = 209934.$$

Dans les deux cas, nous avons retranscrit l'algorithme naïf, et la suite des calculs est essentiellement la même, si ce n'est que, dans le cas des entiers, il faut en outre gérer les retenues (dernière égalité de l'exemple). On ne sera donc pas surpris que les résultats obtenus dans les deux cas soient très semblables.

Résultats. Dans toute la suite $(A, +, \times)$ désignera un anneau (commutatif et unitaire). Tout d'abord, nous considérons le cas arithmétique ; il s'agit de minimiser le coût, en termes du nombre d'opérations $(+, -, \times)$ dans A, du produit des polynômes en degré borné. Les premiers résultats à retenir de ce cours sont les suivants.

La multiplication des polynômes de degré au plus n dans $A[X]$ requiert :
- $O(n^2)$ *opérations dans A par* l'algorithme naïf *;*
- $O(n^{1,59})$ *opérations dans A par* l'algorithme de Karatsuba *;*
- $O(n \log n \log\log n)$*, voire dans certains cas $O(n \log n)$ opérations dans A, via* la transformée de Fourier rapide (FFT).

Ainsi, la multiplication des polynômes peut se faire en un coût arithmétique *essentiellement linéaire* en leur degré.

Cette diversité d'algorithmes motive l'introduction de la notion de *fonctions de multiplication* (notées usuellement $\mathsf{M}(n)$), qui estiment le nombre d'opérations suffisantes pour multiplier des polynômes : une application $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ est une fonction de multiplication si on peut multiplier les polynômes de degré au plus n en au plus $\mathsf{M}(n)$ opérations (à quelques détails techniques près ; ces fonctions sont définies plus précisément plus loin). De la sorte, le coût de la multiplication devient un mètre étalon pour mesurer le coût d'autres algorithmes.

La multiplication des polynômes est omniprésente : les algorithmes de calcul de pgcd (plus grand commun diviseur), de pgcd étendu, de factorisation en une ou plusieurs variables, de composition des séries formelles, d'évaluation multipoint, d'interpolation, font tous intervenir des produits de polynômes, et à ce titre, leur complexité s'énonce naturellement en termes de fonctions de multiplication M.

L'analogie entre les entiers et les polynômes va très loin ; la plupart des réponses apportées dans le cadre de la complexité arithmétique trouvent un équivalent en complexité binaire. Cependant, aucun théorème d'équivalence n'est connu ; il se trouve que les mêmes idées algorithmiques s'adaptent plus ou moins facilement dans les deux cadres. Ainsi, on dispose des résultats suivants dans le modèle binaire.

On peut multiplier des entiers de n chiffres binaires par :
- l'algorithme naïf *en $O(n^2)$ opérations binaires ;*
- l'algorithme de Karatsuba *en $O(n^{1,59})$ opérations binaires ;*
- l'algorithme de Schönhage-Strassen *en $O(n \log n \log\log n)$ opérations binaires.*

Les preuves de ces résultats de complexité binaire sont plus délicates que celles de leurs analogues polynomiaux, à cause des problèmes de gestion des retenues. Ainsi, dans la suite, nous ne traitons d'abord en détail que les versions polynomiales de ces résultats, le cas entier étant ensuite brièvement passé en revue.

En pratique. Les constantes cachées dans les $O(\cdot)$ sont déterminantes pour l'efficacité pratique de tels algorithmes. Parlons tout d'abord du cas polynômial, par exemple lorsque A est un corps fini de taille « raisonnable » (typiquement, dont les éléments sont représentés sur quelques mots machine). Dans les meilleures implantations actuelles (magma, NTL) :

- l'algorithme de Karatsuba bat l'algorithme naïf pour des degrés d'environ 20 ;
- les méthodes à base de FFT en $O(n \log n)$ gagnent pour des degrés de l'ordre de 100, mais ne peuvent pas être utilisées pour des degrés arbitrairement grands (vient un moment où on manque de racines de l'unité, voir plus loin) ;

- l'algorithme de type FFT en $O(n \log n \log\log n)$ est utilisé pour des degrés de l'ordre de quelques dizaines ou centaines de milliers.

Certains problèmes, en cryptologie ou en théorie des nombres, nécessitent de manipuler des polynômes de degré de l'ordre de 100 000, tailles auxquelles les algorithmes rapides sont indispensables.

L'implantation des algorithmes rapides pour les entiers est délicate en raison des retenues. Dans les meilleures implantations actuelles (magma, GMP) :

- l'algorithme de Karatsuba bat l'algorithme naïf pour des nombres de l'ordre de 100 chiffres binaires ;
- les méthodes à base de FFT (Schönhage-Strassen) gagnent pour des nombres d'environ 10 000 chiffres binaires.

À nouveau, des problèmes venus de cryptologie ou de théorie des nombres demandent de manipuler des nombres de taille colossale (de l'ordre de 100 000 000 de chiffres ; il faut 10 Mo pour stocker un tel nombre). Ceci justifie amplement les efforts d'implantation d'algorithmes rapides.

Fixons les notations : on travaille avec des polynômes F et G à coefficients dans un anneau A, ayant un degré au plus $n-1$, formellement

$$F = f_0 + \cdots + f_{n-1}X^{n-1} \quad \text{et} \quad G = g_0 + \cdots + g_{n-1}X^{n-1} \ ;$$

le problème est alors de calculer (les coefficients de)

$$H = FG = h_0 + \cdots + h_{2n-2}X^{2n-2}.$$

2. Algorithme naïf

Cet algorithme consiste à développer le produit, c'est-à-dire à écrire

$$H = FG = \sum_{i=0}^{2n-2} h_i X^i \quad \text{avec} \quad h_i = \sum_{j+k=i} f_j g_k.$$

Ainsi, calculer tous les h_i demande $O(n^2)$ opérations dans A. C'est un algorithme de complexité arithmétique *quadratique*.

Exercice 1. Montrer que, pour multiplier deux polynômes de degrés m et n, l'algorithme naïf demande au plus $(m+1) \times (n+1)$ multiplications dans A et mn additions dans A.

Exercice 2. Montrer que, pour multiplier deux entiers à n chiffres chacun, l'algorithme naïf demande $O(n^2)$ opérations binaires.

Exercice 3. Estimer la complexité binaire de la méthode naïve lorsque les polynômes ont degré n et des coefficients entiers bornés par H..

Exercice 4. Shigeru Kondo a calculé 25 000 000 000 décimales de π sur un pc de bureau[1]. Ce type de calcul repose sur la multiplication d'entiers. En supposant que la machine de Kondo était capable d'effectuer 10^{12} opérations à la seconde, montrer que Kondo n'a pas utilisé l'algorithme naïf.

[1]Calcul achevé le 7 mars 2003 sur un Pentium 4 3,2 Gz, 2 Go, après 17 jours et 14 heures, avec 100 Go d'espace disque utilisé. Le record actuel sur un super-ordinateur est de $1{,}2 \times 10^{12}$ chiffres, obtenu par Kanada et son équipe en 2002.

3. Algorithme de Karatsuba

Un premier raffinement de l'algorithme naïf se base sur la remarque suivante : il est possible de gagner *une* multiplication pour le produit des polynômes de degré 1. Soient en effet à multiplier les polynômes

$$F = f_0 + f_1 X \quad \text{et} \quad G = g_0 + g_1 X.$$

Le produit $H = FG$ s'écrit

$$H = f_0 g_0 + (f_0 g_1 + f_1 g_0) X + f_1 g_1 X^2.$$

Effectuer tous les 4 produits $f_0 g_0, f_0 g_1, f_1 g_0, f_1 g_1$ correspond à l'algorithme quadratique. Mais on peut faire mieux en remarquant que le coefficient de X s'écrit

$$f_0 g_1 + f_1 g_0 = (f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1.$$

Cette écriture mène à un algorithme qui effectue au total 3 multiplications et 4 additions. On a perdu quelques additions par rapport à l'algorithme naïf, mais le gain d'une multiplication va se transformer en gain dans l'*exposant* de l'algorithme, par application récursive.

Passons en effet au cas général des degrés quelconques. Inspirés par l'observation précédente, on va scinder F et G en deux. On suppose donc que F et G sont de degré au plus $n-1$, et que l'entier n est pair, $n = 2k$. On pose alors

$$F = F^{(0)} + F^{(1)} X^k, \quad G = G^{(0)} + G^{(1)} X^k,$$

$F^{(0)}, F^{(1)}, G^{(0)}, G^{(1)}$ ayant des degrés au plus $k-1$. Le produit $H = FG$ s'écrit

$$H = F^{(0)} G^{(0)} + (F^{(0)} G^{(1)} + F^{(1)} G^{(0)}) X^k + F^{(1)} G^{(1)} X^{2k}.$$

Pour écrire l'algorithme, on suppose que n est une puissance de 2 afin de pouvoir effectuer tous les appels récursifs que l'on souhaite. On obtient alors, avec les notations ci-dessus :

Algorithme de Karatsuba

Entrée : F, G de degré au plus $n-1$, n étant une puissance de 2.

Sortie : $H = FG$.

1. Si $n = 1$, renvoyer FG.
2. Calculer $A_1 = F^{(0)} G^{(0)}$ et $A_2 = F^{(1)} G^{(1)}$ récursivement.
3. Calculer $A_3 = F^{(0)} + F^{(1)}$ et $A_4 = G^{(0)} + G^{(1)}$.
4. Calculer $A_5 = A_3 A_4$ récursivement.
5. Calculer $A_6 = A_5 - A_1$ et $A_7 = A_6 - A_2$.
6. Renvoyer $A_1 + A_7 X^{n/2} + A_2 X^n$.

On peut maintenant établir la complexité de cet algorithme.

THÉORÈME 1. *Si n est une puissance de 2, l'algorithme de Karatsuba calcule le produit de deux polynômes de degreé au plus $n-1$ en $9n^{\log_2 3}$ opérations dans A.*

DÉMONSTRATION. Lors de l'appel en degré $< n$, on effectue 3 appels récursifs en degré $< n/2$, et quelques additions. Le coût $K(n)$ satisfait donc à la récurrence :

$$K(n) \leq 3K(n/2) + 4n,$$

où le terme $4n$ vient estimer le nombre d'additions. Le lemme « diviser pour régner » permet alors de conclure. □

On en déduit le résultat général suivant en degré quelconque.

COROLLAIRE 1. *On peut multiplier les polynômes de degré n (quelconque) en $O(n^{\log_2 3}) = O(n^{1,59})$ opérations dans A.*

En effet, soit N la plus petite puissance de 2 telle que $N > n$. Alors pour calculer le produit de F et de G il suffit de multiplier $\tilde{F} = X^{N-\deg(F)-1}F$ et $\tilde{G} = X^{N-\deg(G)-1}G$ via l'algorithme vu précédemment ; on perd au pire un facteur constant par rapport à l'estimation du Théorème 1.

EXERCICE 5. Estimer la constante cachée dans l'estimation asymptotique du Corollaire 1.

Observons que pour obtenir une bonne implantation en degré quelconque, la solution précédente (basée sur l'insertion artificielle de coefficients nuls) n'est pas forcément la meilleure, car elle induit la perte d'un facteur constant. Une solution alternative est d'adapter le découpage vu plus haut au cas où l'un des polynômes est de degré pair.

EXERCICE 6. Déterminer la complexité d'une variante de l'algorithme de Karatsuba où tout polynôme de degré impair $n = 2k-1$ est découpé en deux tranches de degré au plus $k-1$ et tout polynôme de degré pair $n = 2k$ est découpé en deux tranches de degré au plus k.

Par ailleurs, en fonction de la nature de l'anneau ou du corps de base, on peut vouloir arrêter les appels récursifs avant d'avoir atteint le degré 0. Ce choix dépend des vitesses relatives de l'addition et de la multiplication.

EXERCICE 7. Soit n une puissance de 2. Établir un algorithme hybride, qui fait appel à l'algorithme de Karatsuba pour $n > 2^d$ et à l'algorithme naïf pour $n \leq 2^d$. Montrer que la complexité arithmétique $\mathsf{C}(n)$ de cet algorithme vérifie $\mathsf{C}(n) \leq \gamma(d)n^{\log_2 3} - 8n$ pour tout $n \geq 2^d$, où $\gamma(d)$ est une fonction qui dépend uniquement de d. Trouver la valeur de d qui minimize $\gamma(d)$ et comparer le résultat avec celui du Théorème 1.

EXERCICE 8. Pensez-vous qu'il soit possible de multiplier deux polynômes de degré au plus 1 en utilisant seulement 2 multiplications dans l'anneau de base ?

EXERCICE 9. Soit A un anneau et soit P et Q deux polynômes de degré au plus 4 dans $A[X]$.

1. Estimer le nombre de multiplications de A requises par l'algorithme de Karatsuba pour calculer le produit AB ;
2. Donner un algorithme qui multiplie P et Q en utilisant au plus 7 multiplications dans A ;
3. En déduire un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{1,4})$;
4. Montrer que, pour tout entier $\alpha \geq 2$, il existe un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{\log_\alpha(2\alpha-1)})$;
5. Montrer que pour tout $\varepsilon > 0$, il existe un algorithme de multiplication polynomiale dans $A[X]$ de complexité arithmétique $O(n^{1+\varepsilon})$, où la constante dans le $O(\cdot)$ dépend de ε, mais pas de n.

4. Transformée de Fourier rapide

Les méthodes à base de transformée de Fourier sont ce que l'on sait faire de mieux pour multiplier les polynômes. Pour simplifier la présentation, on suppose ici que l'on cherche à multiplier des polynômes F et G dans $A[X]$, de degrés strictement inférieurs à $n/2$ (ou plus généralement tels que $\deg FG < n$).

On fait (provisoirement) l'hypothèse que l'anneau A est de cardinal au moins n. On se donne des points distincts $a_0, \ldots, a_{n-1}$ dans A. Le principe de la multiplication par transformation de Fourier est le suivant :

1. *Évaluation.* On calcule les valeurs :
$$\mathsf{Ev}(F) = (F(a_0), \ldots, F(a_{n-1}))\,; \quad \mathsf{Ev}(G) = (G(a_0), \ldots, G(a_{n-1}))\,.$$
2. *Produit point à point.*
$$\mathsf{Ev}(F), \mathsf{Ev}(G) \mapsto \mathsf{Ev}(FG) = (FG(a_0), \ldots, FG(a_{n-1}))\,.$$
3. *Interpolation.*
$$\mathsf{Ev}(FG) \mapsto FG.$$
Cette opération revient à retrouver les coefficients de FG à partir de ses valeurs en $a_0, \ldots, a_{n-1}$ (à supposer que cette opération d'interpolation soit possible, voir ci-dessous).

En termes matriciels, l'opération $F \mapsto \mathsf{Ev}(F)$ est linéaire, et sa matrice (pour des polynômes F de degré au plus $n-1$, dans la base monomiale $\{1, X, \ldots, X^{n-1}\}$) est la matrice de Vandermonde
$$V_{a_0,\ldots,a_{n-1}} = \begin{bmatrix} 1 & a_0 & \cdots & a_0^{n-1} \\ \vdots & & & \vdots \\ 1 & a_{n-1} & \cdots & a_{n-1}^{n-1} \end{bmatrix}.$$
Pour l'instant nous n'avons pas réellement progressé ; il reste la question importante de trouver des jeux de valeurs $a_0, \ldots, a_{n-1}$ pour lesquels les opérations d'évaluation et d'interpolation sont possibles (*i. e.* pour lesquels $V_{a_0,\ldots,a_{n-1}}$ est inversible), et réalisables rapidement. Pour ce faire, nous allons choisir des *racines de l'unité*.

On dit que $\omega \in A$ est une racine n-ième de l'unité si $\omega^n = 1$; ω est une racine n-ième *primitive* de l'unité si en plus $\omega^t - 1$ est non diviseur de zéro dans A pour tout diviseur strict t de n (c'est-à-dire que $\alpha(\omega^t - 1) = 0$ implique $\alpha = 0$).

Remarquons que si A est un corps, cette dernière condition revient simplement à dire que w^t est différent de 1 pour tout diviseur strict t de n. Par exemple, dans $\mathbb{C}$, -1 n'est pas une racine 4-ième primitive de l'unité, et i l'est.

À un tel ω, on associe la suite des points d'évaluation $1, \omega, \ldots, \omega^{n-1}$, et la matrice de Vandermonde associée sera notée V_ω. Un premier intérêt de ce choix apparaît dans le théorème suivant.

Théorème 2. *Soit $\omega \in A$ une racine n-ième primitive de l'unité. Alors $\omega^{-1} = \omega^{n-1}$ est également une racine n-ième primitive de l'unité, et $V_{\omega^{-1}} V_\omega = nI_n$.*

Exercice 10. Prouver le théorème précédent.

Autrement dit, l'interpolation sur une racine primitive de l'unité se ramène à une évaluation sur une racine primitive (dite *conjuguée*). Ceci explique qu'on se consacre donc maintenant à ce dernier problème uniquement, que l'on appelle traditionnellement DFT (Discrete Fourier Transform).

Supposons que n est pair, $n = 2k$; on peut alors regrouper les puissances de ω en puissances paires et impaires, d'une part $1, \omega^2, (\omega^2)^2, \ldots, (\omega^2)^{k-1}$ et d'autre part $\omega, \omega\cdot\omega^2, \omega\cdot(\omega^2)^2, \ldots, \omega\cdot(\omega^2)^{k-1}$. On va utiliser ce groupement pour mettre en place un algorithme de type « diviser pour régner ».

Pour appliquer cette idée, écrivons les divisions euclidiennes

$$F = Q_0(X^k - 1) + R_0 \quad \text{et} \quad F = Q_1(X^k + 1) + R_1,$$

avec $\deg R_0 < k$ et $\deg R_1 < k$. Les racines de $X^k - 1$ sont précisément les puissances paires de ω, et les racines de $X^k + 1$ les puissances impaires. Pour prouver ce point, il suffit d'utiliser l'hypothèse que $\omega^k - 1$ n'est pas un diviseur de zéro. Soit ensuite $\bar{R}_1(X) = R_1(\omega X)$. On a alors les égalités :

$$F(\omega^{2\ell}) = R_0((\omega^2)^\ell) \quad \text{et} \quad F(\omega^{2\ell+1}) = R_1(\omega^{2\ell+1}) = \bar{R}_1((\omega^2)^\ell).$$

Il convient de remarquer que R_0 et R_1 s'obtiennent très facilement à partir de F : pour $0 \leq \ell < k$, le coefficient en X^ℓ de R_0 est la somme des coefficients en X^k et $X^{k+\ell}$ de F ; le coefficient en X^ℓ de R_1 est la différence des coefficients en X^k et $X^{k+\ell}$ de F. Ensuite, passer de R_1 à $\bar{R}_1$ s'effectue en multipliant pour chaque ℓ le coefficient de X^ℓ de R_1 par ω^ℓ.

On dispose maintenant de tous les éléments pour écrire l'algorithme de DFT. Pour mettre en place les idées ci-dessus de manière récursive, il est plus simple de supposer que n est une puissance de 2 ; on obtient alors l'algorithme suivant :

Transformée de Fourier Discrète (DFT_ω)

Entrée : $F = f_0 + \cdots + f_{n-1}X^{n-1}$, et les puissances $1, \omega, \cdots, \omega^{n-1}$ d'une racine n-ième primitive de l'unité, n étant une puissance de 2.

Sortie : $F(1), \ldots, F(\omega^{n-1})$.

1. Si $n = 1$, renvoyer f_0.
2. Calculer R_0, R_1 et $\bar{R}_1$.
3. Soit $k = n/2$. Calculer récursivement $R_0(1), R_0(\omega^2), \ldots, R_0((\omega^2)^{k-1})$ et $\bar{R}_1(1), \bar{R}_1(\omega^2), \ldots, \bar{R}_1((\omega^2)^{k-1})$.
4. Renvoyer $R_0(1), \bar{R}_1(1), R_0(\omega^2), \bar{R}_1(\omega^2), \ldots, R_0((\omega^2)^{k-1}), \bar{R}_1((\omega^2)^{k-1})$.

La complexité de cet algorithme fait l'objet du théorème suivant.

THÉORÈME 3. *L'algorithme* DFT_ω *ci-dessus requiert au plus* $\frac{3n}{2}\log n$ *opérations dans* A.

DÉMONSTRATION. Supposant connues les puissances de ω, le coût de l'appel en degré n est d'au plus $2 \times n/2$ additions et soustractions (pour le calcul de R_0 et R_1), et $n/2$ multiplications (pour le calcul de $\bar{R}_1$), plus 2 appels récursifs en degré $n/2$. Sa complexité $T(n)$ satisfait donc à la récurrence :

$$T(n) \leq \frac{3n}{2} + 2T\left(\frac{n}{2}\right)$$

et le lemme « diviser pour régner » permet de conclure. □

EXERCICE 11. Montrer que l'algorithme DFT_ω ci-dessus requiert $n\log n$ additions dans A et $\frac{1}{2}n\log n$ multiplications d'éléments de A par des puissances de ω.

L'algorithme de transformée de Fourier discrète permet de multiplier les polynômes à coefficients dans A, pourvu que A contienne « suffisamment » de racines primitives de l'unité, d'ordre « suffisamment » élevé.

De manière explicite, si A contient une racine de l'unité ω d'ordre $n = 2^k$, on obtient l'algorithme suivant. Cette algorithme requiert l'inversibilité de n dans A, il ne fonctionne donc que sous l'hypothèse supplémentaire que 2 est inversible dans A.

Multiplication par Transformée de Fourier Rapide (FFT)

Entrée : F et G de degrés au plus $n/2-1$ et ω, une racine primitive n-ième de l'unité, n étant une puissance de 2.

Sortie : FG.

1. Calculer les n premières puissances de ω (qui donnent également les puissances de ω^{-1}, à permutation près).
2. Calculer $\mathsf{DFT}_\omega(F)$ et $\mathsf{DFT}_\omega(G)$.
3. Effectuer les multiplications de ces valeurs point-à-point ; on définit $(h_0, \ldots, h_{n-1})$ le vecteur résultat et H le polynôme $h_0 + \cdots + h_{n-1}X^{n-1}$.
4. Calculer $(k_0, \ldots, k_{n-1}) = \mathsf{DFT}_{\omega^{-1}}(H)$.
5. Renvoyer le polynôme $\frac{k_0}{n} + \cdots + \frac{k_{n-1}}{n}X^{n-1}$.

Remarquons que, en pratique, on manipule uniquement des tableaux d'éléments de A. La présentation du pseudo-code ci-dessus, qui fait explicitement intervenir le polynôme H, est trompeuse de ce point de vue. Par ailleurs, on a tout intérêt à stocker les valeurs des puissances de ω d'un appel à l'autre.

La complexité de cet algorithme est de 3 DFT en degré n, soit $\frac{9}{2}n\log n$ opérations, plus $O(n)$ divisions par n et $O(n)$ multiplications pour calculer les puissances de ω. Le coût pour la multiplication de polynômes de degré au plus $n-1$ s'en déduit aisément.

COROLLAIRE 2. *Soit $n = 2^k$, et supposons que 2 est inversible dans A et qu'il existe dans A une racine $2n$-ième primitive de l'unité. Cette racine étant connue, on peut multiplier les polynômes de degré au plus $n-1$ en $9n\log n + O(n)$ opérations dans A.*

EXERCICE 12. Montrer que sous les hypothèses du corollaire précédent, on peut multiplier les polynômes de degré au plus $n-1$ en utilisant $6n\log n + O(n)$ additions dans A, $3n\log n + O(n)$ multiplications par des puissances de ω, $2n$ multiplications arbitraires dans A et $2n$ divisions par $2n$.

EXERCICE 13. Soit n dans $\mathbb{N}$, soit n_0 la plus petite puissance de 2 supérieure ou égale à n, et supposons qu'il existe dans A une racine $2n_0$-ième primitive de l'unité. Cette racine étant connue, on peut multiplier les polynômes de degré au plus $n-1$ en $18n\log n + O(n)$ opérations dans A.

EXERCICE 14. Soit $n = 2^k$, et supposons qu'on dispose d'une racine n-ième primitive de l'unité $\omega \in A$. Soit P et Q deux polynômes dans $A[X]$ de degré au plus $n-1$. Supposons que les coefficients de $X^0, X^1, \ldots, X^{n-1}$ du produit $R = PQ$ sont connus. Montrer que R peut être calculé en $\frac{9}{2}n\log n + O(n)$ opérations dans A.

Même dans le cas favorable où A est un corps, il n'y existe pas nécessairement toutes les racines primitives de l'unité que l'on souhaite. Dans le cas particulier des corps finis, on sait donner une réponse précise à cette question d'existence.

THÉORÈME 4. *Soient $\mathbb{F}_q$ le corps fini à q éléments et $n \in \mathbb{N}$. Le corps $\mathbb{F}_q$ contient une racine n-ième primitive de l'unité si et seulement si n divise $q-1$.*

EXERCICE 15.
1. Prouver le théorème précédent.
2. Montrer que si n divise $q-1$ et si α est un élément primitif de $\mathbb{F}_q$ (*i. e.* tel que α engendre le group multiplicatif $(\mathbb{F}_q \setminus \{0\}, \times)$) alors $\alpha^{(p-1)/n}$ est une racine n-ième primitive de l'unité.

Ce résultat mène à la notion de *premier de Fourier*, qui sont les nombres premiers p tels que $p-1$ soit divisible par une grande puissance de 2, c'est-à-dire des nombres premiers de la forme $\ell 2^k + 1$, avec k « sufisamment grand ». Par exemple,

$$4179340454199820289 = 29 \times 2^{57} + 1$$

est un tel nombre premier. Ainsi, dans $\mathbb{Z}/4179340454199820289\,\mathbb{Z}$, on dispose de racines primitives 2^{57}-ièmes de l'unité (21 en est une) ; on peut donc y multiplier des polynômes de degrés colossaux par l'algorithme en $O(n \log n)$.

Nous donnons dans la Figure 1 la courbe de complexité pratique[2] de la multiplication polynomiale à coefficients dans le corps fini $A = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

L'allure de cette courbe confirme que les estimations théoriques de complexité sont respectées en pratique. Le comportement quasi-linéaire *par morceaux*, agrémenté de sauts entre les puissances successives de 2, est typique pour les implantations actuelles de la FFT.

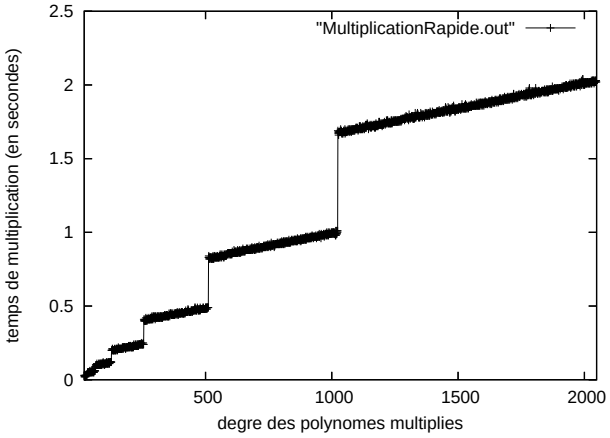


FIG. 1. Courbe de complexité pratique de la multiplication dans $A[X]$, où $A = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

[2]Calculs effectués avec le logiciel Magma, version V2.12-1, sur un Opteron 150 (2,4 GHz)

5. L'algorithme de Schönhage et Strassen

Quand les racines de l'unité font défaut, il reste possible de faire fonctionner les idées à base de transformée de Fourier. Ceci est réalisé par l'algorithme de Schönhage et Strassen, qui fait l'objet de cette section. Cet algorithme s'applique quel que soit l'anneau de base, pourvu que 2 y soit inversible ; l'idée est de rajouter les racines de l'unité qui manquent en étendant l'anneau de base de manière judicieuse.

Soient F et G des polynômes de degré strictement inférieur à $n/2$. L'algorithme a en fait pour vocation de calculer le produit FG modulo X^n+1 ; ce n'est pas une limitation ici, car le produit a un degré inférieur à n, et cela a l'avantage d'assurer la cohérence dans les appels récursifs.

La première idée est de réécrire F et G comme des polynômes $\bar{F}$ et $\bar{G}$ en deux variables X, Y, de degré strictement inférieur à $d = \sqrt{n}$ en chacune des deux variables. Par exemple, si $n = 9$, on voit le polynôme $a_0 + a_1X + \cdots + a_8X^8$ comme $(a_0+a_1X+a_2X^2)+(a_3+a_4X+a_5X^2)Y+(a_6+a_7X+a_8X^2)Y^2$. Savoir multiplier les polynômes dans cette représentation est suffisant, puisque $FG = (\bar{F}\bar{G})(X, X^d)$, et cette dernière évaluation se fait en complexité linéaire. Ici, la restriction de calculer FG modulo X^n+1 se traduit par le fait qu'on calcule $\bar{F}\bar{G}$ modulo Y^d+1.

Ensuite, on pose $B = A[X]/(X^{2d}+1)$ et on va multiplier $\bar{F}$ et $\bar{G}$ dans $B[Y]$: cela permet de retrouver $\bar{F}\bar{G} \mod (Y^d+1)$ dans $A[X, Y]$ puisque tous les coefficients du produit ont un degré en X strictement plus petit que $2d$.

Dans B, $\omega = X^2$ est une racine primitive de l'unité d'ordre $2d$. On peut donc appliquer l'algorithme de FFT afin de multiplier des polynômes de degré inférieurs strictement à d dans $B[Y]$ en $O(d \log d)$ opérations dans B. En regardant de près l'algorithme précédent, on voit que celui-ci nécessite :

- $O(d \log d)$ opérations $(+, -)$ dans B (chacune demande $O(d)$ opérations dans A) ;
- $O(d)$ divisions par d dans B (chacune demande $O(d)$ opérations dans A) ;
- $O(d \log d)$ multiplications par une puissance de ω dans B (chacune demande $O(d)$ opérations dans A, car elle revient simplement à décaler les indices et éventuellement changer un signe) ;
- d produits dans B, qui sont gérés par des appels récursifs.

On en déduit que la coût $T(n)$ obéit à la récurrence :

$$T(n) \leq Cn \log n + \sqrt{n} T(2\sqrt{n}),$$

où C est une constante universelle indépendante de n, bien sûr, mais aussi de A.

Exercice 16. Montrer que si α, β, γ sont des constantes positives telles que $T(n) \leq \alpha n \log n + \beta\sqrt{n} T(\gamma\sqrt{n})$, pour tout $n \geq 2$, alors

1. $T(n) \in O(n \log n)$, si $\beta\gamma < 2$;
2. $T(n) \in O(n \log n \log\log n)$, si $\beta\gamma = 2$;
3. $T(n) \in O(n \log^{\beta\gamma} n)$, si $\beta\gamma > 2$.

En admettant le résultat de l'exercice prédédent, nous pouvons donc établir le résultat suivant.

Théorème 5. *Soit A un anneau dans lequel 2 est inversible (d'inverse connu). On peut multiplier des polynômes de $A[X]$ de degré au plus n en $O(n \log n \log\log n)$ opérations $(+, -, \times)$ dans A.*

Il est possible d'étendre cette idée au cas où 3 est inversible (FFT triadique), et plus généralement à un anneau quelconque. On obtient alors l'algorithme de complexité $O(n \log n \log\log n)$ mentionné dans l'introduction.

6. Algorithmes pour les entiers

Les algorithmes ainsi que les résultats présentés ci-dessus s'étendent à la multiplication des entiers[3]. Nous allons brièvement présenter cette problématique à travers un exemple. Soient à multiplier les entiers 2087271 et 1721967, qu'on suppose donnés en base 2,

$$A = 111111101100101100111 \quad \text{et} \quad B = 110100100011001101111,$$

chacun ayant $D = 21$ chiffres binaires. On peut ramener leur multiplication à un produit de polynômes. Plus exactement, on associe à A et B les polynômes de degré strictement inférieur à D :

$$P = X^{20}+X^{19}+X^{18}+X^{17}+X^{16}+X^{15}+X^{14}+X^{12}+X^{11}+X^8+X^6+X^5+X^2+X+1$$

et

$$Q = X^{20} + X^{19} + X^{17} + X^{14} + X^{10} + X^9 + X^6 + X^5 + X^3 + X^2 + X + 1.$$

La stratégie est la suivante : on calcule $R = PQ$ dans $\mathbb{Z}[X]$, et ensuite on évalue R en $X = 2$. Pour multiplier P et Q dans $\mathbb{Z}[X]$, il suffit d'effectuer leur produit dans $A[X] = \mathbb{Z}/p\,\mathbb{Z}[X]$, où p est un nombre premier tel que $2D > p > D$. Par le Théorème 5, cette multiplication polynomiale en degré D peut se faire en $O(D \log D \log\log D)$ opérations $(+, -, \times)$ dans $A = \mathbb{Z}/p\,\mathbb{Z}$.

Puisque chaque opération de A nécessite au plus $O(\log^2 D)$ opérations binaires, il s'ensuit que le coût binaire du calcul de R est de $O(D \log^3 D \log\log D)$. Ici $p = 23$ et R (écrit en base 2) vaut $R = X^{40} + 10X^{39} + 10X^{38} + 11X^{37} + 11X^{36} + 11X^{35} + 100X^{34} + 11X^{33} + 11X^{32} + 100X^{31} + 11X^{30} + 100X^{29} + 101X^{28} + 11X^{27} + 101X^{26} + 1000X^{25} + 101X^{24} + 101X^{23} + 1000X^{22} + 1001X^{21} + 1010X^{20} + 1000X^{19} + 111X^{18} + 1000X^{17} + 110X^{16} + 110X^{15} + 110X^{14} + 11X^{13} + 100X^{12} + 110X^{11} + 100X^{10} + 11X^9 + 100X^8 + 100X^7 + 100X^6 + 11X^5 + 10X^4 + 11X^3 + 11X^2 + 10X + 1$. Enfin, l'évaluation de R en 2 revient à gérer les retenues et cela peut se faire en un coût négligéable. Ici $AB = R(2) = 110100010011010111101101110110110101001$, ou encore, en écriture décimale $AB = 3594211782057$.

Une légère amélioration est possible si l'on choisit pour p un premier de Fermat supérieur à $2D$. Dans notre exemple, on peut prendre $p = 193$. En effet, il existe dans $A = \mathbb{Z}/193\,\mathbb{Z}$ une racine primitive $\omega = 125$, d'ordre 64, donc supérieur à $2D = 40$. Ce choix mène à un algorithme de complexité binaire $O(D \log^3 D)$.

Une idée alternative est de calculer PQ dans $\mathbb{Z}[X]$ en faisant ce calcul dans $\mathbb{C}[X]$. En estimant les erreurs numériques, on peut montrer qu'il suffit de calculer en précision fixe $O(\log^2 D)$. La complexité binaire est donc toujours $O(D \log^3 D)$ via cette approche.

On peut faire un peu mieux, en remplaçant la base 2 par une base B telle que $B \log B$ soit de l'ordre de $\log D$ et en appliquant l'un des raisonnements précédents ; on peut aboutir ainsi à un algorithme de complexité binaire $O(D \log^2 D)$. Qui plus est, en appelant récursivement l'algorithme pour multiplier les petits entiers, on peut descendre cette complexité à $O(D \log D \log\log D \log\log\log D \ldots)$.

[3]Historiquement, les algorithmes pour la multiplication des entiers ont été introduits *avant* leurs homologues polynomiaux, alors que ces derniers sont souvent bien plus simples à énoncer.

Le record est cependant detenu par la version entière, à base de FFT dans des anneaux de type $\mathbb{Z}/(2^{2^\kappa}+1)\mathbb{Z}$, de l'algorithme de Schönhage-Strassen, dont nous nous contentons de mentionner l'existence.

THÉORÈME 6. *On peut multiplier deux entiers de D chiffres binaires en utilisant $O(D \log D \log\log D)$ opérations binaires.*

7. Un concept important : les fonctions de multiplication

Un bon nombre des résultats de complexité donnés dans la suite du cours reposent sur la notion de *fonction de multiplication*. Une fonction $\mathsf{M} : \mathbb{N} \to \mathbb{N}$ sera dite fonction de multiplication (pour un anneau A) si :

- On peut multiplier les polynômes de $A[X]$ de degré au plus n en au plus $\mathsf{M}(n)$ opérations dans A ;
- M vérifie l'inégalité $\mathsf{M}(n+n') \geq \mathsf{M}(n) + \mathsf{M}(n')$.

Ainsi, on sait d'après ce qui précède que des fonctions de la forme $n^{\log_2 3}$ ou $n \log n$ sont (à des constantes près) des fonctions de multiplication (respectivement pour tous les anneaux, ou ceux qui permettent la transformée de Fourier). L'intérêt de cette notion est qu'elle permet d'énoncer des résultats de complexité indépendants du choix de l'algorithme utilisé pour multiplier les polynômes (même si parfois cela mène à des estimations légèrement trop pessimistes).

La seconde condition est utilisée pour écarter l'hypothèse d'une fonction qui croît trop lentement (si M est constante, elle ne vérifie pas cette condition ...). Concrètement, elle permettra d'établir que dans des algorithmes du type « inversion de série formelle par l'opérateur de Newton », le coût total est essentiellement celui de la dernière étape.

De la même manière, on est amenés à introduire la notion de *fonction de multiplication* pour les entiers. Une fonction $\mathsf{M}_\mathbb{Z} : \mathbb{N} \to \mathbb{N}$ est une fonction de multiplication pour les entiers si :

- On peut multiplier des entiers de n chiffres en $\mathsf{M}_\mathbb{Z}(n)$ opérations binaires.
- $\mathsf{M}_\mathbb{Z}$ vérifie l'inégalité $\mathsf{M}_\mathbb{Z}(n+n') \geq \mathsf{M}_\mathbb{Z}(n) + \mathsf{M}_\mathbb{Z}(n')$.

Ce concept est très proche de son analogue polynomial, et les contextes d'utilisation sont souvent les mêmes : on utilise la seconde condition pour contrôler les coûts de multiplication lors de l'analyse d'algorithmes récursifs.

EXERCICE 17. Soit A un anneau, soit $a \in A$ et soit P un polynôme de $A[X]$, de degré au plus n. On se propose de calculer le polynôme $Q(X) = P(X+a)$.

1. Donner un algorithme pour le calcul de Q et estimer sa complexité en termes de n ;
2. Montrer qu'il est possible de calculer $Q(X)$ en utilisant seulement $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ dans A.

EXERCICE 18. Soit A un anneau, soit $\kappa, n \in \mathbb{N}$ et soit P un polynôme de $A[X]$, de degré au plus n.

1. Donner un algorithme pour le calcul de $P(X)^\kappa$ et estimer sa complexité en fonction de κ et de n ;
2. Montrer qu'il est possible de calculer $P(X)^\kappa$ en utilisant seulement $O(\mathsf{M}(n\kappa))$ opérations $(+,-,\times)$ dans A.

Notes

Le mathématicien russe A. N. Kolmogorov avait conjecturé au début des années 1960 qu'il serait impossible de multiplier deux entiers de n chiffres en un coût binaire sous-quadratique. En 1962 cette conjecture fut infirmée par Karatsuba et Ofman [9]. Une généralisation de l'algorithme de [9] a été proposée quelques années plus tard par Toom [14] et Cook [3]. L'algorithme de Toom-Cook a une complexité binaire de $O(n\,32^{\sqrt{\log n}})$ pour multiplier deux entiers de taille binaire n ; ce résultat peut être importé dans le monde polynomial (voir Exercice 9 pour une version plus faible).

L'algorithme DFT a une longue histoire, voir par exemple [6, 8]. Il s'agit d'un progrès algorithmique très fameux : J. Dongarra et F. Sullivan [7] le placent parmi les dix algorithmes qui ont marqué le plus le développement des sciences de l'ingénieur du 20e siècle. L'article fondateur de Cooley et Tukey [5] est sans doute l'un des plus cités en informatique[4]. L'article [1] constitue une bonne référence pour le lecteur désireux de se familiariser avec la myriade de techniques de type FFT.

Une avancée importante a été la découverte de Schönhage et Strassen [13] du résultat équivalent pour la multiplication des nombres entiers. Pendant longtemps, on a cru que cet algorithme ne pourrait présenter qu'un intérêt purement théorique. À ce jour, la plupart des logiciels généralistes de calcul formel disposent d'une multiplication rapide d'entiers : Maple et Mathematica utilisent la bibliothèque GMP, Magma dispose de sa propre implantation.

L'analogue polynomial de l'algorithme de Schönhage-Strassen traité en Section 5 a été suggéré dans [11]. Le cas particulier de la caractéristique 2 est traité dans l'article [12]. Plus récemment, Cantor et Kaltofen [2] ont donné un algorithme qui multiplie des polynômes de degré n sur une algèbre A (non nécessairement commutative, ni associative) en $O(n \log n \log\log n)$ opérations dans A.

Un problème ouvert est de trouver un algorithme de multiplication polynomiale en complexité $O(n)$, ou de prouver qu'un tel algorithme n'existe pas. Morgenstern a donné une réponse partielle, en montrant que dans un modèle simplifié où $A = \mathbb{C}$ et où les seules opérations admises sont les additions et les multiplications par des nombres complexes de module inférieur à 1, au moins $\frac{1}{2} n \log n$ opérations sont *nécessaires* pour calculer une DFT en taille n. Concernant les entiers, Cook et Aanderaa [4] ont prouvé une borne inférieure de la forme $cn \log n/(\log\log n)^2$ pour la multiplication en taille binaire n sur une machine de Turing.

Une autre question importante est de savoir si les n coefficients du *produit court* $FG \bmod X^n$ de deux polynômes F et G de degré au plus n peuvent être calculés plus efficacement que l'ensemble des $2n$ coefficients du produit FG. L'article [10] apporte une réponse positive au cas où l'algorithme de multiplication est celui de Karatsuba. Par contre, on ne connaît aucun élément de réponse dans le cas où la multiplication polynomiale repose sur la DFT.

Une faiblesse de l'algorithme DFT est son comportement quasi-linéaire *par morceaux* (voir la Figure 1), dû aux sauts de complexité entre deux puissances successives de 2. Récemment, van der Hoeven [15] a donné un nouvel algorithme, appelé TFT (de *Truncated Fourier Transform* en anglais), qui coïncide avec la DFT si n est une puissance de 2 et ayant une courbe de complexité « plus lisse » que la DFT.

[4] Plus de 2000 citations, d'après la base bibliographique Science Citation Index (SCI ®).

Un nombre premier p de la forme $2^e k + 1$ est appelé nombre premier de Fourier d'exposant e. Ces premiers sont utiles pour la FFT sur des corps finis, voir Ex. 15. On peut montrer qu'il y a approximativement $(x/\log(x))/2^{e-1}$ premiers de Fourier d'exposant e inférieurs à x. Il s'ensuit qu'il y a au environ 130 corps finis de la forme $k = \mathbb{F}_p$ tel que p a la taille d'un mot machine (32 bits) et tel que dans k on puisse multiplier par FFT des polynômes de degré de l'ordre d'un million. Les racines de l'unité de $k = \mathbb{F}_p$ peuvent être calculées à partir des éléments primitifs, cf. Ex. 15. On peut montrer que si on choisit au hasard un élément de $\mathbb{F}_p$, la probabilité qu'il soit primitif est égale à $6/\pi^2$, soit plus de 0,6.

Bibliographie

[1] Bernstein (D. J.). – Multidigit multiplication for mathematicians. – Preprint, available from http://cr.yp.to/papers.html.

[2] Cantor (D. G.) and Kaltofen (E.). – On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, vol. 28, n° 7, 1991, pp. 693–701.

[3] Cook (S. A.). – *On the minimum computation time of functions*. – PhD thesis, Harvard, 1966.

[4] Cook (S. A.) and Aanderaa (S. O.). – On the minimum computation time of functions. *Transactions of the American Mathematical Society*, vol. 142, 1969, pp. 291–314.

[5] Cooley (James W.) and Tukey (John W.). – An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, vol. 19, 1965, pp. 297–301.

[6] Cooley (James W.) and Tukey (John W.). – On the origin and publication of the FFT paper. *Current Contents*, vol. 33, n° 51–52, 1993, pp. 8–9.

[7] Dongarra (J.) and Sullivan (F.). – Top Ten Algorithms. *Computing in Science & Engineering*, vol. 2, n° 1, 2000.

[8] Duhamel (P.) and Vetterli (M.). – Fast Fourier transforms : a tutorial review and a state of the art. *Signal Processing. An Interdisciplinary Journal*, vol. 19, n° 4, 1990, pp. 259–299.

[9] Karatsuba (A.) and Offman (Y.). – Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, vol. 7, 1963, pp. 595–596.

[10] Mulders (Thom). – On short multiplications and divisions. *Applicable Algebra in Engineering, Communication and Computing*, vol. 11, n° 1, 2000, pp. 69–88.

[11] Nussbaumer (Henri J.). – Fast polynomial transform algorithms for digital convolution. *Institute of Electrical and Electronics Engineers. Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n° 2, 1980, pp. 205–215.

[12] Schönhage (A.). – Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Numerische Mathematik*, vol. 20, 1973, pp. 409–417.

[13] Schönhage (A.) and Strassen (V.). – Schnelle Multiplikation großer Zahlen. *Computing*, vol. 7, 1971, pp. 281–292.

[14] Toom (A. L.). – The complexity of a scheme of functional elements simulating the multiplication of integers. *Doklady Akademii Nauk SSSR*, vol. 150, 1963, pp. 496–498.

[15] van der Hoeven (Joris). – The truncated Fourier transform and applications. In *ISSAC 2004*, pp. 290–296. – ACM, New York, 2004.

COURS 3

Calculs rapides sur les séries

Résumé

La multiplication rapide de séries vue au Cours 2 permet des calculs efficaces comme l'inversion, l'exponentiation ou la prise de logarithme d'une série. Ces opérations sont effectuées grâce à une version formelle de la méthode de Newton. Elles entraînent une algorithmique efficace sur les polynômes. La composition de séries est en générale plus coûteuse que le produit, mais peut aussi tirer parti d'une multiplication rapide.

Ce cours porte sur le calcul des premiers termes de développements en séries. Pour fixer les notations, N sera utilisé pour représenter le nombre de termes à calculer, et « série » sera employé pour « série tronquée à l'ordre N ». Ainsi, « calculer une série » voudra toujours dire en calculer les N premiers termes. Comme dans le cours précédent, $\mathsf{M}(N)$ dénote une borne supérieure sur le nombre d'opérations arithmétiques nécessaires pour multiplier deux polynômes de degré au plus N. L'efficacité des algorithmes sera mesurée par leurs complexités arithmétiques.

1. Séries formelles

Si $\mathbb{A}$ est un anneau, on note $\mathbb{A}[[X]]$ l'anneau des séries formelles sur $\mathbb{A}$. Ses éléments sont des suites $(a_i)_{i\in\mathbb{N}}$ de $\mathbb{A}$, notées

$$\sum_{i\geq 0} a_i X^i.$$

Les opérations de $\mathbb{A}[[X]]$ sont l'addition des suites et une multiplication qui généralise la multiplication des polynômes :

$$\sum_{i\geq 0} a_i X^i \times \sum_{i\geq 0} b_i X^i = \sum_{i\geq 0} c_i X^i, \qquad \text{avec} \quad c_i = \sum_{j+k=i} a_j b_k,$$

la dernière somme étant finie.

Exercice 1. Vérifier que ces deux opérations font de $\mathbb{A}[[X]]$ un anneau. Quels en sont les éléments inversibles ?

Lorsque les coefficients sont des nombres complexes, la question de la convergence des séries entières représentées par ces séries formelles ne se posera pas pour les algorithmes considérés dans ce cours. En revanche, il est possible de définir une distance entre deux séries f et g par $d(f,g) = 2^{-\operatorname{val}(f-g)}$, où la *valuation* $\operatorname{val}(f)$ d'une série f est l'indice de son premier terme non-nul (et par convention $\operatorname{val}(0) = \infty$).

Exercice 2. Vérifier que la fonction d définie ci-dessus est bien une distance. Prouver que muni de cette distance, l'anneau des séries formelles forme un espace métrique complet (les suites de Cauchy convergent).

Comme $\mathbb{A}[[X]]$ est un anneau, il peut être utilisé comme anneau de base pour définir des séries formelles en une autre variable y, ce qui définit de la même manière l'anneau des séries formelles bivariées $\mathbb{A}[[X]][[Y]]$, noté aussi $\mathbb{A}[[X, Y]]$.

Algorithmiquement, on ne manipule pas de série à précision « infinie », mais seulement des troncatures, c'est-à-dire un certain nombre des premiers termes. Les séries tronquées deviennent alors de simples polynômes, pour lesquels on dispose en particulier d'algorithmes rapides de multiplication. Étant donnés les N premiers termes d'une série f, l'objectif de ce cours est de donner des algorithmes permettant de calculer efficacement les N premiers termes d'autres séries définies à partir de f.

Pour étendre la notation utilisée avec les polynômes, étant donnée la série

$$S = \sum_{i \geq 0} a_i X^i,$$

on notera $S \bmod X^N$ le polynôme

$$S \bmod X^N := \sum_{0 \leq i < N} a_i X^i.$$

On notera $S = f + O(X^N)$ si les séries ou polynômes S et f coïncident jusqu'au terme de degré N.

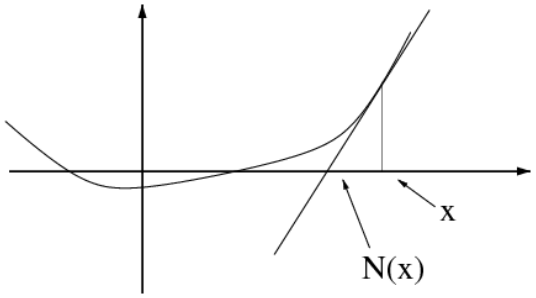
2. La méthode de Newton

L'opérateur de Newton est très largement utilisé en calcul numérique, pour trouver des solutions approchées d'équations. C'est un processus itératif, qui consiste à chaque étape à remplacer le système que l'on veut résoudre par son linéarisé au voisinage de la dernière approximation trouvée. L'adaptation de cette méthode dans un cadre formel est extrêmement fructueuse : elle permet de trouver les solutions séries de grandes variétés d'équations, avec un très bon comportement vis-à-vis de la complexité.

2.1. Version numérique. Soit f une fonction $\mathbb{R} \to \mathbb{R}$. Pour résoudre $f(g) = 0$ dans $\mathbb{R}$, on choisit $g_0 \in \mathbb{R}$ et on itère selon

$$g_{k+1} = N(g_k) = g_k - \frac{f(g_k)}{f'(g_k)}.$$

Graphiquement, l'itération correspond à la figure suivante :



Opérateur de Newton de $\mathbb{R}$ dans $\mathbb{R}$.

La solution serait exacte si f était une fonction linéaire ; dans le cas général, si g_0 est bien choisi (par exemple assez proche d'une racine simple), la suite g_k converge vers une limite g_∞. En outre, à l'itération k la distance à la limite est de l'ordre du carré de cette distance à l'itération $k - 1$. On parle dans ce cas de

convergence *quadratique*. Dans ces conditions, pour k suffisamment grand, le nombre de décimales correctes est approximativement *doublé* à chaque itération.

2.2. Version formelle. L'idée de l'itération de Newton s'adapte au calcul sur les séries, et la convergence reste quadratique, ce qui veut dire dans ce contexte que le nombre de termes corrects double à chaque itération.

THÉORÈME 1 (Itération de Newton sur les séries). *Soit $\Phi(X,Y)$ une série bivariée en X et $Y-\lambda$, où λ est une constante telle que $\Phi(0,\lambda)=0$ et $\frac{\partial\Phi}{\partial Y}(0,\lambda)$ est inversible dans $\mathbb{A}$. Alors, l'équation*

$$\Phi(X,Y)=0$$

admet une série y solution telle que $y(0)=\lambda$. L'itération

$$y_{k+1}=y_k-\frac{\Phi(X,y_k)}{\frac{\partial\Phi}{\partial Y}(X,y_k)} \bmod X^{2^{k+1}}$$

avec $y_0=\lambda$ converge vers cette solution avec $y-y_k=O(X^{2^k})$.

Avant de prouver ce théorème, il faut observer qu'en pratique il est souhaitable de programmer cette itération de manière récursive en partant de la précision N souhaitée et en divisant par 2 la précision à chaque étape. Si N est juste au-dessus d'une puissance de 2, le gain, quoique d'un facteur constant seulement, est appréciable.

La convergence mentionnée dans ce théorème est relative à la métrique des séries formelles.

DÉMONSTRATION. La preuve consiste à montrer par récurrence sur k que d'une part $\Phi(X,y_k)=O(X^{2^k})$ et d'autre part $y_k-y_{k+1}=O(X^{2^k})$ (et donc aussi que $y_k(0)=\lambda$).

Ces deux conditions permettent de conclure. En effet, la seconde condition montre que les y_k forment une suite de Cauchy. La limite de cette suite existe donc. Notons la y. En faisant tendre k vers l'infini dans

$$\Phi(X,y_k)=\Phi(X,y)+(y_k-y)\frac{\partial\Phi}{\partial Y}(X,y)+O((y_k-y)^2),$$

on voit que la première condition entraîne $\Phi(X,y)=0$. Ceci prouve qu'il existe une solution dans l'anneau des séries formelles, et que cette solution est y. Cette même équation donne alors

$$(y_k-y)\frac{\partial\Phi}{\partial Y}(X,y)+O((y_k-y)^2)=O(X^{2^k}),$$

dont se déduit ensuite $y-y_k=O(X^{2^k})$: comme $y_k(0)=\lambda$ pour tout k, $y(0)$ a la même valeur et donc d'une part $\mathrm{val}((y_k-y)^2)>\mathrm{val}(y_k-y)$ et d'autre part $\mathrm{val}(\frac{\partial\Phi}{\partial Y}(X,y))=0$ d'après l'hypothèse $\frac{\partial\Phi}{\partial Y}(0,\lambda)$ inversible.

Voici maintenant la preuve de la récurrence mentionnée ci-dessus. Pour $k=0$ la première condition est une conséquence de $\Phi(0,\lambda)=0$ et la seconde de

$$y_0-y_1=\frac{\Phi(X,\lambda)}{\frac{\partial\Phi}{\partial Y}(X,\lambda)}+O(X^2),$$

le dénominateur ayant un terme constant inversible et le numérateur valuation au moins 1. Pour passer de k à $k+1$, on utilise d'abord un développement de Taylor qui nous donne

$$\Phi(X, y_{k+1}) = \Phi(X, y_k) - (y_k - y_{k+1})\frac{\partial \Phi}{\partial Y}(X, y_k) + O(X^{2^{k+1}}).$$

Ensuite, la définition de y_{k+2} permet de conclure :

$$y_{k+1} - y_{k+2} = \frac{\Phi(X, y_{k+1})}{\frac{\partial \Phi}{\partial Y}(X, y_{k+1})} \bmod X^{2^{k+2}} = O(X^{2^{k+1}}).$$

□

3. Opérations quasi-optimales

Les applications les plus directes de la méthode de Newton sont résumées dans le théorème suivant.

THÉORÈME 2. *Soit f une série de $\mathbb{A}[[X]]$. On peut calculer en $O(\mathsf{M}(N))$ opérations les N premiers termes de*

1. *$1/f$, lorsque $f(0)$ est inversible ;*
2. *$\log f$ et f^α lorsque $f(0) = 1$;*
3. *$\exp(f)$ lorsque $f(0) = 0$.*

Pour les points 2 et 3, nous supposons aussi que $2, 3, \ldots, N-1$ sont inversibles dans $\mathbb{A}$.

3.1. Inversion de séries. L'inversion de séries donne une illustration de l'usage du théorème 1. Si g est une série avec terme constant inversible, il est aisé de vérifier que la série $\Phi(X, Y) = g - 1/Y$ vérifie les conditions du théorème avec $\lambda = 1/g(0)$. Il y a donc convergence quadratique de l'itération

$$y_k = y_{k-1} + y_{k-1}(1 - g y_{k-1}) \bmod X^{2^k} \tag{1}$$

vers $1/g$ avec $y_0 = 1/g(0)$.

Il nous reste maintenant à estimer la complexité de cet algorithme.

PROPOSITION 1. *Soit g dans $\mathbb{A}[[X]]$ avec $g(0)$ inversible et $k \geq 0$. On peut calculer l'inverse de g modulo X^{2^k} en $4\mathsf{M}(2^k) + O(2^k)$ opérations dans $\mathbb{A}$.*

Autrement dit, le coût total du calcul coïncide avec celui de la dernière étape, à une constante près.

DÉMONSTRATION. L'itération (1) demande 2 multiplications modulo X^{2^k}, soit $2\mathsf{M}(2^k)$ opérations, plus $C \cdot 2^k$ opérations supplémentaires (additions, ...), C étant une constante que l'on ne va pas déterminer. Notons $\mathsf{N}(k)$ le coût du calcul modulo 2^k ; on a alors, avec $\mathsf{N}(0) = 0$:

$$\mathsf{N}(k) \leq \mathsf{N}(k-1) + 2\mathsf{M}(2^k) + C2^k.$$

L'application du lemme « Diviser pour régner » du Cours 1 donne donc

$$\mathsf{N}(k) \leq 4\mathsf{M}(2^k) + C2^{k+1},$$

en utilisant l'inégalité $\mathsf{M}(d) \leq \frac{1}{2}\mathsf{M}(2d)$.

EXERCICE 3. Effectuer l'application du lemme.

□

REMARQUE. On peut montrer, en se donnant davantage de peine, que le résultat s'étend pour calculer l'inverse de G modulo n'importe quel X^ℓ (ℓ n'étant pas nécessairement une puissance de 2).

Par ailleurs, le coût ci-dessus est largement surestimé.

- En regardant de plus près le calcul de y_k, on constate que le deuxième produit est pris entre y_{k-1} et $1-y_{k-1}g$. Ce dernier terme est de la forme $X^{2^{k-1}}R_{k-1}$, alors que y_{k-1} est de degré au plus 2^{k-1}. Ainsi, seul le produit $y_{k-1}R_{k-1}$ est nécessaire, et celui-ci ne demande que $\mathsf{M}(2^{k-1})$ opérations.
- Le produit $y_{k-1}g$ est de la forme $1 + X^{2^{k-1}}R_{k-1}$. Nous sommes donc dans la situation on l'on multiplie un polynôme de degré 2^{k-1} (ici, y_{k-1}) par un polynôme de degré 2^k (ici, g), et où l'on connaît les 2^{k-1} premiers coefficients du produit. Dans cette situation, il est possible de descendre le coût du produit de $\mathsf{M}(2^k)$ à $\mathsf{M}(2^{k-1})$, en utilisant des techniques non triviales de « transposition » de code [4] qui seront abordées au Cours 9.

Au total, on peut donc gagner un facteur 2 sur l'estimation du théorème précédent (et cela se répercute sur le coût de la division euclidienne ci-dessous).

3.2. Logarithme. Pour calculer la série $\log f$ avec $f(0) = 1$, il suffit d'utiliser l'identité

$$\log f = \int \frac{f'}{f}.$$

Le calcul demande une division de séries, une dérivation et une intégration, mais ces deux dernières opérations sont linéaires. Le bilan est donc une complexité en $O(\mathsf{M}(N))$.

3.3. Exponentielle. Pour calculer la série $\exp(f)$ avec $f(0) = 0$, l'idée est d'appliquer une méthode de Newton avec

$$\Phi(Y) = f - \log Y.$$

EXERCICE 4. Montrer que les conditions du Théorème 1 sont vérifiées.

Il s'ensuit une convergence quadratique vers $\exp(f)$ pour l'itération

$$y_{k+1} = y_k + y_k(f - \log y_k).$$

Chaque itération demande donc le calcul d'une multiplication et d'un logarithme, d'où la complexité en $O(\mathsf{M}(N))$.

3.4. Puissance. La série f^α se déduit des précédentes par

$$f^\alpha = \exp(\alpha \log f).$$

Une application remarquable de cette trivialité apparaît dans la section suivante sur l'inverse compositionnel.

EXERCICE 5. Donner une itération de Newton qui calcule directement la racine carrée, sans passer par l'exponentielle et le logarithme. Plus difficile, estimer le gain par rapport à l'algorithme général de calcul de puissance.

EXERCICE 6. Donner des bornes sur les constantes dans les $O(\cdot)$ pour le logarithme, l'exponentielle et la puissance.

3.5. Sommes de Newton. Une conséquence intéressante de l'efficacité du calcul de l'exponentielle est l'utilisation efficace des sommes de Newton comme structure de données.

Si $P \in \mathbb{K}[X]$ est un polynôme de degré d, avec $\mathbb{K}$ est un corps, alors P a d racines $\alpha_1, \ldots, \alpha_d$ dans la clôture algébrique $\overline{\mathbb{K}}$ de $\mathbb{K}$. Les *sommes de Newton* de P sont les sommes $p_i = \alpha_1^i + \cdots + \alpha_d^i \in \mathbb{K}$. Leur utilisation efficace repose sur la proposition suivante.

PROPOSITION 2. *Les sommes de Newton $p_1, \ldots, p_d$ d'un polynôme $P \in \mathbb{K}[X]$ de degré d peuvent être calculées en $O(\mathsf{M}(d))$ opérations dans $\mathbb{K}$. Lorsque la caractéristique de $\mathbb{K}$ est 0 ou supérieure à d, la conversion inverse est également possible en $O(\mathsf{M}(d))$ opérations.*

DÉMONSTRATION. La décomposition en éléments simples

$$S = \frac{XP'}{P} = \sum_{P(\alpha)=0} \frac{X}{X-\alpha} = \sum_{\substack{P(\alpha)=0 \\ i \geq 0}} \alpha^i X^{-i}$$

montre que les coefficients du développement en puissances de X^{-1} de XP'/P sont les p_i. Le calcul des d premiers termes de cette série demande $O(\mathsf{M}(d))$ opérations. L'opération inverse, à savoir le calcul de P à partir des d premiers termes de la série $S = XP'/P$ est effectué en $O(\mathsf{M}(d))$ opérations par la formule

$$P = \exp \int S/X$$

si $P(0) \neq 0$. Sinon, il faut d'abord récupérer la valuation, et le reste se calcule de la façon précédente. La contrainte sur la caractéristique permet de définir l'exponentielle tronquée par son développement en série. □

3.6. ⋆ Application à la somme et au produit composés ⋆. On suppose dans cette section que $\mathbb{K}$ est un corps de caractéristique nulle ou supérieure à d^2.

THÉORÈME 3. *Soient P et Q deux polynômes unitaires de $\mathbb{K}[X]$, de degré d. Alors, les polynômes*

$$P \oplus Q = \prod_{\substack{P(\alpha)=0 \\ Q(\beta)=0}} X - (\alpha + \beta), \qquad P \otimes Q = \prod_{\substack{P(\alpha)=0 \\ Q(\beta)=0}} X - (\alpha\beta)$$

peuvent être calculés en $O(\mathsf{M}(d^2))$ opérations.

Comme ces polynômes sont de degré d^2, il s'agit là encore d'une complexité quasi-optimale. Dans cet énoncé, les polynômes sont définis à l'aide des racines de P et Q dans une clôture algébrique, mais leurs coefficients sont dans $\mathbb{K}$. Il est également possible de les définir sur un anneau quelconque comme des *résultants* bivariés. Les résultants et leur calcul seront vus au cours 10. Dans le cas bivarié général, il n'existe pas encore d'algorithme quasi-optimal et le résultat ci-dessus est l'un des rares cas particuliers qui se traite efficacement.

Ces polynômes $P \oplus Q$ et $P \otimes Q$ sont très utiles en pratique. Les polynômes constituent une structure de données bien commode pour coder leurs racines et calculer avec. Ces polynômes $P \oplus Q$ et $P \otimes Q$ fournissent l'addition et la multiplication avec cette structure de données.

DÉMONSTRATION. Le produit de sommes de Newton est la somme de Newton du produit :

$$\sum_{\alpha} \alpha^i \sum_{\beta} \beta^i = \sum_{\alpha,\beta} (\alpha\beta)^i.$$

Il suffit donc de multiplier *terme à terme* les séries XP'/P et XQ'/Q pour obtenir la série génératrice des sommes de Newton de $P \otimes Q$. Si l'on a calculé d^2 termes de ces séries, le résultant, dont le degré est d^2, est alors reconstruit par une exponentielle en $O(\mathsf{M}(d^2))$ opérations.

Diviser le i^{e} coefficient de XP'/P par $i!$, pour tout $i \geq 0$, produit la série

$$\sum_{\substack{P(\alpha)=0 \\ i \geq 0}} \frac{\alpha^i X^{-i}}{i!} = \sum_{P(\alpha)=0} \exp(\alpha/X).$$

En effectuant la même opération sur Q et en effectuant le produit des séries, on obtient donc

$$\sum_{P(\alpha)=0} \exp(\alpha/X) \sum_{Q(\beta)=0} \exp(\beta/X) = \sum_{\substack{P(\alpha)=0 \\ Q(\beta)=0}} \exp((\alpha+\beta)/X).$$

Il suffit alors de remultiplier le i^{e} coefficient par $i!$ pour obtenir la série génératrice des sommes de Newton de $P \otimes Q$, qui est reconstruit par une exponentielle. L'ensemble des opérations est borné par $O(\mathsf{M}(d^2))$. □

3.7. ⋆ Inverse compositionnel ⋆. Étant donnée une série f avec $f(0) = 0$ et $f'(0)$ inversible, il s'agit ici de calculer une série $f^{(-1)}$ telle que $f(f^{(-1)}(X)) = f^{(-1)}(f(X)) = X$.

3.7.1. *Les N premiers termes.* Les conditions d'application de la méthode de Newton sont vérifiées pour la fonction

$$\Phi(X, Y) = f(Y) - X,$$

avec $\lambda = 1/f'(0)$.

EXERCICE 7. Vérifier les conditions.

L'itération correspondante est donc

$$y_{k+1} = y_k - \frac{f(y_k) - X}{f'(y_k)} \bmod X^{2^{k+1}}.$$

En dérivant $f(Y) - X = 0$, on voit que cette itération se simplifie en

$$y_{k+1} = y_k - y'_k(f(y_k) - X) \bmod X^{2^{k+1}}.$$

Le coût est alors dominé soit par celui du produit, soit plus généralement par celui de la composition par f.

EXERCICE 8. La k^{e} racine positive de l'équation $x = \tan x$ admet un développement asymptotique de la forme

$$r_k = (2k+1)\frac{\pi}{2} + \frac{a_1}{k} + \frac{a_2}{k^2} + \dots.$$

Pour tout i, le coefficient a_i est un polynôme de $1/\pi\mathbb{Q}[1/\pi^2]$ de degré i. Borner le nombre d'opérations dans $\mathbb{Q}$ nécessaires pour calculer $a_1, \dots, a_N$, pour N grand.

3.7.2. *Le N^e terme.* Bien qu'en général le coût de l'inverse fonctionnel soit dominé par celui de la composition et donc en $O(\sqrt{N \log N}\mathsf{M}(N))$ (voir la section suivante), il est possible d'obtenir le N^{e} terme sans calculer les précédents pour $O(\mathsf{M}(N))$ opérations. L'idée repose sur la *formule d'inversion de Lagrange* :

$$[X^N]f^{(-1)}(X) = \frac{1}{N}[X^{N-1}]\frac{1}{(f(X)/X)^N},$$

où la notation $[X^k]f$ représente le coefficient de X^k dans la série f. Ainsi, le calcul par cette formule ne requiert qu'une puissance et donc peut être effectué en $O(\mathsf{M}(N))$ opérations.

3.7.3. *Exemple.* La série génératrice des arbres généraux étiquetés (appelés aussi *arbres de Cayley*) vérifie l'équation

$$y = X\exp(y).$$

Son développement à l'origine est obtenu en $O(\mathsf{M}(N))$ opérations dans $\mathbb{Q}$ en calculant d'abord celui de $y\exp(-y)$ par les méthodes de la section précédente, puis en inversant ce développement par la méthode présentée ci-dessus.

Pour déterminer le comportement asymptotique des coefficients ainsi calculés lorsque N tend vers l'infini (c'est-à-dire l'asymptotique du nombre d'arbres de taille N), une méthode présentée dans le cours d'analyse d'algorithmes passe par le calcul du développement de y au voisinage du point singulier $\exp(-1)$, où y vaut 1. En posant $u = \sqrt{2}\sqrt{1-eX}$, on voit qu'il s'agit alors de calculer le développement de y solution de

$$\sqrt{2 - 2y\exp(1-y)} = u,$$

au voisinage de $y = 1$ et $u = 0$, équation à laquelle la méthode présentée dans cette section s'applique pour donner le résultat en $O(\mathsf{M}(N))$ opérations.

Remarque. La formule d'inversion de Lagrange donne immédiatement une jolie expression pour les coefficients de y solution de $y = X\exp(y)$ de l'exemple précédent. L'asymptotique mentionné plus haut donne alors la formule de Stirling efficacement.

4. La composition des séries

Si $f(X)$ et $g(X)$ sont des séries, avec $g(0) = 0$, il s'agit de calculer efficacement la série $f(g(X))$. Il s'avère que les algorithmes connus n'atteignent pas une complexité quasi-optimale (c'est-à-dire en $O(N)$ à des facteurs logarithmiques près).

4.1. Méthode naïve. La composition peut être calculée par la méthode de Horner. Si $f(X) = \sum_{i\geq 0} a_i X^i + O(X^N)$, le calcul utilise alors

$$f(g(X)) = a_0 + g(a_1 + g(a_2 + \dots)) + O(X^N).$$

Exercice 9. Montrer que la complexité du calcul par cette formule est $O(N\mathsf{M}(N))$.

4.2. ⋆ Pas de bébés—pas de géants ⋆. Une technique dite *pas de bébés, pas de géants* réduit ce coût à $O(\sqrt{N}(\mathsf{M}(N) + N^{\omega/2}))$, où ω est l'exposant de la complexité du produit de matrices, sur lequel reviendra le Cours 7. La série f est d'abord écrite

$$f(X) = f_0(X) + X^k f_1(X) + \cdots + X^{\lceil N/k\rceil} f_{\lceil N/k\rceil}(X) + O(X^N),$$

où les polynômes f_i en nombre $1+\lceil N/k\rceil$ comportent chacun k termes. Ensuite, on calcule les puissances $g^2,\dots,g^k$ de g en $k\mathsf{M}(N)$ opérations. Soient $f_{i,j}$ les coefficients des f_i et $g_{i,j}$ ceux des g^i. Le développement

$$f_i(g)=\sum_{j=0}^{N-1}\left(\sum_{\ell=0}^{k-1} f_{i,\ell}g_{\ell,j}\right)x^j+O(x^N)$$

montre que les coefficients des $f_i(g)$ sont les coefficients du produit de la matrice $(f_{i,j})$ de taille $(\lceil N/k\rceil+1)\times k$ par la matrice $(g_{i,j})$ de taille $k\times N$. En découpant chacune de ces matrices en blocs de taille $k\times k$, ce produit est effectué en au plus $O(N^2k^{\omega-3})$ opérations. Ensuite, il ne reste plus qu'à effectuer $\lceil N/k\rceil$ produits à précision N suivis d'autant d'additions pour un coût total de $O(N\mathsf{M}(N)/k)$. Le choix $k=\lfloor\sqrt{N}\rfloor$ mène à la complexité annoncée.

4.3. L'algorithme de Brent & Kung. Bien qu'il n'y ait pas d'algorithme permettant d'abaisser la complexité au même ordre que $\mathsf{M}(N)$, il est possible de faire sensiblement mieux que la méthode naïve grâce à un algorithme sophistiqué dû à Brent & Kung, détaillé dans cette section.

THÉORÈME 4. *Si $2,3,\dots,\lceil\sqrt{N\log N}\rceil$ sont inversibles dans l'anneau $\mathbb{A}$, la série $f(g(X))$ peut être calculée en $O(\sqrt{N\log N}\mathsf{M}(N))$ opérations arithmétiques, ou en $O(\sqrt{N}\mathsf{M}(N))$ opérations arithmétiques si $\log\mathsf{M}(N)/\log N\to\gamma>1$.*

L'idée de départ de l'algorithme permettant d'aboutir à cette complexité est d'utiliser la formule de développement de Taylor, sous la forme

$$f(g_1+X^mg_2)=f(g_1)+f'(g_1)X^mg_2+f''(g_1)\frac{X^{2m}g_2^2}{2!}+\cdots, \tag{2}$$

où $g=g_1+X^mg_2$ et g_1 est un polynôme de degré inférieur à m.

Le premier gain de complexité par rapport à la méthode naïve provient de l'observation suivante.

LEMME 1. *Étant données les séries g et $f\circ g$, avec $g'(0)$ inversible, la série $f'\circ g$ peut être calculée en $O(\mathsf{M}(N))$ opérations arithmétiques.*

DÉMONSTRATION. La dérivation de séries a une complexité linéaire, et la division est en $O(\mathsf{M}(N))$. Le résultat provient alors de la formule de dérivation d'une composée :

$$f'\circ g=\frac{(f\circ g)'}{g'}.$$

L'hypothèse sur $g'(0)$ permet de garantir l'inversion de la série du dénominateur. Elle est un peu plus forte que nécessaire : il suffit que le premier coefficient non nul de la série g' soit inversible. □

L'utilisation répétée de cette idée dans la formule (2) mène à une complexité

$$C(f(g_1))+\frac{N}{m}O(\mathsf{M}(N))\text{ opérations}$$

pour l'ensemble de la composition, où $C(f(g_1))$ représente la complexité du calcul de la première composition, donnée par le lemme suivant.

LEMME 2. *Soient f et g deux* polynômes *de degrés k et m, avec $g(0) = 0$. Le calcul des N premiers coefficients de la* série *$f \circ g$ peut être effectué en $O(km \log N\mathsf{M}(N)/N)$ opérations arithmétiques, ou en $O(km\mathsf{M}(N)/N)$ opérations si $\log \mathsf{M}(N)/\log N \to \gamma > 1$.*

DÉMONSTRATION. [du théorème à l'aide de ce lemme] Ce lemme donne $C(f(g_1)) = O(m\mathsf{M}(N)\log N)$. La complexité totale est donc bornée par

$$O(m\mathsf{M}(N)\log N) + \frac{N}{m}O(\mathsf{M}(N)) = O\left(\mathsf{M}(N)(m\log N + \frac{N}{m})\right).$$

Cette dernière somme est minimisée par le choix de $m = \sqrt{N/\log N}$ qui donne le résultat du théorème.

Le cas sans le facteur $\log N$ est laissé en exercice. □

DÉMONSTRATION. [du lemme] Sans perte de généralité, on peut supposer que le degré k de f est une puissance de 2. Il suffit de considérer sinon que les coefficients de f pour les degrés allant de $k+1$ jusqu'à la puissance de 2 immédiatement supérieure sont nuls, et la perte de complexité est d'au plus un facteur 2, absorbé dans la constante du $O(\cdot)$.

L'idée est d'effectuer ce calcul par « diviser pour régner » en récrivant le polynôme f sous la forme

$$f = f_1 + X^{k/2} f_2,$$

où f_1 et f_2 sont deux polynômes de degré au plus $k/2$. Dans la composition

$$f(g) = f_1(g) + g^{k/2} f_2(g),$$

les deux polynômes du second sommant ont degré au plus $km/2$. La complexité C_k^m découlant de l'utilisation de cette formule vérifie donc

$$C_k^m \leq 2C_{k/2}^m + 2\mathsf{M}(\min(N, km/2)) + O(N).$$

C'est dans cette prise de minimum que réside toute la subtilité : tant que k est grand, les calculs sont tronqués à l'ordre N, et dès que k devient suffisamment petit, on exploite l'arithmétique polynomiale. Il vient donc

$$\begin{aligned} C_k^m \leq & 2\mathsf{M}(N) + 4\mathsf{M}(N) + \cdots + 2^{\ell-1}\mathsf{M}(N) \\ & + 2^\ell \left(\mathsf{M}\left(\frac{km}{2^\ell}\right) + 2\mathsf{M}\left(\frac{km}{2^{\ell+1}}\right) + \cdots\right), \end{aligned}$$

où ℓ est déterminé par

$$\frac{km}{2^\ell} < N \leq \frac{km}{2^{\ell-1}}.$$

Cette valeur de ℓ, combinée à l'utilisation du lemme « Diviser pour régner » conclut la preuve du lemme. □

Exercices

EXERCICE 10 (Composition itérée). Soit F une série de la forme $F(x) = f_1 x + f_2 x^2 + \cdots$ à coefficients dans un corps $\mathbb{K}$. En notant $F^{[0]}(x) = x$, on définit la $q$ième itérée de F par

$$F^{[q]}(x) = F^{[q-1]}(F(x)). \tag{3}$$

L'objectif de cet exercice est d'étudier la complexité $C_q(N)$ du calcul des N premiers termes de cette série pour N grand en nombre d'opérations dans $\mathbb{K}$. On note $M(N)$

une borne sur le nombre d'opérations dans $\mathbb{K}$ nécessaires pour calculer le produit de deux polynômes de degré N dans $\mathbb{K}[x]$. On fait sur M les hypothèses habituelles de régularité.

La notation $g(q,N) = O(f(q,N))$ pour des fonctions de q et N voudra dire dans cet exercice qu'il existe une constante K telle que pour tout q, il existe un entier N_0 tel que pour tout $N > N_0$,

$$|g(q,N)| \leq K|f(q,N)|.$$

1. En n'utilisant que de l'algorithmique naïve, montrer que le calcul des N premiers termes de $F^{[q]}$ peut être effectué en $C_q(N) = O(qN^3)$ opérations dans $\mathbb{K}$.
2. En exploitant la composition rapide de séries, montrer que cette complexité peut être abaissée à $C_q(N) = O(q\sqrt{N \log N} M(N))$.
3. Décrire une idée simple permettant de réduire la dépendance en q pour aboutir à $C_q(N) = O(\log q \sqrt{N \log N} M(N))$.

Le reste de l'exercice étudie un algorithme permettant d'abaisser encore cette complexité pour finir à la même complexité que la composition avec F, indépendamment de q. L'idée de départ est que dans le cas de la puissance, le calcul de $F(x)^q$ peut être réalisé efficacement sous la forme $\exp(q \log F(x))$. L'objectif est de transposer cette technique à la composition.

On suppose maintenant que $f_1 \neq 0$ et que f_1 n'est pas une racine de l'unité.

On définit la série de Schroeder $S(x) = s_1 x + s_2 x^2 + \cdots$ par

$$S(F(x)) = f_1 S(x), \qquad s_1 = 1. \tag{4}$$

4. Montrer que cette équation définit bien une série $S(x)$, et que celle-ci est unique.
5. Montrer que pour tout entier q
$$F^{[q]}(x) = S^{[-1]}(f_1^q S(x)),$$
où $S^{[-1]}$ est l'inverse de S pour la composition. En déduire une borne sur la complexité du calcul de $F^{[q]}$ une fois la série S connue à précision N.
6. Pour calculer S solution de (4), on va s'intéresser à la résolution d'une équation plus générale :
$$A(x)Y(F(x)) - B(x)Y(x) - C(x) = 0 \bmod x^k, \tag{5}$$
où A, B, C, F sont des séries données et Y est l'inconnue. L'approche est une technique de diviser-pour-régner. En posant
$$Y(x) = U(x) + x^n V(x),$$
où U est un polynôme de degré inférieur à n, montrer que U et V sont solutions d'équations du même type que (5). Expliciter les coefficients de ces équations.
7. Décrire un algorithme de calcul des N premiers coefficients de Y en nombre d'opérations $O(\sqrt{N \log N} M(N))$. En notant a_0, b_0, c_0 les termes constants des séries A, B, C, on supposera que $a_0 f_1^m \neq b_0$, pour $m = 1, 2, 3, \ldots$, et que si $a_0 = b_0$, alors $c_0 = 0$.
8. Décrire enfin l'algorithme de calcul de $F^{[q]}$ dans la complexité annoncée.

9. À l'inverse, décrire un algorithme calculant une série G telle que $F = G^{[q]}$, dans la même complexité.

Notes

L'essentiel de ce cours provient de l'article exceptionnel [3] où ces techniques ont été étudiées du point de vue de la complexité pour la première fois. L'application aux sommes et produits composées est beaucoup plus récente et se trouve dans [2]. En ce qui concerne la composition, il est possible de descendre à une complexité quasi-optimale dans le cas où la caractéristique est finie [1]. La méthode de Newton permet également la résolution d'équations ou de systèmes différentiels. Ceci fera l'objet du cours 8. En pratique, les constantes cachées dans les $O(\cdot)$ jouent un grand rôle, et nécessitent généralement l'implantation à la fois des algorithmes asymptotiquement meilleurs et des algorithmes plus classiques, souvent plus efficaces en petite taille.

Bibliographie

[1] Bernstein (Daniel J.). – Composing power series over a finite ring in essentially linear time. *Journal of Symbolic Computation*, vol. 26, n° 3, 1998, pp. 339–341.

[2] Bostan (Alin), Flajolet (Philippe), Salvy (Bruno), and Schost (Éric). – Fast computation of special resultants. *Journal of Symbolic Computation*, vol. 41, n° 1, January 2006, pp. 1–29.

[3] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n° 4, 1978, pp. 581–595.

[4] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Applicable Algebra in Engineering, Communication and Computing*, vol. 14, n° 6, March 2004, pp. 415–438.

[5] Karp (A. H.) and Markstein (P.). – High-precision division and square root. *ACM Transactions on Mathematical Software*, vol. 23, n° 4, 1997, pp. 561–589.

COURS 4

Division euclidienne, fractions rationnelles et récurrences linéaires à coefficients constants

Résumé

La multiplication et l'inversion rapide de séries permettent le calcul efficace de la division euclidienne de polynômes et du développement en série des fractions rationnelles. Ils mènent en particulier au calcul rapide d'un ou de plusieurs termes d'une suite récurrente linéaire à coefficients constants.

1. Division de polynômes

Étant donnés deux polynômes F et G de $\mathbb{A}[X]$, avec G unitaire, à coefficients dans un anneau $\mathbb{A}$, il existe d'uniques polynômes Q et R, quotient et reste de la division euclidienne de F par G, tels que :

$$F = QG + R \quad \text{avec} \quad \deg R < \deg G.$$

Calculer rapidement le quotient Q et le reste R est d'importance vitale dans toute la suite du cours. Outre l'application aux suites récurrentes qui sera détaillée plus loin, les algorithmes de division seront utilisés dans le Cours 5 pour l'évaluation multipoint et l'interpolation et dans le Cours 10 pour le calcul de pgcd. À leur tour, ces deux algorithmes sont centraux dans nombre d'applications.

1.1. Méthode naïve. L'algorithme naïf pour calculer Q et R consiste à poser la division. Pour cela, les termes dominants sont d'abord isolés :

$$F = aX^m + T_F, \quad G = X^n + T_G, \qquad \text{avec} \quad \deg T_F < m, \quad \deg T_G < n.$$

Si $m < n$, il n'y a rien à faire. Sinon, la boucle élémentaire de la division de F par G consiste à « tuer » le terme de plus haut degré de F en effectuant la soustraction

$$F - aX^\delta G = (aX^m + T_F) - \left(aX^{\delta+n} + aX^\delta T_G\right) = T_F - aX^\delta T_G,$$

où $\delta = m - n$. Le polynôme obtenu est congru à F modulo G, mais de degré strictement inférieur à m. Il n'y a plus qu'à itérer ce procédé jusqu'à obtenir un polynôme de degré strictement inférieur à n, et à garder trace des quotients successifs.

La complexité de cet algorithme est facile à estimer. La boucle élémentaire présentée ci-dessus s'effectue en $O(n)$ opérations de type $(+, -, \times)$; dans le pire des cas, l'algorithme effectue $(m - n + 1)$ passages dans cette boucle, de sorte que la complexité de la division de F par G est de $O(n(m - n))$ opérations. Le pire des cas est atteint pour $m = 2n$. C'est donc un algorithme *quadratique* (mais un bon algorithme quadratique : la constante dans le $O(\cdot)$ est en fait petite, c'est 2). Le même genre d'estimation s'obtient pour la division euclidienne classique des entiers.

1.2. Algorithme rapide. Un bien meilleur résultat peut être obtenu à base d'itération de Newton.

THÉORÈME 1. *Soient G un polynôme unitaire de $\mathbb{A}[X]$ de degré n et $F \in \mathbb{A}[X]$ de degré $m \geq n$. Alors on peut calculer le quotient Q et le reste R de la division euclidienne de F par G en $5\mathsf{M}(m-n) + \mathsf{M}(n) + O(m)$ opérations $(+, -, \times)$ de $\mathbb{A}$.*

Ainsi, la division euclidienne ne coûte pas plus cher que la multiplication (à une constante près). En particulier, si on utilise une multiplication à base de FFT, le coût de la division est *linéaire* en le degré, à des facteurs logarithmiques près.

DÉMONSTRATION. L'idée principale de l'algorithme part de la réécriture de $F = QG + R$ en

$$\frac{F}{G} = Q + \frac{R}{G}.$$

Si on pense que $\mathbb{A} = \mathbb{R}$, on voit donc que le développement asymptotique de $\frac{F}{G}$ *à l'infini* a ses premiers termes donnés par Q, puisque $\frac{R}{G}$ tend vers 0 à l'infini (à cause des contraintes de degré sur R et G). Ceci suggère que l'on va obtenir Q par calcul du développement de Taylor de $\frac{F}{G}$ au voisinage de l'infini.

Concrètement, il suffit de ramener d'abord l'infini en zéro (par le changement de variable $X = 1/T$), pour réduire le calcul à la division de séries formelles. Plus précisément, le point de départ est l'identité

$$\frac{T^m F(1/T)}{T^n G(1/T)} = T^{m-n} Q(1/T) + \frac{T^m R(1/T)}{T^n G(1/T)}.$$

Dans cette identité les numérateurs et dénominateurs des fractions sont des polynômes, et le polynôme $T^n G(1/T)$ a 1 pour terme constant. En outre, la valuation du second sommant du membre droit est supérieure à $m - n$. En découle donc l'algorithme suivant :

1. Calculer $T^m F(1/T)$ et $T^n G(1/T)$ (aucune opération arithmétique n'est nécessaire, il suffit d'inverser l'ordre des coefficients) ;
2. Calculer le quotient $(T^m F(1/T))/(T^n G(1/T))$ mod T^{m-n+1} par une inversion de série formelle suivie d'un produit ;
3. en déduire Q en inversant l'ordre des coefficients ;
4. en déduire R, qui est donné par $R = F - QG \bmod X^n$.

Ces formules mènent au résultat de complexité du Théorème. D'après la Proposition 1 et la remarque qui la suit, l'inverse du dénominateur à l'étape 2 s'obtient en

$$4\mathsf{M}(m-n) + O(m-n)$$

opérations dans $\mathbb{A}$, puis une multiplication supplémentaire donne Q. Pour retrouver R, le dernier produit est effectué modulo X^n, c'est-à-dire pour un coût de $\mathsf{M}(n)$. Toutes les additions sont prises en compte dans le terme $O(m)$. □

REMARQUE. Si Q a un degré beaucoup plus petit que G, on peut accélérer le calcul de QG, en découpant G en « tranches » de taille $m - n$; de la sorte, le dernier produit peut se faire en $\frac{n}{m-n}\mathsf{M}(m-n)$ opérations. Enfin, si de nombreuses réductions sont à faire modulo le même polynôme G, il est évidemment recommandé de stocker l'inverse du réciproque de $T^n G(1/T)$ de G.

1.3. Le cas des entiers. Comme pour la multiplication, il est possible de poser le problème dans $\mathbb{Z}$ comme dans un anneau de polynômes. C'est ce dernier cas qui est le plus simple à étudier, puisqu'il n'y a pas à y gérer de retenue. On obtient cependant un résultat analogue sur les entiers.

THÉORÈME 2. *Soit $\mathsf{M}_{\mathbb{Z}}$ une fonction de multiplication pour $\mathbb{Z}$, et f et g deux entiers positifs avec $g \leq f \leq 2^n$. Soient q et r les quotient et reste de la division euclidienne de f par g :*

$$f = qg + r \quad \textit{avec} \quad 0 \leq r < g.$$

On peut calculer q et r en $O(\mathsf{M}_{\mathbb{Z}}(n))$ opérations binaires.

EXERCICE 1. Estimer la constante dans la complexité ci-dessus.

1.4. Application aux calculs modulaires. Une application importante de la division euclidienne rapide est le calcul efficace dans des structures algébriques de type « quotient », par exemple dans n'importe quel corps fini. Si $\mathbb{A}$ est un anneau et si P est un polynôme unitaire de $\mathbb{A}[X]$, il s'agit de calculer modulo P, c'est-à-dire de rendre effectives les opérations $(+, -, \times)$ dans $\mathbb{A}[X]/(P)$. L'addition dans $\mathbb{A}[X]/(P)$ est facile à opérer : on peut la faire terme à terme, ainsi la complexité est linéaire en le degré n de P. Ensuite, pour multiplier deux éléments $A + (P)$ et $B + (P)$ de $\mathbb{A}[X]/(P)$, on commence par multiplier A et B dans $\mathbb{A}[X]$, puis on réduit le résultat modulo P. D'après ce qui précède, la complexité du produit dans $\mathbb{A}[X]/(P)$ est en $O(\mathsf{M}(n))$.

La question du calcul dans $\mathbb{Z}/n\mathbb{Z}$ est légèrement plus délicate que son analogue dans $\mathbb{A}[X]$, à cause des retenues. Malgré tout, les résultats s'étendent.

2. Fractions rationnelles et récurrences à coefficients constants

Si $\mathbb{A}$ est un anneau, on note $\mathbb{A}(X)$ l'anneau des fractions rationnelles sur $\mathbb{A}$. Ses éléments sont de la forme $A(X)/B(X)$, avec $A, B \in \mathbb{A}[X]$ et $B \neq 0$. Les opérations de $\mathbb{A}(X)$ sont l'addition et la multiplication habituelle des fractions

$$\frac{A(X)}{B(X)} + \frac{C(X)}{D(X)} = \frac{A(X)D(X) + B(X)C(X)}{B(X)D(X)}, \quad \frac{A(X)}{B(X)} \times \frac{C(X)}{D(X)} = \frac{A(X)C(X)}{B(X)D(X)}.$$

Le degré d'une fraction rationnelle A/B est défini comme $\max(\deg(A), \deg(B))$. Deux fractions rationnelles de degré strictement inférieur à n peuvent donc être additionnées et multipliées en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.

Si, de plus, $B(0)$ est inversible dans $\mathbb{A}$, alors on peut voir la fraction rationnelle A/B comme une série de $\mathbb{A}[[X]]$, obtenue en divisant la série A par B. Le résultat $\sum_i c_i X^i$ de cette division sera appelé *le développement de Taylor* de $A(X)/B(X)$. On dit alors que $\sum_i c_i X^i$ est une *série rationnelle*. Les séries rationnelles forment un sous-anneau de $\mathbb{A}[[X]]$.

Mathématiquement, les fractions rationnelles et les séries rationnelles sont deux incarnations du même concept. Du point de vue algorithmique, ce dédoublement permet de coder une fraction rationnelle A/B de deux manières différentes, soit par le couple (A, B), soit par le développement en série tronqué de A/B. Les deux structures de données nous seront utiles dans ce cours, ainsi nous nous intéressons au passage de l'une à l'autre en bonne complexité.

EXERCICE 2. Montrer qu'une fraction rationnelle est uniquement déterminée par les $\deg(A) + \deg(B)$ premiers termes de son développement en série de Taylor.

On peut calculer le développement de Taylor à l'ordre N d'une fraction rationnelle de degré $d \leq N$ en $O(\mathsf{M}(N))$ opérations, en utilisant la méthode de Newton, décrite dans le Cours 3. Mais $O(\mathsf{M}(N))$ est en général beaucoup plus gros que les $O(dN)$ opérations requises par la méthode naïve. Une meilleure solution est fournie par la Théorème suivant. Le problème inverse, tout aussi fondamental, consiste à retrouver la forme rationnelle à partir des premiers termes de la série rationnelle, et sera traité efficacement dans le Cours 10.

THÉORÈME 3. *Soit $A(X)/B(X)$ dans $\mathbb{A}(X)$ de degré au plus d, avec $B(0)$ inversible. Le développement de Taylor de $A(X)/B(X)$ à précision $N \geq d$ peut se calculer en $O(N\mathsf{M}(d)/d)$ opérations $(+, -, \times)$ dans $\mathbb{A}$. Le N^e coefficient c_N peut se calculer en $O(\mathsf{M}(d)\log N)$ opérations dans $\mathbb{A}$.*

Si l'anneau $\mathbb{A}$ permet la FFT, ces estimations de complexité deviennent $O(N \log d)$ et $O(d \log d \log N)$; elles sont quasi-optimales, simultanément vis-à-vis de N et de d.

DÉMONSTRATION. Les deux assertions découlent du lemme suivant.

LEMME 1. *Soit $A(X)/B(X)$ dans $\mathbb{A}(X)$ avec $B(0)$ inversible. Soit d le degré de B et soit $\kappa \geq 0$. Alors les coefficients $c_\kappa, c_{\kappa+1}, \ldots, c_{\kappa+d-1}$ du développement de $A/B = \sum_i c_i X^i$ sont les coefficients de $X^{2d-2}, \ldots, X^d, X^{d-1}$ du produit*

$$\big(X^\kappa \bmod B(1/X)X^d\big)\big(c_{2d-2} + \ldots + c_0 X^{2d-2}\big).$$

En admettant ce lemme, il est aisé de démontrer le Théorème 3. En effet, pour calculer $c_0, \ldots, c_N$, il suffit d'appliquer le lemme $\lceil N/d \rceil$ fois en prenant $\kappa = 0, d, 2d, \ldots$, ce qui ramène le problème au calcul des restes de $X^{d\kappa}$ modulo $\bar{B} = B(1/X)X^d$. Ceci peut se faire en utilisant $O(N/d)$ opérations dans $\mathbb{B} = \mathbb{A}[X]/(\bar{B})$, en posant d'abord $y = x^d$, où x est l'image de X dans l'anneau quotient $\mathbb{B}$, et en calculant ensuite $y^2, y^3, \ldots$ par multiplications successives par y.

De même, pour calculer un seul terme c_N, on applique le lemme à $\kappa = N$ et tout revient à déterminer le reste de X^N modulo $\bar{B}$, donc au calcul de la N^{e} puissance de x dans $\mathbb{B}$. Or, cela peut se faire en $O(\log N)$ multiplications dans $\mathbb{B}$ en exploitant récursivement l'identité

$$x^\kappa = \begin{cases} (x^{\kappa/2})^2, & \text{si } \kappa \text{ est pair,} \\ x \cdot (x^{\frac{\kappa-1}{2}})^2, & \text{sinon.} \end{cases}$$

Comme chaque opération dans $\mathbb{B}$ coûte $O(\mathsf{M}(d))$ opérations dans $\mathbb{A}$, on obtient les estimations de la Proposition. Notons pour conclure que les $2d-1$ premiers coefficients $c_0, \ldots, c_{2d-2}$ de A/B se calculent par itération de Newton en $O(\mathsf{M}(d))$ opérations dans $\mathbb{A}$, ce qui représente dans les deux cas un coût négligeable. □

DÉMONSTRATION. [du Lemme] Partons de l'observation que la matrice de l'application $\mathbb{A}$-linéaire $X\cdot : \mathbb{B} \to \mathbb{B}$ dans la base canonique $\{1, X, \ldots, X^{d-1}\}$ est la matrice compagnon C telle que $[c_{n+1}, \ldots, c_{n+d}] = [c_n, \ldots, c_{n+d-1}] \cdot C$ pour tout $n \geq 0$. De cette dernière égalité il s'ensuit que $c_{\kappa+i}$ est égal au produit du vecteur $[c_i, \ldots, c_{d+i-1}]$ et de la première colonne de la matrice C^κ. Or, puisque C^κ est la matrice de multiplication par X^κ, les entrées de sa première colonne sont précisément les coefficients du polynôme $X^\kappa \bmod \bar{B}$. □

3. Suites récurrentes linéaires à coefficients constants

3.1. Prélude : les nombres de Fibonacci. La suite de Fibonacci, introduite vers 1202 par Leonardo Pisano[1] dans un problème récréatif décrivant la croissance d'une population de lapins, est définie par les conditions initiales $F_0 = 0, F_1 = 1$ et la récurrence

$$F_{n+2} = F_{n+1} + F_n, \quad n \geq 0. \tag{1}$$

La suite de Fibonacci jouit de riches propriétés algébriques, arithmétiques et combinatoires. Par exemple : (a) F_n est le nombre de façons différentes de paver un rectangle $2 \times (n-1)$ au moyen de dominos 2×1 ; (b) $(F_n)_n$ est une *suite de divisibilité, i. e.* F_n divise F_m dès lors que n divise m ; (c) si n est impair, alors

$$F_n = 2^{n-1} \prod_{k=1}^{\frac{n-1}{2}} \left(\frac{1}{4} + \cos^2 \frac{k\pi}{n} \right)$$

Exercice 3. Prouver les assertions (a)–(c).

Malgré sa simplicité, la suite de Fibonacci fait l'objet de nombreux problèmes ouverts. Par exemple, on ignore s'il existe une infinité de nombres de Fibonacci premiers. Les nombres de Fibonacci sont omniprésents en mathématiques et en informatique[2] : ils interviennent aussi bien dans l'analyse de l'algorithme d'Euclide pour le calcul du plus grand commun diviseur de deux entiers, que dans la solution négative de Matiyasevich du dixième problème de Hilbert[3].

Du point de vue algorithmique, on s'intéresse typiquement au calcul efficace d'un terme F_N de la suite de Fibonacci (pour N « grand »)[4] ou de ses N premiers termes. Nous traitons ces questions dans un cadre plus général.

3.2. Calcul rapide des termes d'une suite. Une suite $(a_n)_{n\geq 0}$ d'éléments de l'anneau $\mathbb{A}$ est appelée suite récurrente linéaire à coefficients constants (srlcc) d'ordre d si elle satisfait une récurrence de la forme

$$a_{n+d} = p_{d-1}a_{n+d-1} + \cdots + p_0 a_n, \qquad n \geq 0,$$

où les p_i sont des éléments de $\mathbb{A}$. Le polynôme $P = X^d - p_{d-1}X^{d-1} - \cdots - p_0$ de $\mathbb{A}[X]$ est appelé *polynôme caractéristique* de la suite $(a_n)_{n\geq 0}$.

Exercice 4. Soit M une matrice de taille $d \times d$ à coefficients dans $\mathbb{A}$, de polynôme caractéristique $\chi_M(X) = \det(XI_d - M)$, soit u une matrice ligne et v une matrice colonne. Montrer que la suite $(uM^n v)_{n\geq 0}$ est une suite récurrente linéaire à coefficients constants admettant χ_M comme polynôme caractéristique.

Exercice 5. Soit $a_1, \ldots, a_r$ des entiers strictement positifs. Soit A_n et B_n le nombre de r-uplets non-ordonnés, resp. ordonnés, $(x_1, \ldots, x_r) \in \mathbb{N}^r$, solutions de l'équation $a_1x_1 + \cdots + a_rx_r = n$. Montrer que les suites (A_n) et (B_n) sont récurrentes linéaires à coefficients constants, admettant $(X^{a_1} - 1) \cdots (X^{a_r} - 1)$ et $X^{a_1} + \cdots + X^{a_r} - 1$ comme polynômes caractéristiques.

---

[1] Mieux connu sous le pseudonyme de Fibonacci.

[2] Le journal *The Fibonacci Quarterly* est entièrement dédié à l'étude de ses propriétés.

[3] Ce problème proposait de trouver un algorithme pour décider si un système d'équations diophantiennes (polynômes à coefficients entiers) admet une solution en nombres entiers.

[4] On pourra admirer $F_{1\,000\,000}$ à l'url http://www.upl.cs.wisc.edu/~bethenco/fibo/

EXERCICE 6. Si $P \in \mathbb{A}[X]$ est de degré d, alors la suite $(P(n))_{n\geq 0}$ est une srlcc de polynôme caractéristique $(X-1)^{d+1}$.

3.2.1. *Exponentiation binaire.* L'approche naïve pour le calcul du N^{e} terme d'une srlcc consiste tout simplement à dérouler la récurrence et requiert $O(dN)$ opérations dans $\mathbb{A}$. Une alternative à cette méthode naïve est basée sur l'exponentiation binaire de la matrice compagnon associée à la suite. Par exemple, la récurrence (1) se récrit matriciellement

$$\begin{pmatrix} F_N \\ F_{N-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{C} \begin{pmatrix} F_{N-1} \\ F_{N-2} \end{pmatrix} = C^{N-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}, \qquad N \geq 1.$$

La puissance de la matrice constante C se calcule alors récursivement par

$$C^N = \begin{cases} (C^{N/2})^2, & \text{si } N \text{ est pair,} \\ C \cdot (C^{\frac{N-1}{2}})^2, & \text{sinon.} \end{cases}$$

On en déduit que le calcul de F_N peut être effectué sans calculer les précédents en $O(\log N)$ produits de matrices 2×2, soit $O(\log N)$ opérations dans $\mathbb{A}$.

EXERCICE 7. Montrer que le N^{e} terme de toute récurrence linéaire d'ordre d à coefficients constants peut se calculer en $O(d^3 \log N)$ opérations dans $\mathbb{A}$.

En utilisant des algorithmes plus évolués pour la multiplication matricielle, on peut abaisser cette complexité à $O(d^\omega \log N)$ opérations dans $\mathbb{A}$, où $2 \leq \omega < 2,38$ (voir Cours 7). La dépendance en d reste cependant plus que quadratique. La section suivante présente une meilleure méthode, de complexité quasi-linéaire en d.

3.2.2. *Exponentiation modulaire.* La complexité arithmétique de l'algorithme de la section précédente (Exercice 7) est quasi-optimale par rapport à l'indice N, mais pas par rapport à l'ordre d de la récurrence. Une méthode plus efficace, quasi-optimale à la fois en d et N, exploite le lien très étroit entre fractions rationnelles et suites récurrentes à coefficients linéaires. Ce lien est précisé dans le résultat suivant.

LEMME 2. *La série génératrice d'une srlcc est rationnelle. Plus exactement,*

$$\sum_{n\geq 0} a_n X^n = \frac{N(X)}{\bar{P}(X)},$$

où N_0 est un polynôme de degré $< d$ et $\bar{P}(X) = P(1/X)X^{\deg(P)}$ est le polynôme réciproque du polynôme caractéristique de la suite.

EXERCICE 8. Prouver ce lemme.

EXEMPLE 1. La série génératrice de la suite de Fibonacci est $\frac{X}{1-X-X^2}$.

Le lemme 3.2.2 montre donc qu'il y a équivalence entre le calcul des premiers termes de suites récurrentes linéaires et le développement en série de Taylor de fractions rationnelles. En vue du Théorème 3, cela entraîne le résultat suivant.

THÉORÈME 4. *Soit (a_n) une suite récurrente linéaire, donnée par une récurrence d'ordre d et des conditions initiales $a_0, \ldots, a_{d-1}$. Soit $N \geq d$. Alors, les N premiers termes $a_0, \ldots, a_N$ peuvent se calculer en $O(N\mathsf{M}(d)/d)$ opérations dans $\mathbb{A}$; le calcul du N^e terme peut se faire en seulement $O(\mathsf{M}(d)\log N)$ opérations dans $\mathbb{A}$.*

Exercice 9. Un polynôme $P \in \mathbb{A}[X]$ de degré d peut être évalué aux $N \gg d$ points $1, 2, \ldots, N$ en $O(N\,\mathsf{M}(d)/d)$ opérations dans $\mathbb{A}$.

3.3. Propriétés de clôture. La classe des srlcc admet de nombreuses propriétés de clôture : si $\mathbf{a} = (a_n)_n$ et $\mathbf{b} = (b_n)_n$ sont deux srlcc de polynômes caractéristiques P et Q, alors

1. la somme $\mathbf{a} + \mathbf{b} = (a_n + b_n)_n$ et le produit de Cauchy $\mathbf{a} \star_{\mathbf{C}} \mathbf{b}$, de terme général $\sum_{i=0}^{n} a_i b_{n-i}$, sont deux srlcc de polynôme caractéristique PQ ;
2. le produit de Hadamard $\mathbf{a} \star_{\mathbf{H}} \mathbf{b} = (a_n b_n)_n$ est une srlcc de polynôme caractéristique le produit composée $P \otimes Q$ définie au Cours 3 ;
3. la suite $\left(\sum_{i=0}^{n} \binom{n}{i} a_i b_{n-i}\right)_n$ est une srlcc de polynôme caractéristique la somme composée $P \oplus Q$.

Exercice 10. Prouver les assertions précédentes.

Exercice 11. Lorsque les coefficients de la récurrence sont des entiers, étudier la complexité binaire de tous les algorithmes traités dans ce cours.

3.4. Application : tests de primalité. Une application du calcul rapide d'un terme d'une récurrence est une famille de tests probabilites de primalité, de complexité polynomiale. L'idée est de construire une suite récurrente (a_n) d'entiers telle que la primalité de n soit équivalente (ou presque équivalente) à $a_n = 0 \bmod n$.

Un cas particulier important en est *le test de Fermat* (pour lequel $a_n = a^{n-1} - 1$) implanté dans la plupart des systèmes de calcul formel. Bien qu'il soit probabiliste (si n ne passe pas le test, n est composé, mais si n passe le test, alors il est premier seulement avec une grande probabilité), sa grande simplicité le rend souvent préférable à d'autres algorithmes sophistiqués.

Exercice 12. Soit (a_n) une srlcc d'ordre d. Montrer qu'il existe des constantes entières $c_0, c_1, \ldots, c_d$ telles que p divise $c_0 + c_1 a_{p-1} + \cdots + c_d a_{p-d}$ dès lors que p est un nombre premier. De plus, pour tout premier p, les constantes $c_i \bmod p$ peuvent être trouvées en $O(\mathsf{M}(d))$ opérations arithmétiques dans $\mathbb{A} = \mathbb{Z}/p\mathbb{Z}$.

Par exemple, si (F_n) est la suite de Fibonacci, alors p divise $F_{p-2} + 3F_{p-1} - 1$ dès lors que p est premier et la réciproque est vraie avec une bonne probabilité. L'exercice précédent fournit un test de primalité similaire au test de Fermat. D'après le Théorème 4 son coût est de $O(\mathsf{M}(d) \log p)$ opérations arithmétiques dans $\mathbb{Z}/p\mathbb{Z}$, soit $O(\mathsf{M}(d)\mathsf{M}_{\mathbb{Z}}(\log p) \log p)$ opérations binaires.

Exercice 13. Soient a et N deux entiers premiers entre eux. Montrer que N est premier *si et seulement si* $X^N + a = (X + a)^N \bmod N$ dans $\mathbb{Z}[X]$.

Exercice 14. Montrer que si N est premier, $0 \leq a < N$ et $P(X) \in \mathbb{Z}[X]$, alors $X^N + a = (X + a)^N \bmod P(X)$ dans $\mathbb{Z}/N\mathbb{Z}[X]$; si de plus, $P(X)$ est de degré $r = O(\log^c N)$, pour un $c > 0$, alors cette égalité peut être testée « en temps polynomial », c'est-à-dire en un nombre d'opérations binaires polynomial en $\log N$.

Notes

Historiquement, le premier algorithme rapide pour la division des polynômes est dû à Moenck et Borodin [7]. Son point clef est que *le quotient de la division euclidienne de deux polynômes ne dépend que de leurs coefficients de poids fort.* Cette remarque sera également à la base du calcul rapide de pgcd, étudié au Cours 10.

L'algorithme de la Section 1.2 est dû à Strassen [10]. Une alternative, de même complexité asymptotique, à l'algorithme esquissé en Section 1.4 pour les calculs modulaires a été proposée par Montgomery [8].

Calculer les N premiers termes d'une srlcc de polynôme caractéristique fixé est une opération linéaire en les conditions initiales ; c'est le *dual* de l'opération de division d'un polynôme de degré N par un polynôme fixé. Les conséquences algorithmiques de ce fait seront décrites dans le Cours 32.

Le théorème de Skolem-Mahler affirme que pour toute srlcc (a_n), l'ensemble de ses zéros (les indices i pour lesquels $a_i = 0$) est la réunion d'un ensemble fini et d'un nombre fini de suites arithmétiques. Son étude est une question subtile. Par exemple, déterminer si l'ensemble des zéros est vide est un problème NP-dur [2]. Les srlcc sont étudiées dans [3, 11]. Le livre [4] constitue une référence très complete.

L'exercice 4 est le point clef de la fameuse méthode de Wiedemann pour la résolution de systèmes linéaires creux, qui sera traitée dans le Cours 16.

Le calcul rapide d'un terme de la suite de Fibonacci par exponentiation binaire de la matrice compagnon associée relève du folklore mathématique. Sa généralisation (Exercice 7) est décrite dans [6], mais était probablement connue bien avant.

Les Théorèmes 3 et 4 sont tirés de [5] et [9]. Leur récente généralisation au cas des matrices polynomiales est à la base du meilleur algorithme pour la résolution de systèmes linéaires à coefficients polynomiaux, qui sera exposé dans le Cours 9.

Il n'existe pas de tests *déterministes* de primalité basés sur le test modulaire d'un terme d'une récurrence à coefficients constants. Par contre, on peut caractériser un nombre premier N à l'aide de suites qui vérifient des récurrences à coefficients polynomiaux (comme la factorielle, via le test de Wilson $(N-1)! = -1 \bmod N$). Malheureusement, cela ne fournit pas d'algorithme efficace. Le premier algorithme déterministe qui prouve la primalité en temps polynomial est très récent [1]. Cet article part de la caractérisation de type Fermat donnée en Exercice 14, et exhibe une constante c et un polynôme P tels que la primalité de N est impliquée par la vérification de l'identité de l'Exercice 14 pour seulement $r^{1/2}\log(N)$ valeurs de a.

Bibliographie

[1] Agrawal (Manindra), Kayal (Neeraj), and Saxena (Nitin). – PRIMES is in P. *Annals of Mathematics. Second Series*, vol. 160, n° 2, 2004, pp. 781–793.

[2] Blondel (Vincent D.) and Portier (Natacha). – The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, vol. 351/352, 2002, pp. 91–98. – Fourth special issue on linear systems and control.

[3] Cerlienco (L.), Mignotte (M.), and Piras (F.). – Suites récurrentes linéaires. Propriétés algébriques et arithmétiques. *L'Enseignement Mathématique*, vol. XXXIII, 1987, pp. 67–108. – Fascicule 1-2.

[4] Everest (Graham), van der Poorten (Alf), Shparlinski (Igor), and Ward (Thomas). – *Recurrence sequences*. – American Mathematical Society, Providence, RI, 2003, *Mathematical Surveys and Monographs*, vol. 104, xiv+318p.

[5] Fiduccia (C. M.). – An efficient formula for linear recurrences. *SIAM Journal on Computing*, vol. 14, n° 1, 1985, pp. 106–112.

[6] Miller (J. C. P.) and Brown (D. J. Spencer). – An algorithm for evaluation of remote terms in a linear recurrence sequence. *Computer Journal*, vol. 9, 1966, pp. 188–190.

[7] Moenck (R. T.) and Borodin (A.). – Fast modular transforms via division. *Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972)*, 1972, pp. 90–96.

[8] Montgomery (Peter L.). – Modular multiplication without trial division. *Mathematics of Computation*, vol. 44, n° 170, 1985, pp. 519–521.

[9] Shoup (V.). – A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *Proceedings of ISSAC'91*. pp. 14–21. – ACM Press, 1991.

[10] Strassen (V.). – Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik*, vol. 20, 1972/73, pp. 238–251.

[11] van der Poorten (A. J.). – Some facts that should be better known, especially about rational functions. In *Number theory and applications*, pp. 497–528. – Kluwer, Dordrecht, 1989. Proceedings of a Conference held at Banff, AB, 1988.

COURS 5

Calculs modulaires, évaluation et interpolation

Résumé

Nous introduisons la notion d'algorithme modulaire. Nous nous intéressons aux cas particuliers importants de l'évaluation multipoint et de l'interpolation, pour lesquels nous donnons des algorithmes rapides. Ces algorithmes peuvent être vus comme des généralisations de la FFT.

1. Introduction

Un ingrédient essentiel en calcul formel est l'utilisation de différents types de représentations pour les objets manipulés. Par exemple, un polynôme est classiquement codé par la liste de ses coefficients, mais on peut très bien le représenter par les valeurs qu'il prend en un nombre suffisant de points. D'ailleurs, nous avons vu au Cours 2 que c'est bien cette seconde représentation qui est la plus adaptée pour le calcul efficace, via la FFT, du produit de polynômes. Par ailleurs, d'autres opérations (comme la division polynomiale, traitée dans le Cours 3) se font mieux dans la première représentation. D'autres exemples de codages alternatifs déjà rencontrés dans ce cours sont l'écriture des entiers en différentes bases de numération, ou encore la représentation des suites récurrentes linéaires à coefficients constants, soit par leurs éléments, soit par leurs séries génératrices, soit par une récurrence et des conditions initiales.

L'exemple de la FFT montre à quel point il est crucial d'examiner dans quelle représentation un problème donné est plus facile à traiter, et aussi, de trouver des algorithmes rapides pour la conversion d'une représentation à l'autre.

Une incarnation de ce concept général est la notion d'*algorithme modulaire*, dont le paradigme *évaluation-interpolation* est un cas particulier très important. L'approche modulaire consiste à choisir des *moduli* m_i, à faire des calculs modulo chaque m_i et à reconstruire tout à la fin le résultat à l'aide d'une version effective du *théorème des restes chinois*. Pour assurer l'unicité du résultat et la correction de ce schéma, il faut que les moduli soient suffisamment *nombreux et indépendants*. Typiquement, s'il s'agit d'entiers, ils doivent être choisis premiers entre eux et tels que leur produit dépasse le résultat final. En particulier, pour mettre en place une approche modulaire, on a besoin de bornes *a priori* sur la taille de l'objet calculé.

Cette approche porte ses fruits surtout lorsque les coefficients dans les calculs intermédiaires sont beaucoup plus gros que ceux du résultat final. Il s'agit du phénomène de *l'explosion des expressions intermédiaires*, qui se présente par exemple dans le calcul du déterminant d'une matrice entière ou polynomiale, ou encore au cours du calcul du pgcd de polynômes à coefficients entiers.

Exemple 1 (calcul du déterminant d'une matrice polynomiale). Soit à calculer le déterminant d'une matrice $n \times n$, aux entrées polynômes de degrés au plus d (le cas particulier $d = 1$ correspond au calcul d'un polynôme caractéristique). Le point de départ est la remarque que le pivot de Gauss produit sur la i^{e} ligne des fractions rationnelles de degré $2^i d$. Ainsi, sa complexité ne semble pas polynomiale par rapport à n. Une approche évaluation-interpolation résout cette anomalie. Le déterminant étant un polynôme de degré au plus nd, il suffit de choisir un ensemble de $nd+1$ points, d'y évaluer les entrées de la matrice, de calculer les $nd+1$ déterminants de matrices scalaires et de finir par une interpolation fournissant le determinant cherché. Le même raisonnement s'applique aux matrices entières.

Choix des moduli. Dans certaines situations, le programmeur a le choix des moduli. Dans l'exemple précédent, on peut choisir comme moduli des polynômes de la forme $X - \omega^i$, dans le cas favorable où l'anneau de base contient une racine primitive de l'unité ω d'ordre suffisamment élevé ($\approx 2dn$). En théorie ce choix est donc possible dès lors que l'anneau de base permet une multiplication polynomiale à base de FFT. En pratique, il existe des plages de degrés pour lesquels, même si elle est faisable, la FFT n'est pas encore rentable et pour lesquels ce choix est déconseillé. Dans d'autres situations, le choix des moduli est intrinsèquement imposé par le problème à traiter ; c'est le cas pour le calcul de la factorielle exposé au Cours 6.

2. Présentation, résultats

Les problèmes d'évaluation et d'interpolation sont inverses l'un de l'autre. Dans leur version standard, ils s'énoncent ainsi. Soient $a_0, \ldots, a_{n-1}$ des points dans $\mathbb{A}$, où $\mathbb{A}$ est un anneau commutatif et unitaire.

Évaluation. Étant donné un polynôme P dans $\mathbb{A}[X]$, de degré strictement inférieur à n, calculer les valeurs :

$$P(a_0), \ldots, P(a_{n-1}).$$

Interpolation. Étant donnés $b_0, \ldots, b_{n-1} \in \mathbb{A}$, trouver un polynôme $P \in \mathbb{A}[X]$ de degré strictement inférieur à n tel que

$$P(a_0) = b_0, \ldots, P(a_{n-1}) = b_{n-1}.$$

Le problème d'interpolation admet toujours une solution unique sous l'hypothèse technique suivante :

(H) $\qquad a_i - a_j$ est inversible dans $\mathbb{A}$ lorsque $i \neq j$.

En effet, si $\mathbb{V} = (a_{j-1}^{i-1})$ est la matrice de Vandermonde associée aux points a_i, dont le déterminant vaut $\det(\mathbb{V}) = \prod_{i<j} (a_i - a_j)$, alors le problème d'interpolation se traduit par la résolution en $\mathbf{p}$ du système linéaire $\mathbb{V}\mathbf{p} = \mathbf{b}$, où $\mathbf{b}$ est le vecteur des b_i. Or, ce système admet une solution unique, puisque l'hypothèse **(H)** entraîne l'inversibilité du déterminant $\det(\mathbb{V})$ et donc aussi celle de la matrice $\mathbb{V}$.

Exercice 1. Montrer que $\det(\mathbb{V}) = \prod_{i<j}(a_i - a_j)$.

Cette discussion suggère un algorithme naïf pour l'interpolation, produisant l'unique solution $\mathbf{p} = \mathbb{V}^{-1}\mathbf{b}$ du système $\mathbb{V}\mathbf{p} = \mathbf{b}$ à l'aide du pivot de Gauss, en $O(n^3)$ opérations dans $\mathbb{A}$. De manière analogue, l'évaluation multipoint de P en

les a_i se traduit par le produit matrice-vecteur $\mathbb{V}\mathbf{p}$, où $\mathbf{p}$ est le vecteur des coefficients de P ; elle peut donc être effectuée de manière naïve en $O(n^2)$ opérations arithmétiques. En s'y prenant *vraiment naïvement*, l'interpolation est donc plus couteuse que l'évaluation multipoint. Nous verrons un peu plus loin qu'une méthode exploitant *la formule d'interpolation de Lagrange* permet de résoudre le problème d'interpolation en complexité *quadratique* en n.

L'objectif de ce cours est de montrer qu'il est possible d'atteindre un complexité quasi-optimale, tant pour l'évaluation multipoint que pour l'interpolation, en utilisant des algorithmes de type « diviser pour régner » qui ramènent ces questions à des multiplications de polynômes. Les principaux résultats sont les suivants :

THÉORÈME 1. *On peut effectuer l'évaluation et l'interpolation sur n points en utilisant $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ de $\mathbb{A}$. Cette complexité peut être abaissée à $O(\mathsf{M}(n))$ si les points sont en progression géométrique.*

En termes pratiques, si l'anneau de base $\mathbb{A}$ est un corps fini, les algorithmes rapides deviennent avantageux pour des degrés de l'ordre de quelques dizaines : c'est sensiblement mieux que pour les algorithmes rapides de type « diviser pour régner » utilisés dans le Cours 10 pour le calcul de pgcd ou d'approximants de Padé-Hermite rapide ; cela reflète une relative simplicité des algorithmes.

Extensions. L'évaluation multipoint et l'interpolation sont des instances particulières des problèmes *modulos multiples* et *restes chinois* s'énonçant comme suit :

Modulos multiples. Étant donné un polynôme P dans $\mathbb{A}[X]$, de degré strictement inférieur à $\sum_i \deg m_i$, calculer les restes :

$$P \bmod m_1, \ldots, P \bmod m_r.$$

Restes Chinois. Étant donnés des polynômes $b_1, \ldots, b_r, m_1, \ldots, m_r$ dans $\mathbb{A}[X]$, avec m_i unitaires, retrouver le polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que

$$P \bmod m_1 = b_1, \ldots, P \bmod m_r = b_r.$$

Les techniques de ce cours s'y généralisent[1] et mènent aux résultats suivants :

THÉORÈME 2. *On peut résoudre les deux problèmes ci-dessus en $O(\mathsf{M}(n)\log n)$ opérations $(+,-,\times)$ dans $\mathbb{A}$.*

Naturellement, on peut se poser les questions précédentes dans le cas où l'anneau de polynômes $\mathbb{A}[X]$ est remplacé par l'anneau $\mathbb{Z}$, les polynômes m_i sont remplacés par des moduli entiers a_i et où l'on cherche un P entier à n chiffres. Il est possible d'obtenir le même type de résultats de complexité, cette fois en comptant les opérations binaires, mais nous nous limiterons dans la suite au cas polynomial.

La suite de ce cours est organisée comme suit : dans la Section 3 nous décrivons la méthode d'interpolation de Lagrange, de complexité quadratique. La Section 4 est consacrée aux algorithmes rapides sous-jacents à la preuve du Théorème 1.

[1] Pour les restes chinois, l'unicité du résultat est garantie par l'hypothèse que le résultant $\mathrm{Res}(m_i, m_j)$ est un élément inversible dans $\mathbb{A}$, pour tous $i \neq j$. Cette condition technique généralise la condition d'inversibilité des $a_i - a_j$; se rapporter au Cours 10 pour la définition du résultant.

3. Interpolation de Lagrange

Un algorithme de complexité quadratique pour l'interpolation polynomiale repose sur une écriture explicite du polynôme interpolant. Soient b_i les valeurs à interpoler. On peut alors écrire P sous la forme :

$$P(X) \;=\; \sum_{i=0}^{n-1} b_i \prod_{0\le j\le n-1}^{j\neq i} \frac{X-a_j}{a_i-a_j}.$$

Cette égalité est usuellement appelée la *formule d'interpolation de Lagrange*. Pour la prouver, il suffit d'observer que pour tout i, le produit s'annule en a_j ($j \neq i$), et vaut 1 en a_i. Afin de simplifier l'écriture de la formule de Lagrange, posons

$$A = \prod_j (X-a_j) \quad \text{et} \quad A_i = \prod_{j\neq i}(X-a_j) = \frac{A}{X-a_i},$$

pour $i = 0, \ldots, n-1$. Pour $i \neq j$, on a donc les relations $A(a_i) = 0$ et $A_i(a_j) = 0$. De plus, $A_i(a_i)$ est inversible sous l'hypothèse **(H)**. On obtient alors l'écriture

$$P \;=\; \sum_{i=0}^{n-1} b_i \frac{A_i(X)}{A_i(a_i)}.$$

Une approche directe pour l'interpolation revient à calculer tout d'abord les polynômes A_i, puis leurs valeurs en les points a_i, et enfin à faire les combinaisons linéaires nécessaires. Le seul point non-trivial est le calcul des A_i : si on n'y fait pas attention, on risque de sortir des bornes en $O(n^2)$ (par exemple, si on les calcule tous indépendamment, et chacun de manière quadratique). Pour faire mieux, on partage le gros des calculs, en calculant d'abord le polynôme A.

Interpolation de Lagrange

Entrée : $a_0, \ldots, a_{n-1} \in \mathbb{A}$ vérifiant **(H)** et $b_0, \ldots, b_{n-1}$ dans $\mathbb{A}$.

Sortie : L'unique polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que $P(a_i) = b_i$ pour tout i.

1. $A \leftarrow 1$, $P \leftarrow 0$
2. Pour $i = 0, \ldots, n-1$ faire
   $A \leftarrow A \cdot (X - a_i)$
3. Pour $i = 0, \ldots, n-1$ faire
   $A_i \leftarrow A/(X - a_i)$
   $q_i \leftarrow A_i(a_i)$
   $P \leftarrow P + b_i A_i / q_i$

Proposition 1. *L'algorithme ci-dessus utilise $O(n^2)$ opérations dans $\mathbb{A}$.*

Démonstration. La multiplication d'un polynôme de degré d par un polynôme de degré 1 prend un temps linéaire en d, disons Cd opérations, C étant une constante. Calculer A demande donc $C(1 + 2 + \cdots + (n-1)) = O(n^2)$ opérations. Ensuite, chaque passage dans la seconde boucle prend un temps linéaire en n. Il y a n passages, d'où à nouveau du $O(n^2)$. □

4. Algorithmes rapides

4.1. Les idées. L'idée fondamentale est à nouveau d'utiliser une technique de type « diviser pour régner ». Supposons pour simplifier que n est pair et séparons l'ensemble des points $a_0, \dots, a_{n-1}$ en deux paquets, $a_0, \dots, a_{n/2-1}$ et $a_{n/2}, \dots, a_{n-1}$. Il est naturel de *scinder* également P en deux polynômes P_0 et P_1 de degré strictement inférieur à $n/2$, tels que :

$$P_0(a_0) = P(a_0), \dots, P_0(a_{n/2-1}) = P(a_{n/2-1}) \quad \text{et}$$
$$P_1(a_{n/2}) = P(a_{n/2}), \dots, P_1(a_{n-1}) = P(a_{n-1}).$$

Pour effectuer ce scindage, le choix suivant s'impose :

$$P_0 = P \bmod (X - a_0) \cdots (X - a_{n/2-1})$$
$$P_1 = P \bmod (X - a_{n/2}) \cdots (X - a_{n-1}).$$

On a donc ramené le calcul en degré n à deux calculs en degré $n/2$, plus deux divisions euclidiennes ; on va ensuite itérer ce processus. En bout de course, on retrouve les valeurs $P(a_i)$, car celles-ci peuvent également s'écrire $P \bmod (X - a_i)$.

Remarquons de suite que les polynômes par lesquels on effectue les divisions euclidiennes ne sont pas « gratuits » : on a besoin de les calculer. Pour cela, l'idée est de les empiler dans une structure d'arbre binaire, dont les nœuds internes sont étiquetés par des polynômes.

4.2. L'arbre des sous-produits. Pour simplifier, dans tout ce qui suit, on suppose que le nombre de points est une puissance de 2, $n = 2^\kappa$. Quand ce n'est pas le cas, on adapte toutes les constructions (on fabrique toujours un arbre binaire, mais il lui manque alors des nœuds internes à certains niveaux). On peut donner la définition récursive suivante pour cet arbre (noté $\mathcal{B}$).

- Si $\kappa = 0$, $\mathcal{B}$ se réduit à un nœud unique qui contient le polynôme $X - a_0$.
- Si $\kappa > 0$, soient $\mathcal{B}_0, \mathcal{B}_1$ les arbres associés à $a_0, \dots, a_{2^{\kappa-1}-1}$ et $a_{2^{\kappa-1}}, \dots, a_{2^\kappa-1}$. Soient M_0 et M_1 les polynômes présents aux racines de $\mathcal{B}_0$ et $\mathcal{B}_1$. Alors $\mathcal{B}$ est l'arbre dont la racine contient le produit $M_0 M_1$ et dont les fils sont $\mathcal{B}_0$ et $\mathcal{B}_1$.

Graphiquement, on a la représentation suivante :

$$A = \textstyle\prod_{i=0}^{n-1}(X - a_i)$$

$$B_0 = \textstyle\prod_{i=0}^{n/2-1}(X - a_i) \qquad\qquad B_1 = \textstyle\prod_{i=n/2}^{n-1}(X - a_i)$$

$$\vdots \qquad\qquad \vdots \qquad \vdots \qquad\qquad \vdots$$

$$(X - a_0)(X - a_1) \qquad\qquad (X - a_{n-2})(X - a_{n-1})$$

$$X - a_0 \qquad X - a_1 \qquad \cdots \qquad X - a_{n-2} \qquad X - a_{n-1}$$

De manière équivalente, on peut représenter cet arbre par un tableau à 2 dimensions $B_{i,j}$, pour $0 \leq i \leq \kappa$, $0 \leq j \leq 2^{\kappa-i} - 1$. Alors

$$B_{i,j} = \prod_{\ell=2^i j}^{2^i(j+1)-1} (X - a_\ell).$$

Par exemple, si $\kappa = 2$ et $n = 4$, l'arbre associé à a_0, a_1, a_2, a_3 est représenté par :

$$B_{0,0} = X - a_0, B_{0,1} = X - a_1, B_{0,2} = X - a_2, B_{0,3} = X - a_3,$$
$$B_{1,0} = (X - a_0)(X - a_1), B_{1,1} = (X - a_2)(X - a_3),$$
$$B_{2,0} = (X - a_0)(X - a_1)(X - a_2)(X - a_3).$$

L'intérêt de cette structure d'arbre est double : d'une part, il contient tous les polynômes par lesquels on va diviser lors de l'algorithme d'évaluation, d'autre part, le coût de son calcul s'amortit, et devient essentiellement linéaire en le degré.

PROPOSITION 2. *On peut calculer tous les polynômes contenus dans l'arbre $\mathcal{B}$ pour $O(\mathsf{M}(n) \log n)$ opérations $(+, -, \times,)$ dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $\mathsf{T}(n)$ le coût du calcul de l'arbre pour n points. La définition récursive implique l'inégalité suivante pour $\mathsf{T}(n)$:

$$\mathsf{T}(n) \leq 2\mathsf{T}\left(\frac{n}{2}\right) + \mathsf{M}\left(\frac{n}{2}\right).$$

Le résultat s'ensuit aisément en invoquant le lemme « diviser pour régner ». □

4.3. Évaluation. L'algorithme d'évaluation devient simple à écrire de manière récursive, si on s'autorise un peu de souplesse dans l'écriture, en supposant précalculés tous les nœuds de l'arbre.

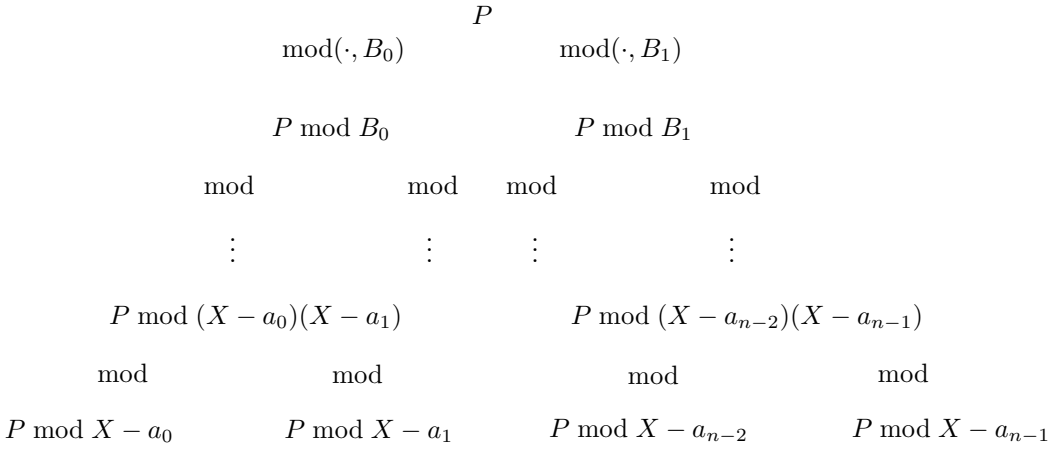
EvaluationRapide

Entrée : P dans $\mathbb{A}[X]$ et $a_0, \ldots, a_{n-1}$ dans $\mathbb{A}$, avec $\deg P < n$.

Sortie : $P(a_0), \ldots, P(a_{n-1})$.

1. Si $n = 0$, renvoyer P
2. $P_0 \leftarrow P \bmod (X - a_0) \cdots (X - a_{n/2-1})$
3. $P_1 \leftarrow P \bmod (X - a_{n/2}) \cdots (X - a_{n-1})$
4. Calculer (par un appel récursif) $L_0 = P_0(a_0), \ldots, P_0(a_{n/2-1})$
5. Calculer (par un appel récursif) $L_1 = P_1(a_{n/2}), \ldots, P_1(a_{n-1})$
6. Renvoyer L_0, L_1

Graphiquement, cela correspond à faire *descendre* les restes de divisions euclidiennes dans l'arbre des sous-produits :



Or, la division avec reste par un polynôme unitaire de degré n se fait en $O(\mathsf{M}(n))$ opérations de $\mathbb{A}$ (Cours 2). Nous obtenons le résultat de complexité suivant :

PROPOSITION 3. *Supposant l'arbre $\mathcal{B}$ précalculé, la complexité de l'algorithme précédent est de $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $\mathsf{T}(n)$ le coût du calcul pour n points. L'algorithme ci-dessus implique la récurrence suivante pour $\mathsf{T}(n)$:

$$\mathsf{T}(n) \leq 2\,\mathsf{T}\left(\frac{n}{2}\right) + O(\mathsf{M}(n)),$$

d'où l'on déduit le résultat, à l'aide du lemme « diviser pour régner ». □

EXERCICE 2. Montrer que la FFT est un cas particulier de l'algorithme ci-dessus. (Indication : pour les moduli $m_i X - \omega^i$, où $\omega \in \mathbb{A}$ est une racine primitive de l'unité, l'arbre des sous-produits contient des polynômes de la forme $X^d - c$, $c \in \mathbb{A}$, modulo lesquels les divisions se font en complexité linéaire).

4.4. Sommes de fractions. Un problème intermédiaire à traiter avant de résoudre l'interpolation est celui du calcul efficace d'une somme de fractions. Rappelons les définitions introduites dans la Section 3. À partir des points a_i, nous y avions défini les polynômes

$$A = \prod_j (X - a_j) \qquad \text{et} \qquad A_i = \prod_{j \neq i} (X - a_j) = \frac{A}{X - a_i}.$$

Il était apparu que pour effectuer l'interpolation, il fallait savoir effectuer la tâche suivante : étant données des valeurs c_i, calculer le numérateur et le dénominateur de la fraction rationnelle

$$\sum_{i=0}^{n-1} \frac{c_i}{X - a_i}. \tag{1}$$

C'est à cette question que nous allons répondre maintenant ; à nouveau, on utilise une idée de type « diviser pour régner » : par deux appels récursifs, on calcule

$$S_1 = \sum_{i=0}^{n/2-1} \frac{c_i}{X - a_i}\ 1 \quad \text{et} \quad S_2 = \sum_{i=n/2}^{n} \frac{c_i}{X - a_i}$$

et on renvoie $S = S_1 + S_2$. Soit $\mathsf{T}(n)$ le coût du calcul pour n points. Le coût de l'algorithme SommesFractions esquissé ci-dessus satisfait à la récurrence suivante :

$$\mathsf{T}(n) \leq 2\mathsf{T}\left(\frac{n}{2}\right) + O(\mathsf{M}(n)),$$

d'où l'on déduit le résultat suivant, à l'aide du lemme « diviser pour régner ».

PROPOSITION 4. *L'algorithme* SommesFractions *calcule le dénominateur et le numérateur de la fraction rationnelle (1) en* $O(\mathsf{M}(n)\log n)$ *opérations dans* $\mathbb{A}$.

EXERCICE 3. Estimer la complexité de l'algorithme direct pour évaluer un polynôme de degré au plus n et ses premières n dérivées en un point. Imaginer un algorithme de complexité quasi-linéaire résolvant ce problème.

4.5. Interpolation. Arrivés à ce stade, l'interpolation ne pose plus de réelle difficulté. Au vu des formules de la Section 3, le polynôme interpolant les valeurs b_i aux points a_i est donné par :

$$P(X) \;=\; \sum_{i=0}^{n-1} b_i \frac{A_i(X)}{A_i(a_i)} \;=\; A(X) \times \sum_{i=0}^{n-1} \frac{b_i/A_i(a_i)}{X - a_i}.$$

D'après les paragraphes précédents, on sait comment calculer rapidement la fraction rationnelle $\sum c_i/(X-a_i)$; il ne reste plus qu'à calculer les constantes $c_i = b_i/A_i(a_i)$. Cela devient évident au vu de la proposition suivante :

PROPOSITION 5. *Pour tout* $i = 0, \ldots, n-1$, *on a* $A_i(a_i) = A'(a_i)$.

DÉMONSTRATION. On dérive A, ce qui donne :

$$A' = \sum_{i=0}^{n-1} \prod_{j \neq i} (X - a_j) = \sum_{i=0}^{n-1} A_i.$$

On a vu précédemment que $A_i(a_j) = 0$ pour $i \neq j$, ce qui donne le résultat. □

L'algorithme d'interpolation et l'estimation de son coût s'en déduisent facilement.

InterpolationRapide

Entrée : $a_0, \ldots, a_{n-1} \in \mathbb{A}$ vérifiant **(H)** et $b_0, \ldots, b_{n-1}$ dans $\mathbb{A}$.

Sortie : L'unique polynôme $P \in \mathbb{A}[X]$ de degré inférieur à n tel que $P(a_i) = b_i$ pour tout i.

1. Calculer l'arbre des sous-produits associé aux points a_i (de racine A).
2. $(d_0, \ldots, d_{n-1}) \leftarrow$ EvaluationRapide$(A', (a_0, \ldots, a_{n-1}))$
3. $(c_0, \ldots, c_{n-1}) \leftarrow (b_0/d_0, \ldots, b_{n-1}/d_{n-1})$
4. $R \leftarrow$ SommesFractions$(c_0/(X - a_0), \ldots, c_{n-1}/(X - a_{n-1}))$
5. Renvoyer le numérateur de R.

PROPOSITION 6. *La complexité de l'algorithme ci-dessus est* $O(\mathsf{M}(n)\log n)$ *opérations de* $\mathbb{A}$.

DÉMONSTRATION. C'est une conséquence des Propositions 2, 3 et 4. □

EXERCICE 4. Soit k un corps et soient $m_1, m_2 \in k[X]$ de degrés au plus n, premiers entre eux et soit $v_1, v_2 \in k[X]$ de degrés inférieurs à n. Donner un algorithme de complexité $O(\mathsf{M}(n)\log n)$ qui calcule $f \in k[X]$ de degré inférieur à $2n$ tel que $f = v_1 \bmod m_1$ et $f = v_2 \bmod m_2$. En déduire un algorithme diviser pour régner pour l'interpolation polynomiale en degré n, de complexité $O(\mathsf{M}(n)\log^2 n)$.

4.6. Évaluation et interpolation sur une suite géométrique. Dans cette section nous montrons que tout polynôme de degré n peut être évalué et interpolé sur des points en progression géométrique $\{1, q, \ldots, q^{n-1}\}$ en $O(\mathsf{M}(n))$ opérations.

PROPOSITION 7. *Soit $q \in \mathbb{A}, n \geq 1$ tels que $q, q-1, q^2-1, \ldots, q^{n-1}-1$ soient inversibles dans $\mathbb{A}$. Si $F \in \mathbb{A}[X]$ est de degré strictement inférieur à n alors :*

- *On peut évaluer F sur les points $a_i = q^i$, pour $0 \leq i \leq n-1$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.*
- *On peut interpoler F sur les points $a_i = q^i$, pour $0 \leq i \leq n-1$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.*

DÉMONSTRATION. Soit $F = f_0 + f_1 X + \cdots + f_{n-1}X^{n-1}$. Pour $i = 0, \ldots, 2n-2$, on introduit les nombers triangulaires $t_i = i(i-1)/2$ et la suite $b_i = q^{t_i}$; pour $i = 0, \ldots, n-1$ on construit $c_i = f_i/b_i$. Tous les éléments q^{t_i}, c_i et b_i se calculent en $O(n)$ opérations, car $q^{t_{i+1}} = q^i q^{t_i}$. L'algorithme exploite la formule

$$F(q^i) = \sum_{j=0}^{n-1} f_j q^{ij} = b_i^{-1} \cdot \sum_{j=0}^{n-1} c_j b_{i+j},$$

qui montre que les valeurs $F(q^i)$ sont, à des facteurs constants près, données par les coefficients de $X^{n-1}, \ldots, X^{2n-2}$ dans le produit de $\sum_{i=0}^{n-1} c_i X^{n-i-1}$ et $\sum_{i=0}^{2n-2} b_i X^i$.

Pour l'interpolation, on commence par noter que la racine $A(X)$ de l'arbre des sous-produits peut être calculée en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$. En effet, puisque les points a_i forment une suite géométrique, les nœds $B_{i,j}$ à l'étage i peuvent se déduire l'un de l'autre par une homothétie de coût $O(n)$. Par ailleurs, observons que dans l'algorithme InterpolationRapide, le calcul de tout l'arbre des sous-produits est nécessaire *uniquement* pour l'évaluation multipoint de A'. Or, par la première partie du théorème, l'évaluation de A' sur les points a_i se fait en $O(\mathsf{M}(n))$ par un algorithme *qui ne requiert pas le calcul de tout l'arbre des sous-produits*.

Il nous reste à prouver que la somme des fractions (1), qui devient ici

$$\frac{N}{A} = \sum_{i=0}^{n-1} \frac{c_i}{X - q^i} \tag{2}$$

peut également être déterminée pour le même prix.

On adopte la stratégie suivante : on calcule le développement en série à l'ordre n de la fraction (2). Celui-ci suffit pour en déduire son numérateur $N(X)$. Pour ce faire, on utilise la formule $1/(a-X) = \sum_{i \geq j} a^{-j-1}X^j$ valable dans $\mathbb{A}[[X]]$ pour tout $a \in \mathbb{A}$ inversible. On obtient :

$$\sum_{i=0}^{n-1} \frac{c_i}{X - q^i} \mod X^n = -\sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} c_i q^{-i(j+1)} X^j \right) = -\sum_{j=0}^{n-1} C(q^{-j-1}) X^j,$$

où $C(X)$ est le polynôme $\sum_{i=0}^{n-1} c_i X^i$. Or, les points $q^{-1}, q^{-2}, \ldots, q^{-n}$ étant en progression géométrique, on sait évaluer C sur ces points en $O(\mathsf{M}(n))$. □

Exercice 5. Montrer que tout l'arbre des sous-produits associé aux points $a_i = q^i$, $i = 0, \ldots, n-1$, peut être calculé en $\mathsf{M}(n) + O(n \log n)$ opérations dans $\mathbb{A}$.

Exercice 6. Soient $a, b, c, d \in \mathbb{A}$. Montrer qu'on peut évaluer tout polynôme de degré au plus n en $\{ba^{2i} + ca^i + d, \quad 0 \leq i < n\}$, en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$.

Notes

Pour évaluer un polynôme $P(X) = p_{n-1}X^{n-1} + \cdots + p_0$ en un seul point a, le *schéma de Horner* $P(a) = (\cdots((p_{n-1}a + p_{n-2})a + p_{n-3})a + \cdots + p_0)$ minimise le nombre d'opérations (additions ou multiplications). Cet algorithme effectue $n-1$ additions et $n-1$ multiplications dans $\mathbb{A}$, et on ne peut pas faire mieux en général (si les coefficients du polynôme sont algébriquement indépendants — moralement, s'ils sont des indéterminées). Ce résultat d'optimalité a été conjecturé par Ostrowski [13] et prouvé par Pan [14]. Par contre, pour un polynôme donné, il peut être possible de faire mieux : l'évaluation de $P(X) = X^{n-1}$ se fait en temps logarithmique.

Une forme particulière du Théorème de restes Chinois a été enoncée il y plus de 2000 ans par le mathématicien chinois Sun Tsu. Le traitement moderne est dû à Gauss [8]. Ce théorème admet une formulation très générale, en termes d'idéaux d'anneaux non nécessairement commutatifs : Si R est un anneau et $I_1, \ldots, I_r$ des idéaux (à gauche) de R mutuellement premiers (*i. e.* $I_i + I_j = R$ pour $i < j$), alors l'anneau quotient $R/\cap_i I$ est isomorphe à l'anneau produit $\prod R/I_i$ via l'isomorphisme $x \bmod I \quad \mapsto (x \bmod I_1, \ldots, x \bmod I_r)$. La formule d'interpolation de Lagrange est due à Waring [20]. Les avantages pratiques de l'arithmétique modulaire ont été mises en évidence dans les années 1950 par Svoboda et Valach [19].

Le premier algorithme sous-quadratique, de complexité arithmétique $O\big(n^{1.91}\big)$, pour l'évaluation multipoint est dû à Borodin et Munro [4] et repose sur une approche *pas de bébés / pas de géants* exploitant la multiplication sous-cubique des matrices de Strassen [17]. Horowitz [10] a étendu ce résultat à l'interpolation. La Proposition 2 calcule efficacement les fonctions symétriques élémentaires de n éléments $a_0, \ldots, a_{n-1}$ de $\mathbb{A}$; elle est due également à Horowitz [10].

S'inspirant de la FFT, Fiduccia [7] eut l'idée d'un algorithme pour l'évaluation multipoint à base de division polynomiale récursive. Ne disposant pas de division de complexité sous-quadratique, son algorithme récursif reste quadratique. Borodin et Moenck corrigent ce point dans deux articles successifs [11, 3], reposant sur les algorithmes quasi-optimaux pour la division de [11, 18]. Montgomery [12] améliore la constante dans la complexité $O\big(\mathsf{M}(n) \log n\big)$ de l'évaluation multipoint, sous l'hypothèse que la FFT est utilisée pour la multiplication polynomiale. L'algorithme d'évaluation sur une suite géométrique exposé dans ce cours vient de [15, 2]. Les algorithmes fournissant les meilleures constantes actuellement connues dans les $O(\cdot)$ du Théorème 1 sont obtenus à l'aide du principe de transposition de Tellegen [5, 6].

On connaît peu de choses sur la complexité intrinsèque de l'évaluation multipoint et de l'interpolation en degré n. Strassen [18] a prouvé une borne inférieure de type $n \log n$. Shoup et Smolensky [16] ont montré que ces bornes restent essentiellement vraies même dans un modèle où des précalculs en quantité illimitée sur les points d'évaluation sont permis. Cependant, l'optimalité des meilleurs algorithmes connus, de complexité $O(n \log^2 n)$ reste un problème ouvert. Même dans le cas particulier où les points d'évaluation sont en progression arithmétique, on ne dispose d'aucun

algorithme d'évaluation multipoint de complexité $O(\mathsf{M}(n))$, ni d'une preuve qu'un tel algorithme n'existe pas. De même, pour des points quelconques $a_0, \ldots, a_{n-1}$ on ne connaît pas d'algorithme de complexité $O(\mathsf{M}(n))$ pour calculer le polynôme $\prod_i (X - a_i)$. Notons toutefois que tels algorithmes existent dans les cas particuliers où les a_i forment une progression arithmétique ou géométrique [6].

L'exercice 4 est provient de [9] ; historiquement, ce fut le premier algorithme rapide pour les restes chinois. Les exercices 3 et 6 sont tirés de [1].

Bibliographie

[1] Aho (A. V.), Steiglitz (K.), and Ullman (J. D.). – Evaluating polynomials at fixed sets of points. *SIAM Journal on Computing*, vol. 4, n° 4, 1975, pp. 533–539.

[2] Bluestein (L. I.). – A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Trans. Electroacoustics*, vol. AU-18, 1970, pp. 451–455.

[3] Borodin (A.) and Moenck (R. T.). – Fast modular transforms. *Comput. Sys. Sci.*, vol. 8, n° 3, 1974, pp. 366–386.

[4] Borodin (A.) and Munro (I.). – Evaluation of polynomials at many points. *Information Processing Letters*, vol. 1, n° 2, 1971, pp. 66–68.

[5] Bostan (Alin), Lecerf (Grégoire), and Schost (Éric). – Tellegen's principle into practice. In Sendra (J. R.) (editor), *Symbolic and Algebraic Computation*. pp. 37–44. – ACM Press, 2003. Proceedings of ISSAC'03, Philadelphia, August 2003.

[6] Bostan (Alin) and Schost (Éric). – Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, vol. 21, n° 4, 2005, pp. 420–446. – Festschrift for Arnold Schönhage.

[7] Fiduccia (C. M.). – Polynomial evaluation via the division algorithm : The fast fourier transform revisited. In *Conference Record, Fourth Annual ACM Symposium on Theory of Computing*, pp. 88–93. – 1972.

[8] Gauss (Carl Friedrich). – *Disquisitiones arithmeticae*. – Yale University Press, New Haven, Conn., 1966, *Translated into English by Arthur A. Clarke, S. J*, xx+472p.

[9] Heindel (L. E.) and Horowitz (E.). – On decreasing the computing time for modular arithmetic. In *12 th annual symposium on switching and automata theory*. pp. 126–128. – IEEE Computer Society Press, 1971.

[10] Horowitz (E.). – A fast method for interpolation using preconditioning. *Information Processing Letters*, vol. 1, n° 4, 1972, pp. 157–163.

[11] Moenck (R. T.) and Borodin (A.). – Fast modular transforms via division. *Thirteenth Annual IEEE Symposium on Switching and Automata Theory (Univ. Maryland, College Park, Md., 1972)*, 1972, pp. 90–96.

[12] Montgomery (P. L.). – *An FFT extension of the elliptic curve method of factorization*. – PhD thesis, University of California, Los Angeles CA, 1992.

[13] Ostrowski (A.). – On two problems in abstract algebra connected with Horner's rule. In *Studies in mathematics and mechanics presented to Richard von Mises*, pp. 40–48. – Academic Press Inc., New York, 1954.

[14] Pan (V. Ja.). – On means of calculating values of polynomials. *Uspehi Mat. Nauk*, vol. 21, n° 1 (127), 1966, pp. 103–134.

[15] Rabiner (L. R.), Schafer (R. W.), and Rader (C. M.). – The chirp z-transform algorithm and its application. *Bell System Tech. J.*, vol. 48, 1969, pp. 1249–1292.

[16] Shoup (Victor) and Smolensky (Roman). – Lower bounds for polynomial evaluation and interpolation problems. *Computational Complexity*, vol. 6, n° 4, 1996/97, pp. 301–311.

[17] Strassen (V.). – Gaussian elimination is not optimal. *Numerische Mathematik*, vol. 13, 1969, pp. 354–356.

[18] Strassen (V.). – Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numerische Mathematik*, vol. 20, 1972/73, pp. 238–251.

[19] Svoboda (A.) and Valach (M.). – Oper'atorov'e obvody (operational circuits). *Stroje na Zpracov 'an'i Informac'i III, Nakl. CSAV, Praha*, 1955, pp. 247–295.

[20] Waring (E.). – Problems concerning interpolations. *Philosophical Transactions of the Royal Society of London*, vol. 59, 1779, pp. 59–67.

COURS 6

Récurrences linéaires à coefficients polynomiaux : n-ième terme, n premiers termes

Résumé

Le calcul du n-ième terme d'une suite donnée par une récurrence linéaire à coefficients polynomiaux intervient fréquemment en combinatoire et pour calculer des troncatures de séries, ainsi que comme étape clé dans des algorithmes de recherche de solutions polynomiales d'équations fonctionnelles linéaires et dans un algorithme de factorisation déterministe d'entiers. Pour une récurrence d'ordre r dont les coefficients ont degré au plus d, l'algorithme naïf par déroulement de la récurrence a complexité arithmétique $O(rdn)$ et complexité binaire $O\big(rn^2 I(d\log n)\big)$. Ces complexités sont optimales pour le calcul de tous les n premiers termes. Pour le calcul du n-ième terme seul, la technique des pas de bébés et pas de géants atteint une meilleure complexité arithmétique en $O\big(r^2 M(\sqrt{dn}\,)\log dn + r^\omega M(\sqrt{dn}\,)\big)$; la méthode du scindage binaire fournit une complexité binaire quasi-optimale en $O\big(r^\omega I(dn\log n)\log n\big)$.

Le chapitre 4 a montré comment calculer le n-ième terme d'une suite donnée par une récurrence à coefficients contants en complexité arithmétique en $O(\log n)$. Dans ce chapitre, nous nous attaquons au cadre plus général des récurrences à coefficients polynomiaux. Les algorithmes n'ont pas une aussi bonne complexité que pour des coefficients constants, mais cette fois encore, il existe des algorithmes plus efficaces que le simple déroulement de la récurrence.

La situation se distingue de celle des problèmes et algorithmes présentés jusqu'ici : les idées qui permettent le calcul du n-ième terme en bonne complexité arithmétique ne sont pas les mêmes que pour abaisser la complexité binaire. Ainsi, l'algorithme par pas de bébés et pas de géants de la section 2 donne un gain en racine de n sur la complexité arithmétique, par rapport à la méthode naïve. Ce gain est typique de la technique. Quant à l'algorithme par scindage binaire de la section 3, il n'abaisse en rien la complexité arithmétique, mais améliore pourtant la complexité binaire jusqu'à la rendre quasi-linéaire en la taille de sa sortie.

Nous terminons cette introduction par un peu de terminologie. Les solutions u de suites de la forme

$$p_r(n)u_{n+r} + \cdots + p_0(n)u_n = 0, \qquad (n \in \mathbb{N}), \tag{1}$$

pour des polynômes p_i sont dites *polynomialement récursives* ou en abrégé *P-récursives*. Dans le cas spécifique d'ordre 1, une suite u est dite *hypergéométrique*. Remarquons que la définition se récrit alors sous la forme suivante, plus facile à mémoriser :

$$\frac{u_{n+1}}{u_n} = -\frac{p_0(n)}{p_1(n)}.$$

Autrement dit, le quotient de deux termes successifs de la suite est donné par l'évaluation d'une fraction rationnelle fixée, sauf peut-être en un nombre fini de valeurs de n. Les suites P-récursives et encore plus les suites hypergéométriques joueront un rôle important dans des chapitres ultérieurs.

1. Calcul naïf de $n!$ et de suites P-récursives

La définition de la factorielle comme produit,

$$n! = 1 \times 2 \times \cdots \times n,$$

montre que $n!$ peut être calculé en $n-1$ opérations arithmétiques.

L'estimation de la complexité binaire de cette méthode repose sur la *formule de Stirling* :

$$\log n! = n \log n - n \log e + \frac{1}{2} \log n + O(1), \qquad n \to \infty.$$

En particulier, nous retiendrons l'équivalence $\log n! \sim n \log n$ qui donne la taille de $n!$. La k-ième étape du produit multiplie $k!$ par $k+1$, et donc un entier de taille $\log k! = O(k \log n)$ avec un entier de taille $\log(k+1) = O(\log n)$. Ce produit déséquilibré coûte donc moins de $ckI(\log n)$ opérations binaires pour une certaine constante c. Le calcul complet de la factorielle par la méthode naïve effectue donc moins de

$$\sum_{k=1}^{n} ckI(\log n) = O\big(n^2 I(\log n)\big)$$

opérations binaires.

Ces complexités arithmétique et binaire sont quasi-optimales pour le calcul conjoint des nombres $1!$, $2!$, …, $n!$.

Pour une suite P-récursive donnée par une récurrence de la forme (1), le calcul de u_{n+r} à partir des r valeurs précédentes de la suite demande $O(rd)$ opérations arithmétiques pour l'évaluation des polynômes (par exemple par le schéma de Horner) puis $O(r)$ opérations pour effectuer la combinaison linéaire des u_{n+i}. Le calcul de u_n à partir des r premières valeurs de la suite demande donc $O(rdn)$ opérations arithmétiques.

La complexité binaire par cette méthode découle de nouveau essentiellement de la croissance de u_n. Pour estimer celle-ci, il est agréable de faire apparaître u_n comme le quotient v_n/w_n des suites définies par les récurrences

$$v_{n+r} + p_{r-1} v_{n+r-1} \cdots + p_0(n) v_n = 0, \qquad p_r(n) w_{n+r} = w_{n+r-1}, \qquad (n \in \mathbb{N}),$$

et les conditions initiales $v_i = u_i$ et $w_i = 1$ quand $0 \le i < r$. Par une vision matricielle, v_n est donné comme première coordonnée du vecteur V_n de la suite vectorielle définie par la récurrence du premier ordre

$$V_{n+1} = A(n) V_n \quad \text{avec} \quad A(n) = \frac{1}{p_r(n)} \begin{pmatrix} 0 & p_r(n) & & & \\ & 0 & p_r(n) & & \\ & & \ddots & \ddots & \\ & & & 0 & p_r(n) \\ -p_0(n) & & \cdots & & -p_{r-1}(n) \end{pmatrix}.$$

La matrice $A(n)$ se développe asymptotiquement sous la forme $A_\delta n^\delta + A_{\delta-1} n^{\delta-1} + \ldots$, pour des matrices constantes A_i et un entier δ entre $-d$ et d. Il s'ensuit que

sa norme est asymptotiquement équivalente à Cn^δ pour une certaine constante C, après tout choix de norme sur les matrices $r \times r$. Les majorations

$$|v_n| \le |V_n| \le |A(n)| \dots |A(1)||U_0| \le C^n (n!)^\delta \le C^n (n!)^d$$

fournissent la borne $O(dn \log n)$ sur la taille $\log |v_n|$. Par le même raisonnement, la taille du dénominateur w_n est du même ordre, si bien que par le même type d'argument que pour la factorielle, la complexité binaire du calcul naïf de u_n est $O\big(n^2 I(d \log n)\big)$.

2. Pas de bébés et pas de géants

On sait que le n-ième terme d'une récurrence linéaire à coefficients constants peut être calculé en complexité arithmétique $O(\log n)$. Dans le cas des coefficients polynomiaux, les choses se compliquent : à ce jour, on ne connaît pas d'algorithme polynomial en $\log n$. L'exemple typique en est le calcul du n-ième terme de la factorielle $u_n = n!$, qui vérifie la récurrence à coefficients polynomiaux $u_{n+1} = (n+1)u_n$ pour $n \ge 0$. Une solution efficace exploite le théorème 1 du chapitre 5. Elle utilise la technique des *pas de bébés et pas de géants* et requiert un nombre d'opérations arithmétiques en $\sqrt{n}$ (à des facteurs logarithmiques près). Pour simplifier la présentation, supposons que n est un carré parfait. L'idée de l'algorithme est de poser

$$P(X) = (X+1)(X+2)\dots(X+n^{1/2}),$$

afin d'obtenir la valeur de u_n à l'aide de l'équation

$$u_n = \prod_{j=0}^{n^{1/2}-1} P\big(jn^{1/2}\big). \tag{2}$$

Cette égalité suggère la procédure suivante :

1. *Pas de bébés* : Calculer les coefficients de P. Ceci peut être fait en $O\big(\mathsf{M}(\sqrt{n}\,)\log n\big)$ opérations arithmétiques (en construisant un arbre binaire de feuilles $X+i$, voir le chapitre 5).
2. *Pas de géants* : Évaluer P sur les points 0, $\sqrt{n}$, $2\sqrt{n}$, …, $(\sqrt{n}-1)\sqrt{n}$ et retrouver la valeur de u_n à l'aide de l'équation (2). En utilisant le théorème 1 du chapitre 5, ceci se fait en $O\big(\mathsf{M}(\sqrt{n}\,)\log n\big)$ opérations arithmétiques.

Le coût total de cet algorithme est de $O\big(\mathsf{M}(\sqrt{n})\log n\big)$ opérations arithmétiques. Si la FFT est utilisée pour la multiplication des polynômes, le gain par rapport à la méthode directe est de l'ordre de $\sqrt{n}$, à des facteurs logarithmiques près, typique pour la technique des pas de bébés et pas de géants.

Ce résultat se généralise au problème du calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux. À cette fin, le polynôme à considérer est maintenant le polynôme matriciel

$$P(X) = A(X+m)\dots A(X+2)A(X+1),$$

pour $m = (n/d)^{1/2}$. Le produit $A(n)\dots A(1)$ est alors le produit de $(dn)^{1/2}$ évaluations de P, lequel a degré $(dn)^{1/2}$. L'algorithme obtient ces évaluations matricielles en réalisant successivement les évaluations multipoints des r^2 coordonnées de la matrice P.

EXERCICE 1. Généraliser ce résultat au calcul du Nième terme d'une récurrence d'ordre quelconque.

EXERCICE 2. étant donné un polynôme $f \in k[X]$ de degré 2 et un entier $N \geq 0$, quel est le nombre d'opérations dans k nécessaires pour déterminer le coefficient de X^N du polynôme f^N ? (Indication : La solution directe consiste à calculer tous les coefficients u_n de f^N modulo X^{N+1} par exponentiation binaire. Son coût est de $O\big(\mathsf{M}(N)\big)$ opérations arithmétiques. Une approche plus rapide repose sur le fait que les u_n satisfont une récurrence à coefficients polynomiaux d'ordre 2.)

EXERCICE 3. Imaginez un algorithme qui teste si une équation différentielle à coefficients dans $\mathbb{Z}[X]$ admet une solution polynomiale de degré au plus N, en $O(\mathsf{M}(\sqrt{N})\log(N))$ opérations bits.

2.1. Factorisation déterministe des entiers. Supposons que nous devons factoriser un entier N. Tester tous les diviseurs plus petits que $\sqrt{N}$ a un coût linéaire en $\sqrt{N}$ (dans ce paragraphe, par coût on entend complexité bit). Afin d'accélérer ce calcul, Strassen [4] propose de rassembler tous les entiers plus petits que $\sqrt{N}$ en $\sqrt[4]{N}$ blocs, chacun contenant $\sqrt[4]{N}$ entiers consécutifs. Soit c de l'ordre de $\sqrt[4]{N}$ et notons $f_0 = 1\cdots c \bmod N$, $f_1 = (c+1)\cdots(2c) \bmod N$, ..., $f_{c-1} = (c^2-c+1)\cdots(c^2) \bmod N$. Si les valeurs $f_0, \ldots, f_{c-1}$ sont connues, alors il devient facile de déterminer un facteur de N, en prenant des pgcd de $f_0, \ldots, f_{c-1}$ avec N. Ainsi, la principale difficulté est de calculer les valeurs f_i.

Pour ce faire, on prend $R = \mathbb{Z}/N\mathbb{Z}$ et F le polynôme $(X+1)\cdots(X+c) \in R[X]$, qu'on évalue en les points $0, c, 2c, \ldots, c(c-1)$ en $O(\mathsf{M}(c)\log c)$ opérations de R. Par le Théorème 1 et puisque c est d'ordre $\sqrt[4]{N}$, la complexité totale est de $O(\mathsf{M}(\sqrt[4]{N})\log N)$ opérations modulo N, soit $O(\mathsf{M}(\sqrt[4]{N})\,\mathsf{M}(\log N)\log N)$ opérations bits. Notons qu'on ne connaît pas de meilleur algorithme déterministe ; l'existence d'un tel algorithme est un grand problème ouvert.

3. Scindage binaire

3.1. Cas de la factorielle. Pour exploiter la multiplication rapide, l'idée consiste à équilibrer les produits en calculant récursivement $P(a,b) = (a+1)(a+2)\cdots b$ par

$$P(a,b) = P(a,m)P(m,b) \quad \text{où} \quad m = \left\lfloor \frac{a+b}{2} \right\rfloor.$$

Appelons $C(a,b)$ le coût binaire du calcul de $P(a,b)$. Il résulte immédiatement de la méthode que ce coût vérifie l'inégalité

$$C(a,b) \leq C(a,m) + C(m,b) + I\big(\log P(a,m), \log P(m,b)\big).$$

Sous l'hypothèse raisonnable que la complexité de la multiplication d'entiers est croissante avec la taille des entiers, le coût du calcul de $P(a,m)$ est inférieur au coût du calcul de $P(m,b)$, d'où la nouvelle inégalité

$$C(a,b) \leq 2C(m,b) + I\big(P(m,b)\big).$$

À ce stade, le lemme « diviser pour régner » n'est pas suffisant pour conclure, mais l'utilisation de l'inégalité précédente donne les inégalités successives

$$\begin{aligned} C(0,n) &\le 2C(n/2,n) + I\big(\log P(n/2,n)\big) \\ &\le 4C(3n/4,n) + 2I\big(\log P(3n/4,n)\big) + I\big(\log P(n/2,n)\big) \\ &\le \cdots \\ &\le 2^k C(n-2^{-k}n,n) + 2^k I\big(\log P(n-2^{-k}n,n)\big) + \cdots + I\big(\log P(n/2,n)\big), \end{aligned}$$

pour tout entier positif k. L'entier $P(n-2^{-k}n,n)$ est le produit de $2^{-k}n$ facteurs de taille bornée par $\log n$; il a donc pour taille $2^{-k}n\log n$. Par sous-additivité de la fonction I, la dernière inégalité devient

$$C(0,n) \le 2^k C(n/2^k,n) + kI(n\log n).$$

En choisissant finalement d'arrêter la récursion lorsque $n-2^{-k}n$ et n diffèrent d'au plus un, ce qui impose k de l'ordre de $\log n$, on aboutit à la borne

$$C(0,n) \le O(\log n) + I(n\log n)\log n$$

donc à une complexité binaire dans $O\big(I(n\log n)\log n\big)$.

Si la multiplication utilisée est la FFT, cette complexité s'écrit $O(n\log^3 n\log\log n)$; si la multiplication est d'exposant de n strictement plus grand que 1, elle s'écrit $O\big(I(n\log n)\big)$.

3.2. Récurrences d'ordre 1. La factorielle de la section précédente suit la récurrence $u_{n+1} = (n+1)u_n$. On considère ici tout d'abord les solutions de récurrences de la forme

$$u_{n+1} = p(n)u_n, \qquad p \in \mathbb{Z}[X].$$

Si p a degré d, $\log p(n)$ est dans $O(d\log n)$, si bien que la taille de u_n est $O(dn\log n)$.

De même, pour toute récurrence de la forme

$$u_{n+1} = \frac{p(n)}{q(n)}u_n, \qquad p,q \in \mathbb{Z}[X],$$

on peut pour calculer u_n appliquer séparément la même technique de scindage binaire sur le numérateur et le dénominateur. Si d est le maximum des degrés de p et q, le calcul produit deux entiers de taille $O(dn\log n)$ en un nombre d'opérations binaires en $O\big(I(dn\log n)\log n\big)$. Ensuite, si le dénominateur est non nul, la méthode de Newton permet d'effectuer la division finale en $O\big(I(dn\log n)\big)$ opérations binaires.

3.3. Calcul de $e = \exp(1)$. Le point de départ est la suite (e_n) donnée par

$$e_n = \sum_{k=0}^{n} \frac{1}{k!}.$$

Cette suite converge vers e. Plus précisément, il est classique que le terme de reste dans $e - e_n$ se majore par une série géométrique de sorte que

$$0 \le e - e_n \le \frac{1}{n\,n!}.$$

Pour calculer N décimales de e par cette série, il suffit de rendre $\frac{1}{n\,n!}$ inférieur à 10^{-N}. Il s'ensuit qu'il suffit de prendre N de sorte que $n\,n!$ et 10^N soient du

même ordre, c'est-à-dire d'avoir N proportionnel à $n \log n$. Il suffit donc de prendre $n = O(N/\log N)$ termes dans la série.

La suite $(e_n)_{n\geq 0}$ vérifie $e_n - e_{n-1} = 1/n!$ et donc

$$n(e_n - e_{n-1}) = e_{n-1} - e_{n-2}.$$

Cette récurrence se récrit

$$\begin{pmatrix} e_n \\ e_{n-1} \end{pmatrix} = \frac{1}{n} \underbrace{\begin{pmatrix} n+1 & -1 \\ n & 0 \end{pmatrix}}_{A(n)} \begin{pmatrix} e_{n-1} \\ e_{n-2} \end{pmatrix} = \frac{1}{n!} A(n)A(n-1)\cdots A(2) \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Le calcul du produit de matrices peut alors être effectué par scindage binaire. Le calcul de e_N par ce procédé demande donc $O(I(N \log N) \log N)$ opérations binaires.

Pour calculer n décimales de e, la première étape ci-dessus construit deux entiers de taille $O(N \log N) = O(n)$ en $O(I(n) \log n)$ opérations binaires. Il faut ensuite effectuer une division qui ne demande que $O(I(n))$ opérations par l'algorithme de Newton. L'ensemble du calcul est donc quasi-optimal. (En outre, le $\log n$ n'est pas présent pour des multiplications comme celle de Karatsuba dont l'exposant de complexité est supérieur à 1.)

Un autre exemple d'application est donné dans les notes.

3.4. Suites polynomialement récursives.

DEFINITION 1. Une suite $(a_n)_{n\geq 0}$ d'éléments d'un anneau commutatif unitaire $\mathbb{A}$ est appelée suite polynomialement récursive (ou P-récursive) si elle vérifie une récurrence de la forme

$$p_d(n)a_{n+d} + p_{d-1}(n)a_{n+d-1} + \cdots + p_0(n)a_n = 0, \qquad n \geq 0,$$

où les p_i sont des polynômes de $\mathbb{A}[X]$.

Dans la suite on fait l'hypothèse que le coefficient de tête p_d ne s'annule pas sur les entiers $0, \ldots, N-d$, où N est l'indice du terme maximal que l'on souhaite calculer. On note D une borne sur le degrés des polynômes p_i et, par h une borne sur la taille de leurs coefficients lorsque ceux-ci sont entiers.

THÉORÈME 1. *Avec les notations de la definition 1, le calcul de $a_0, \ldots, a_N$ peut être effectué en $O(dNM(D)/D)$ opérations arithmétiques. Dans le cas où $p_i \in \mathbb{Z}[X]$, $i = 0, \ldots, d$, la complexité binaire de la méthode naïve est en $O(dN^2 I(hD \log N))$ et celle du scindage binaire pour calculer a_N est en $O(d^\omega I(hDN \log N) \log N)$.*

Comme précédemment, cette dernière complexité descend à $O(d^\omega I(hDN \log N))$ si l'exposant de la complexité I est supérieur à 1.

EXERCICE 4. Vérifier les formules de ce théorème.

Exercices

EXERCICE 5 (Calcul rapide de factorielle et de coefficients binomiaux centraux). Cet exercice montre comment calculer certaines suites récurrentes linéaires plus vite que par la méthode de scindage binaire.

Soit $N \in \mathbb{N}$ et soit $Q = \sum_{i=0}^{2N} q_i X^i \in \mathbb{Z}[X]$ le polynôme $Q(X) = (1+X)^{2N}$.

1. Montrer que q_N peut être calculé en utilisant uniquement des additions d'entiers du triangle de Pascal, c'est-à-dire l'identité suivante sur les coefficients du binôme :
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \qquad k \geq 1, n \geq 1.$$
Quelle est la complexité binaire de cet algorithme ?

On admet que le calcul de tous les nombres premiers inférieurs à N peut être effectué en $O(N \log N / \log\log N)$ opérations binaires, qu'il existe au plus $3N/\log(N)$ tels nombres premiers et que la multiplicité avec laquelle apparaît le nombre premier p dans la factorisation de $N!$ vaut
$$\mathrm{ind}(p, N) = \sum_{i=1}^{\infty} \left[\frac{N}{p^i}\right],$$
où la notation $[x]$ représente la partie entière de x.

2. Montrer que le calcul de $\mathrm{ind}(p, N)$ peut être effectué en $O(\log N I(\log N))$ opérations binaires.
3. Montrer que la décomposition en facteurs premiers de $N!$ peut être effectuée en $O(\log N I(N))$ opérations binaires, ainsi que celle de q_N.
4. Montrer que $N!$ et q_N peuvent alors être reconstruits en respectivement $O(I(N \log N) \log\log N)$ et $O(I(N) \log N)$ opérations binaires.

Notes

L'algorithme par pas de bébés et pas de géants a été introduit par Strassen [4] et généralisé dans [2] au problème du calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux.

L'exercice 2 est inspiré de [3, Pb. 4].

Le même raisonnement se généralise au calcul des sommes convergentes de suites hypergéométriques. En particulier, c'est ainsi que les systèmes de calcul formel calculent rapidement π par une formule découverte en 1989 par les frères Chudnovsky :
$$\frac{1}{\pi} = \frac{12}{C^{3/2}} \sum_{n=0}^{\infty} \frac{(-1)^n (6n)! (A + nB)}{(3n)! n!^3 C^{3n}}$$
où $A = 13591409$, $B = 545140134$ et $C = 640320$.

La partie de l'exercice 5 consacrée au calcul de $N!$ est due à P. Borwein [1].

Bibliographie

[1] Borwein (Peter B.). – On the complexity of calculating factorials. *Journal of Algorithms*, vol. 6, n° 3, 1985, pp. 376–380.

[2] Chudnovsky (D. V.) and Chudnovsky (G. V.). – Approximations and complex multiplication according to Ramanujan. In *Ramanujan revisited*, pp. 375–472. – Academic Press, Boston, MA, 1988.

[3] Flajolet (Philippe) and Salvy (Bruno). – The Sigsam challenges : Symbolic asymptotics in practice. *SIGSAM Bulletin*, vol. 31, n° 4, December 1997, pp. 36–47.

[4] Strassen (V.). – Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, vol. 78, n° 1, 1976/77, pp. 1–8.

COURS 7

Algèbre linéaire : de Gauss à Strassen

1. Introduction

Les problèmes algorithmiques en algèbre linéaire cachent des questions très subtiles. En général, les première questions que l'on se pose en rapport avec la manipulation des matrices sont celles du calcul du produit matrice-matrice, éventuellement du produit matrice-vecteur, du calcul de l'inverse, de la résolution de systèmes, etc ...

Les réponses à ces questions vont varier selon le type de matrices que l'on considère. Une classification possible est la suivante.

- Les matrices quelconques, sans structure, pas particulièrement creuses, sont appellées matrices *denses*. Nous verrons que l'algorithmique de ces matrice peut se ramener essentiellement au produit de matrices, dont la complexité est une question extrêmement délicate.
- Un grand nombre de probèmes linéaires se formulent en termes de matrices *creuses*, éventuellement définies sur de petits corps finis. Dans ce cas, l'algorithmique dense est inadaptée ; il est possible d'aller beaucoup plus loin en utilisant des outils mieux adaptés, à base de récurrences linéaires.
- Enfin, on a déjà rencontré (ou entendu parler de) familles de matrices structurées, telles que les matrices de Sylvester pour le résultant (et le PGCD), Vandermonde pour l'évaluation et l'interpolation, etc ... Les questions qui se posent dans ce cadre sont principalement celles du produit matrice-vecteur, ou de la résolution de système. Pour ces types de matrices clairement identifiés, on développe une algorithmique ad-hoc.

Schématiquement, l'algorithmique des matrices denses quelconques est la plus lente (de complexité entre $O(n^2)$ et $O(n^3)$, en taille n) ; la complexité du calcul avec les matrices creuses est de l'ordre de $O(n^2)$, et celle des matrices structurées que l'on a déjà rencontrées est en $O(n)$, à des logs près.

On va maintenant détailler ces résultats ; on ne reparlera désormais que des deux premiers cas, les algorithmes pour matrices structurées ne rentrant pas vraiment dans le cadre du cours présent.

Matrices denses. Dans le cas des matrices quelconques, les questions que l'on sera amené à se poser, ou simplement évoquer, dans ce cours sont celles de la complexité :

- du produit,
- du calcul d'inverse,
- du calcul du polynôme caractéristique ou minimal d'une matrice,
- de la mise sous une forme "canonique" (Smith, Hermite, Frobenius, LUP, ...)

– de la résolution de systèmes linéaires, ...

Comme pour de nombreuses questions déjà rencontrées, on dispose pour effectuer ces opérations d'algorithmes "naïfs" : pour les opérations ci-dessus, leur complexité est cubique en la taille. Le résultat principal de ce cours est que l'on peut faire mieux, et qu'il existe une constante ω (dépendante *a priori* du corps de base), qui contrôle la complexité quasiment toutes les opérations d'algèbre linéaire.

Pour formaliser ce point, on associe à chacun des problèmes ci-dessus son *exposant*. Dans le cas du produit, il est défini comme suit :

$$C_{\text{produit}}(n) = \text{coût minimal d'un algorithme pour le produit en taille } n$$

et

$$\omega_{\text{produit}} = \inf\{\tau \mid C_{\text{produit}}(n) \in O(n^\tau)\}.$$

L'algorithme naïf pour le produit est de complexité $O(n^3)$, donc $\omega_{\text{produit}} \leq 3$. On se doute bien que de l'autre côté, $\omega_{\text{produit}} \geq 2$: il faut au moins n^2 opérations.

Ensuite, on peut définir de la même manière les exposants de tous les autres problèmes, de la forme ω_{inverse}, $\omega_{\text{polynôme caractéristique}}$, $\omega_{\text{déterminant}}$, $\omega_{\text{décomposition LUP}}$, etc ... On a alors le résultat suivant.

THÉORÈME 1. *Les exposants* ω_{produit}, ω_{inverse}, $\omega_{\text{polynôme caractéristique}}$, $\omega_{\text{déterminant}}$, $\omega_{\text{décomposition LUP}}$, *sont tous égaux, et inférieurs à* 2.38 *; on note* ω *leur valeur commune. L'exposant correspondant à la résolution de systèmes linéaires est inférieur ou égal à* ω.

Ce théorème contient deux (familles de) résultats :

– des preuves d'*équivalence* entre problèmes,

– des bornes supérieures sur leur complexité, c'est-à-dire des algorithmes.

Dans le présent chapitre, il est impossible de démontrer ce théorème dans son intégralité. On se contentera de prouver (et encore, sous des hypothèses simplificatrices), que le produit et l'inverse ont le même exposant, et que celui-ci est inférieur à $\log_2 7 < 2.81$.

Matrices creuses. Les questions que l'on se pose en algorithmique creuse sont différentes. Ici, on considère une matrice de taille n, qui ne contient que s entrées non nulles, avec $s \ll n^2$ (qui correspondrait à une matrice pleine). Remarquer qu'alors, on sait effectuer le produit matrice-vecteur en $O(s)$ opérations.

Il n'est pas vraiment naturel de se poser la question du produit ou du calcul d'inverse pour les matrices creuses, le caractère creux n'étant pas vraiment stable par produit ou inverse (autrement dit, les exposants précédemment introduits ne sont pas les bons outils pour étudier les matrices creuses).

La question la plus fréquemment rencontrée dans ce cadre est celle de la *résolution de système*. Nous verrons une version simplifiée d'un algorithme probabiliste dû à Wiedemann, qui repose essentiellement sur des produits matrice-vecteur, et un calcul d'approximants de Padé.

THÉORÈME 2. *Soit* A *une matrice* $n \times n$ *inversible, contenant* s *entrées non nulles. On peut calculer une solution du système linéaire* $Ax = y$ *en* $O(ns)$ *opérations, par un algorithme probabiliste.*

Le cas où la matrice A n'est pas inversible demande davantage de travail, mais on obtient au final le même style de résultat.

Applications. Les questions que l'on a évoquées ici sont *très* fréquemment rencontrées en pratique, que ce soit dans la pratique du calcul formel ou du monde réel.

- De nombreux algorithmes de résolution de systèmes polynomiaux reposent sur de l'algèbre linéaire : les méthodes de calcul de base de Gröbner peuvent se ramener à de l'algèbre linéaire (principalement creuse) en très grande taille (milliers ou millions) ; d'autres approches demandent des produits et inverses de matrices de taille plus modérée (de l'ordre de la dizaine), mais remplies de polynômes de degrés potentiellement élevés (potentiellement des milliers).
- Les algorithmes de factorisation d'entiers les plus récents fonctionnent en deux phases. Tout d'abord on effectue une collecte d'information qui peut être effectuée de manière distribuée, et qui vise à fabriquer une (énorme) matrice creuse et définie sur $\mathbb{F}_2$; ensuite, on cherche un vecteur dans le noyau de cette matrice. Cette application a été un des moteurs de la recherche sur les matrices creuses, dans la communauté du calcul formel.

  D'autres algorithmes de cryptanalyse fonctionnent sur le même schéma ; le logarithmes discret dans les corps finis est un exemple typique.
- Les algorithmes factorisation des polynômes dans un corps fini reposent souvent sur des calculs d'algèbre linéaire : c'est manifeste pour certains d'entre eux (Berlekamp), et plus caché pour d'autres (l'algèbre linéaire intervenant pour accélérer des calculs de types "pas de bébé, pas de géant" ; nous y reviendrons dans un chapitre ultérieur).
- Mentionnons enfin qu'une large partie des ordinateurs du vrai monde passent leurs cycles à faire des calculs d'algèbre linéaire, que ce soit pour faire des calculs d'optimisation combinatoire (programmation linéaire par l'algorithme du simplexe), de la simulation numérique (méthode des éléments finis), ou de la recherche de pages web (Google repose sur de la recherche d'un vecteur propre d'une gigantesque matrice creuse).

2. Matrices denses

L'objectif de cette section est modeste : on va dans un premier temps étudier trois algorithmes de multiplication de matrices denses, en reportant à un cours ultérieur une étude approfondie de cette question. Dans un second temps, on montre comment réduire l'inversion à la multiplication, et réciproquement, de sorte que les deux problèmes ont essentiellement la même complexité.

2.1. Multiplication.

Algorithme naïf. Commençons par décrire l'algorithme "cubique" de multiplication. Étant données deux matrices A et B de taille n, leur produit s'écrit $C = AB$ avec

$$C_{i,k} = \sum_{j=1}^{n} A_{i,j} B_{j,k}.$$

Le coût nécessaire pour obtenir une entrée de C est donc de n multiplications et $n-1$ additions, de sorte que la complexité totale est en $O(n^3)$.

Raffinement : deux fois moins de divisions. Un algorithme dû à Winograd [2] permet essentiellement de diviser par deux le nombre de multiplications. Il repose sur l'identité suivante : pour tous $i, j, j'k$ on a

$$(a_{i,j} + b_{j',k})(a_{i,j'} + b_{j,k}) = a_{i,j}b_{j,k} + a_{i,j'}b_{j',k} + a_{i,j}a_{i,j'} + b_{j,k}b_{j',k}.$$

Ainsi, on récupère deux contributions au terme $C_{i,k}$ au prix d'une seule multiplication, à des termes correctifs près.

L'idée est maintenant d'associer j et j' d'une manière ou d'une autre (par exemple en posant $j' = n-j$; on suppose donc que n est pair), et de faire $i = 1, \dots, n$ et $k = 1, \dots, n$. Le point-clé est qu'il y seulement $O(n^2)$ termes correctifs $a_{i,j}a_{i,j'}$ et $b_{j,k}b_{j',k}$, car le premier ne dépend pas de k, et le second ne dépend pas de i. On peut donc tous les calculer pour $O(n^2)$ multiplications. Ensuite, la formule ci-dessus permet de n'effectuer que $n^3/2$ multiplications, au prix de $O(n^3)$ additions supplémentaires.

Au total, le coût de la multiplication de l'algorithme de Winograd est de $\frac{1}{2}n^3 + n^2$ multiplications et $\frac{3}{2}n^3 + 3n^2 - 2n$ additions (quand n est pair). Ainsi, on a essentiellement transformé $n^3/2$ multiplications en additions, ce qui est appréciable (moralement, les multiplications sont toujours plus difficiles que les additions).

Algorithme de Strassen. L'algorithme de Strassen fut le premier algorithme à descendre en-dessous de $O(n^3)$. Tout comme l'algorithme de Karatsuba pour les polynômes, celui-ci repose sur le gain d'une multiplication pour les matrices 2×2, qui devient un gain dans l'exposant quand on l'applique récursivement.

Soient donc A et B des matrices de taille 2. L'algorithme naïf demande d'effectuer 8 multiplications. L'algorithme de Strassen revient à

- calculer des combinaisons linéaires des $a_{i,j}$ et des $b_{i,j}$,
- effectuer **7** produits de ces combinaisons linéaires,
- recombiner les résultats pour obtenir le produit AB.

Explicitement, les formules à appliquer sont les suivantes. On commence par calculer

$$\left|\begin{array}{lcl} q_1 &=& (A_{1,1} - A_{1,2})B_{2,2} \\ q_2 &=& (A_{2,1} - A_{2,2})B_{1,1} \\ q_3 &=& A_{2,2}(B_{1,1} + B_{2,1}) \\ q_4 &=& A_{1,1}(B_{1,2} + B_{2,2}) \\ q_5 &=& (A_{1,1} + A_{2,2})(B_{2,2} - B_{1,1}) \\ q_6 &=& (A_{1,1} + A_{2,1})(B_{1,1} + B_{1,2}) \\ q_7 &=& (A_{1,2} + A_{2,2})(B_{2,1} + B_{2,2}) \end{array}\right.$$

On en déduit les entrées du produit AB de la manière suivante :

$$\left|\begin{array}{lcl} C_{1,1} &=& q_1 - q_3 - q_5 + q_7 \\ C_{1,2} &=& q_4 - q_1 \\ C_{2,1} &=& q_2 + q_3 \\ C_{2,2} &=& -q_2 - q_4 + q_5 + q_6 \end{array}\right.$$

Effectuons maintenant le pas récursif. Comme pour les polynômes, on ne se refuse pas le confort de supposer que les matrices ont une taille qui est une puissance de 2 ; l'analyse s'étendrait facilement dans le cas général.

Soient donc A et B des matrice de taille $2n$, n étant lui-même une puissance de 2. On procède à une découpe en blocs de A et B :

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \quad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}.$$

Dans cette écriture, les $A_{i,j}$ et $B_{i,j}$ sont donc des matrices $n \times n$.

Le point-clé est le suivant : les formules données ci-dessus pour le cas 2×2 s'appliquent toujours si les $A_{i,j}$ et $B_{i,j}$ sont des matrices. Ainsi, elles permettent de ramener le produit en taille $2n$ à 7 produits en taille n, plus un certain nombre d'additions, disons Cn^2, où C est une constante (pour fabriquer les petites combinaisons linéaires avant de faire les produits, et récupérer ensuite les entrées de AB).

En en déduit le résultat suivant :

THÉORÈME 3. *Le produit AB peut se calculer en $O(n^{\log_2(7)}) \simeq O(n^{2.81})$ opérations de k.*

DÉMONSTRATION. Le coût $T(n)$ satisfait la récurrence

$$T(2n) \leq 7T(n) + Cn^2.$$

La conclusion s'en déduit facilement ; la démonstration est la même que celle de la complexité de Karatsuba. □

Il est difficile de donner une intuition pour motiver les formules de Strassen (à la différence de Karatsuba, qui repose de manière cachée sur l'évaluation en des valeurs entières). Voici ce que Strassen en dit :

First I had realized that an estimate tensor rank < 8 for two by two matrix multiplication would give an asymptotically faster algorithm. Then I worked over $\mathbb{Z}/2\mathbb{Z}$ (as far as I remember) to simplify matters.

Autrement dit, il y a dû y avoir recherche "à la main". Quant à la mention du tensor rank, elle est du ressort du prochain cours.

L'algorithme de Winograd mentionné plus haut effectue lui aussi le produit de matrices 2×2 en 7 opérations. Par contre, on ne peut pas l'utiliser de manière récursive comme celui de Strassen : la formule qui sous-tend Winograd repose sur le fait que $ab = ba$, si a et b sont des scalaires (des éléments du corps de base). Lors des appels récursifs, les objets à multiplier sont eux-mêmes des matrices, pour lesquels la loi de commutation n'est plus vérifiée. On dit donc que l'algorithme de Winograd est un algorithme *non-commutatif*.

Pour faire mieux que 2.81 *?* Strassen ayant prouvé que le produit de matrices était sous-cubique, la question naturelle est devenue de savoir si on pouvait descendre jusqu'à du $O(n^2)$. Des progrès ont été faits tout au long des années 1970 et 1980, les meilleurs algorithmes actuels étant en $O(n^{2.38})$.

Pour mesurer la qualité de ces algorithmes, on introduit naturellement la notion d'*exposant de la multiplication*, c'est-à-dire de la borne inférieure ω_{produit} de l'ensemble des réels τ tels que le produit de deux matrices de taille n peut se faire en $O(n^\tau)$ opérations.

On a évidemment $\omega_{\text{produit}} \geq 2$; l'algorithme de Strassen montre que $\omega_{\text{produit}} \leq \log_2(7)$, et les meilleures bornes supérieures actuelles sont $\omega_{\text{produit}} \leq 2.38$. On n'a pas de borne inférieure non-triviale pour ω_{produit} (c'est-à-dire autre que 2).

L'étude des algorithmes et des idées qui permettent de faire mieux que $\log_2(7)$ sera effectuée au prochain cours, si tout va bien.

En pratique. Les algorithmes rapides pour l'algèbre linéaire ont longtemps eu mauvaise presse ; concrètement, l'algorithme de Strassen, et, de manière plus marginale, un algorithme dû à Pan et repris par Kaporin [1], d'exposant 2.77, sont les seuls algorithmes "rapides" (avec un exposant inférieur à 3) qui ont été implantés avec succès.

Dans une bonne implantation, disons sur un corps fini "raisonnable", l'algorithme de Winograd est immédiatement rentable. L'algorithme de Strassen devient ensuite meilleur pour des tailles de l'ordre de quelques dizaines (64 est une borne raisonnable).

L'immense majorié des autres algorithmes connus reposent sur des techniques très complexes, qui se traduisent par la présence d'énormes constantes (et de facteurs logarithmiques) dans leurs estimation de complexité. En conséquence, dans leur versions actuelles, il ne peuvent pas être rentables pour des tailles inférieures à des millions ou des milliards ...

2.2. Autres opérations. Ce n'est pas tout de savoir multiplier les matrices ; il est tout aussi légitime de vouloir les inverser, calculer leur polynôme caractéristique, etc ...

L'objectif de cette sous-section est de montrer (ou de suggérer) que *toutes ces opérations sont équivalentes à la multiplication* en termes de complexité, et donc équivalentes entre elles. Ainsi, l'exposant ω_{produit} devient exposant pour quasiment tous les problèmes "classiques" d'algèbre linéaire. Le seul problème qui résiste est la résolution de systèmes : on sait qu'il n'est pas plus dur que la multiplication, mais peut-être est il plus facile ...

Le but de ce paragraphe est de donner quelques exemples de réductions entre ces problèmes : on montre (presque) l'équivalence entre l'inversion est la multiplication.

La multiplication n'est pas plus difficile que l'inversion. Cette réduction repose sur une formule ingénieuse. Soient A et B des matrices $n \times n$. On souhaite calculer $C = AB$. Pour cela, on pose

$$D = \begin{bmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{bmatrix}$$

Alors

$$D^{-1} = \begin{bmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{bmatrix}$$

On ramène donc le produit en taille n à l'inversion en taille $3n$, ce qui prouve notre affirmation.

L'inversion n'est pas plus difficile que la multiplication. Soit A la matrice à inverser. On prśente ici un algorithme simple d'inversion ; celui-ci nécessite d'inverser des sous-matrices de A, des produits de telles sous-matrices, ... On suppose que *toutes ces matrices sont inversibles.* Le cas général est plus délicat, il passe par le calcul d'une décomposition LUP de A, ce qui est assez technique.

L'algorithme est récursif. Supposons que A est de taille $2n \times 2n$, et écrivons la sous une forme bloc

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

En posant $S = A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}$, on se convainc que A peut se réécrire

$$A = \begin{bmatrix} I_n & 0 \\ A_{2,1}A_{1,1}^{-1} & I_n \end{bmatrix} \begin{bmatrix} A_{1,1} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I_n & A_{1,1}^{-1} \\ 0 & I_n \end{bmatrix}.$$

On en déduit que l'inverse de A s'écrit

$$A = \begin{bmatrix} I_n & -A_{1,1}^{-1} \\ 0 & I_n \end{bmatrix} \begin{bmatrix} A_{1,1}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I_n & 0 \\ -A_{2,1}A_{1,1}^{-1} & I_n \end{bmatrix}.$$

On est donc amenés à effectuer deux inversions, ainsi qu'un certain nombre de produits, ce qui mène au résultat suivant.

THÉORÈME 4. *Soit ω_{produit} un exposant pour la multiplication de matrices. Si toutes les sous-matrices rencontrées au cours de l'algorithme sont inversibles, le coût de l'inversion de A est $O(n^{\omega_{\text{produit}}})$.*

DÉMONSTRATION. L'algorithme ci-dessus montre que la complexité $I(n)$ de l'inversion satisfait la récurrence

$$I(2n) \leq 2I(n) + Cn^{\omega_{\text{produit}}},$$

C étant une constante. On en déduit le résultat (attention, il faut utiliser le fait que $\omega_{\text{produit}} \geq 2$ pour ne pas perdre bêtement un facteur log dans la résolution de la récurrence). □

Exercices

EXERCICE 1 (Évaluation rapide de polynôme en une matrice). Soit $\mathbb{K}$ un corps et soit $\mathcal{M}_n(\mathbb{K})$ l'ensemble des matrices carrées de taille n à coefficients dans $\mathbb{K}$. On considère un réel $\alpha > 2$ tel que la multiplication de deux matrices de $\mathcal{M}_n(\mathbb{K})$ peut se faire en $\mathsf{C}(n) = n^\alpha$ opérations dans $\mathbb{K}$. On suppose que la multiplication dans $\mathbb{K}[X]$ en degré inférieur à d peut se faire en au plus $\mathsf{M}(d)$ opérations dans $\mathbb{K}$, où M vérifie l'inégalité $\mathsf{M}(d+d') \geq \mathsf{M}(d) + \mathsf{M}(d')$ pour tous $d, d' \geq 1$.

On rappelle que le polynôme caractéristique d'une matrice de $\mathcal{M}_n(\mathbb{K})$ peut se calculer en $O(\mathsf{C}(n)\log n)$ opérations de $\mathbb{K}$ et que toute matrice annule son polynôme caractéristique.

Le but de l'exercice est d'étudier la complexité arithmétique du problème suivant :

$\mathcal{E}(P, A)$: étant donnés un polynôme $P = p_0 + p_1X + \cdots + p_DX^D$ de $\mathbb{K}[X]$ et une matrice $A \in \mathcal{M}_n(\mathbb{K})$, calculer la matrice $P(A) = p_0I_n + p_1A + \cdots + p_DA^D$.

1. Montrer que $O(\mathsf{C}(n)\sqrt{D} + n^2 D)$ opérations de $\mathbb{K}$ suffisent pour résoudre $\mathcal{E}(P, A)$.
   Indication : Écrire $P(X)$ sous la forme $P_0(X)+P_1(X)X^d+P_2(X)X^{2d}+\cdots$, avec un bon choix de l'entier d et des polynômes $P_i(X)$ de degrés au plus $d-1$.
2. Montrer que lorsque $D = n$, cette complexité peut être abaissée à $O(\mathsf{C}(n)\sqrt{n})$. Indication : on pourra s'inspirer de l'algorithme de composition de séries de Brent & Kung.
3. Montrer qu'on peut améliorer le résultat de la question 1 en $O(\mathsf{C}(n)\sqrt{n} + \mathsf{M}(D))$ opérations de $\mathbb{K}$. Indication : commencer par calculer le polynôme caractéristique de A.
4. Montrer qu'on peut calculer A^D en $O(\mathsf{C}(n)\sqrt{n} + \mathsf{M}(n)\log D)$ opérations de $\mathbb{K}$.
5. Montrer que si $A \in \mathcal{M}_n(\mathbb{K})$ et $\deg(P) = n$, la première colonne de $P(A)$ peut être calculée en $O(\mathsf{C}(n)\log n)$ opérations de $\mathbb{K}$.
6. Montrer que si $C \in \mathcal{M}_n(\mathbb{K})$ est une matrice compagnon[1], alors on peut résoudre $\mathcal{E}(P, C)$ en $O(n^2 + \mathsf{M}(D))$ opérations de $\mathbb{K}$.
   Indication : Commencer par calculer la première colonne de $P(C)$ en ce coût. Puis considérer l'effet de la multiplication par la matrice $P(C)$ dans une base appropriée du quotient $\mathbb{K}[X]/(c(X))$, où c est le polynôme caractéristique de C, supposé irréductible.
7. Montrer que s'il existe un réel $\beta > 2$ tel que $\mathcal{E}(X^2, A)$ puisse se résoudre en n^β opérations de $\mathbb{K}$ pour toute matrice A, alors la multiplication de deux matrices quelconques de $\mathcal{M}_n(\mathbb{K})$ peut se faire en $O(n^\beta)$ opérations dans $\mathbb{K}$.

Dans la suite, on suppose que la matrice A est « suffisamment générique », en ce sens que la matrice $U \in \mathcal{M}_n(\mathbb{K})$ dont la ℓ^{e} colonne ($1 \le \ell \le n$) coïncide avec la première colonne de $A^{\ell-1}$, est une matrice inversible.

8. Montrer que la matrice $C = UAU^{-1}$ est une matrice de type compagnon et qu'elle peut être calculée en $O(\mathsf{C}(n)\log n)$ opérations de $\mathbb{K}$.
   Indication : Remarquer que l'espace vectoriel $\mathcal{V} = \mathbb{K}^n$ admet $\{A^\ell e\}_{1\le\ell\le n}$ comme base, où $e = (1, 0, \ldots, 0)^t \in \mathcal{V}$ et que U est la matrice de passage entre cette base et la base canonique de $\mathcal{V}$.
9. En déduire que, pour une matrice A « suffisamment générique », le problème $\mathcal{E}(P, A)$ peut se résoudre en $O(\mathsf{C}(n)\log n + \mathsf{M}(D))$ opérations de $\mathbb{K}$.
10. Montrer que que, pour une matrice A « suffisamment générique », on peut résoudre $\mathcal{E}(X^D, A)$ en $O(\mathsf{C}(n)\log n + \mathsf{M}(n)\log D)$ opérations de $\mathbb{K}$.

Exercice 2 (Dérivation automatique et calcul d'inverse). Le but de cet exercice est de démontrer que le calcul de l'inverse d'une matrice n'est pas plus difficile que le calcul du déterminant, en utilisant des techniques de dérivation automatique.

Pour cela, il faut formaliser brièvement ce qu'on entend par *programme*. Dans notre contexte, un programme prend en entrée des indéterminées $Y_1, \ldots, Y_m$ sur

[1]On rappelle qu'une matrice $C = (c_{ij}) \in \mathcal{M}_n(\mathbb{K})$ est dite *compagnon* si ses $n-1$ premières colonnes ne contiennent que des éléments nuls, à l'exception des éléments $c_{2,1}, c_{3,2}, \ldots, c_{n,n-1}$ qui valent tous 1.

un corps $\mathbb{K}$, et effectue une suite d'*instructions* $g_1, \ldots, g_L$. Pour tout $i = 1, \ldots, L$, l'instruction g_i consiste simplement en la définition d'un polynôme G_i, de la forme

$$g_i : \ G_i = \mathsf{Arg}_i \operatorname{op}_i \mathsf{Arg}'_i,$$

selon les règles suivantes :

– Arg_i et Arg'_i sont dans $\mathbb{K} \cup \{G_1, \ldots, G_{i-1}\} \cup \{Y_1, \ldots, Y_m\}$.
– op_i est dans $\{+, -, \times\}$.

La *taille* d'un tel programme est L. On dit qu'un tel programme *calcule* un polynôme F si F appartient à l'ensemble $\{G_1, \ldots, G_L\}$; on dit de même que le programme calcule $\{F_1, \ldots, F_s\}$ si tous les F_i sont dans $\{G_1, \ldots, G_L\}$.

Par exemple, le programme ayant pour entrées Y_1, Y_2, Y_3, Y_4, pour instructions

$$\begin{array}{rcl} G_1 & = & Y_1 \times Y_4 \\ G_2 & = & Y_2 \times Y_3 \\ G_3 & = & G_1 - G_2 \end{array}$$

calcule le déterminant $Y_1Y_4 - Y_2Y_3$ de la matrice

$$\begin{pmatrix} Y_1 & Y_2 \\ Y_3 & Y_4 \end{pmatrix};$$

il a taille 3.

Si G est dans $\mathbb{K}[Y_1, \ldots, Y_m]$, on appelle *gradient* de G l'ensemble de ses m dérivées partielles $\partial G/\partial Y_i$, $i = 1, \ldots, m$.

1. Soient G et H dans $\mathbb{K}[Y_1, \ldots, Y_m]$. Exprimer les dérivées partielles de $G+H$, $G-H$ et $G \times H$, en fonction de G, H et des dérivées partielles de G et H.

   En déduire qu'étant donné un programme de taille L, qui calcule un polynôme G de $\mathbb{K}[Y_1, \ldots, Y_m]$, on peut en déduire un programme de taille $O(Lm)$ qui calcule G et son gradient.

2. Soient G dans $\mathbb{K}[Y_1, \ldots, Y_m]$ et H dans $\mathbb{K}[Y_1, \ldots, Y_{m+1}]$, tels que

   $$G = H(Y_1, \ldots, Y_m, Y_i \times Y_j),$$

   avec $1 \leq i, j \leq m$. Exprimer les dérivées partielles de G en fonction de H et de ses dérivées partielles. Traiter ensuite les cas

   $$G = H(Y_1, \ldots, Y_m, Y_i \pm Y_j).$$

3. Soient G dans $\mathbb{K}[Y_1, \ldots, Y_m]$ et H dans $\mathbb{K}[Y_1, \ldots, Y_{m+1}]$, tels que

   $$G = H(Y_1, \ldots, Y_m, Y_i \operatorname{op} Y_j),$$

   avec $1 \leq i, j \leq m$, et op dans $\{+, -, \times\}$. On suppose connu un programme de taille L qui calcule H et son gradient. Montrer qu'on peut en déduire un programme de taille $L + O(1)$ qui calcule G et son gradient.

   En déduire par récurrence que si un polynôme G de $\mathbb{K}[Y_1, \ldots, Y_m]$ peut être calculé par un programme de taille L, G et son gradient peuvent être calculés par un programme de taille $O(L)$.

4. Soient $X_{1,1}, \ldots, X_{1,n}, \ldots, X_{n,1}, \ldots, X_{n,n}$ des indéterminées, et soit D le déterminant de la matrice

   $$M = \begin{pmatrix} X_{1,1} & \ldots & X_{1,n} \\ \vdots & & \vdots \\ X_{n,1} & \ldots & X_{n,n} \end{pmatrix}.$$

Montrer que le gradient de D permet de retrouver les entrées de la *comatrice* N de M, où N est l'unique matrice satisfaisant

$$MN^t = N^tM = DI_n,$$

I_n étant la matrice identité de taille n, et N^t la transposée de N.

5. Montrer qu'étant donné un programme de taille L qui calcule D, on peut en déduire un programme de taille $O(L)$, effectuant éventuellement des divisions, qui calcule les coefficients de l'inverse de M.

Bibliographie

[1] Kaporin (I). – The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. *Theoretical Computer Science*, vol. 315, n° 2-3, 2004, pp. 469–510.

[2] Winograd (S.). – A new algorithm for inner-product. *IEEE Transactions on Computers*, vol. 17, 1968, pp. 693–694.

COURS 8

Solutions séries d'équations différentielles

Résumé

L'algorithme de Newton dans un cadre différentiel permet de calculer des séries tronquées solutions d'équations différentielles en complexité quasi-optimale. Pour certaines classes particulières importantes d'équations, la complexité peut être encore abaissée.

Comme pour le cours 3 sur les calculs rapides de séries, N sera utilisé pour représenter le nombre de termes à calculer, et « série » sera employé pour « série tronquée à l'ordre N », $\mathsf{M}(N)$ dénote une borne supérieure sur le nombre d'opérations arithmétiques nécessaires pour multiplier deux polynômes de degré au plus N (et par conséquent deux séries tronquées à l'ordre N). On suppose pour simplifier les expressions asymptotiques que la multiplication est plus coûteuse que l'addition, c'est-à-dire que $\mathsf{M}(N)/N$ tend vers l'infini avec N. L'efficacité des algorithmes sera mesurée par leurs complexités arithmétiques. La complexité est alors quasi-optimale lorsqu'elle est linéaire en N à des facteurs logarithmiques près. Nous nous attacherons par ailleurs à expliciter la dépendance de la complexité vis-à-vis des autres paramètres (ordre r de l'équation, degré d des coefficients lorsqu'ils sont polynomiaux). Pour le cas particulier très important des équations et des systèmes différentiels *linéaires*, les résultats de complexité de ce cours sont présentés sur le tableau 1, sur lequel il faut comparer les complexités aux tailles des entrées et des sorties. Par comparaison, la méthode la plus directe (les coefficients indéterminés), est quadratique en N pour le cas où les coefficients sont des séries.

Tous les algorithmes sont énoncés pour des coefficients dans un anneau $\mathbb{A}$ (avec en vue le cas de coefficients polynomiaux par exemple). Dans tous ce cours nous ferons *l'hypothèse supplémentaire que* $2, 3, \ldots, N$ *sont inversibles dans* $\mathbb{A}$.

Problème (entrée, sortie)	coefficients séries	coefficients polynômes	coefficients constants	taille résultat
(équation, base)	$O(r^2\mathsf{M}(N))$	$O(dr^2N)$	$O(rN)$	rN
(équation, une solution)	$O(r\,\mathsf{M}(N)\log N)$	$O(drN)$	$O(\mathsf{M}(r)N/r)$	N
(système, base)	$O(r^2\mathsf{M}(N))$	$O(dr^\omega N)$	$O(r\mathsf{M}(r)N)$	r^2N
(système, une solution)	$O(r^2\,\mathsf{M}(N)\log N)$	$O(dr^2N)$	$O(\mathsf{M}(r)N)$	rN

TAB. 1. Complexité de la résolution d'équations et de systèmes différentiels linéaires pour $N \gg r$.

1. Équations différentielles d'ordre 1

Le cas de l'ordre 1 est plus simple que le cas général. Sa présentation sert d'introduction à la section suivante, où les techniques employées ici seront généralisées.

1.1. L'équation linéaire homogène du premier ordre. Cette équation,

$$y' = A(X)y,$$

où $A(X)$ est une série, admet la solution

$$y(X) = y(0)\exp\left(\int A(X)\right). \tag{1}$$

Le calcul utilisant cette formule est dominé par celui de l'exponentielle, donc en $O(\mathsf{M}(N))$ par les algorithmes du cours 3.

1.2. L'équation linéaire inhomogène du premier ordre. Il s'agit de l'équation

$$y' = A(X)y + B(X),$$

où A et B sont des séries. La méthode de la variation de la constante consiste à poser

$$y(X) = f(X)\exp\left(\int A(X)\right),$$

et à injecter cette expression dans l'équation pour obtenir que f vérifie

$$f'(X) = B(X)\exp\left(-\int A(X)\right).$$

L'ensemble du calcul de y est donc encore borné par $O(\mathsf{M}(N))$ opérations.

1.3. Le cas non-linéaire. Ces préliminaires avaient pour but de prouver le résultat plus général suivant.

Proposition 1. *Soit $F(X,Y)$ une série bivariée telle que pour toute série $s(X)$, les N premiers termes de $F(X,s(X))$ et de $\frac{\partial F}{\partial y}(X,s(X))$ se calculent en $O(\mathsf{L}(\mathsf{N}))$ opérations. Alors, l'équation différentielle*

$$y' = F(X,y), \qquad y(0) = a,$$

admet une série formelle solution, et ses N premiers termes peuvent être calculés en $O(\mathsf{L}(N) + \mathsf{M}(N))$ opérations.

L'énoncé de cette proposition en terme d'une fonction L permet de l'adapter à différents besoins : dans le cas le pire, il est possible d'invoquer l'algorithme de Brent et Kung du cours 3 pour avoir $\mathsf{L}(N) = O(\sqrt{N\log N}\mathsf{M}(N))$ qui domine alors la complexité de la résolution. Cependant, pour des équations plus simples, la composition avec F et sa dérivée pourra être en $O(\mathsf{M}(N))$. Ce sera le cas par exemple si F s'obtient à l'aide d'un nombre constant de sommes, d'exponentielles, de logarithmes, et de puissances.

Démonstration. L'idée de la preuve repose sur la méthode de Newton, appliquée cette fois-ci à un opérateur :

$$\Phi : y \mapsto y' - F(X,y).$$

Le principe consiste à linéariser l'opérateur Φ au voisinage d'une approximation et à en annuler la partie linéaire, pour obtenir une nouvelle approximation. À partir de

$$\Phi(y+h) = \Phi(y) + h' - \frac{\partial F}{\partial y}(X,y)h + O(h^2),$$

il s'agit donc de calculer un incrément h solution de l'équation *linéaire* inhomogène du premier ordre

$$h' = \frac{\partial F}{\partial y}(X,y)h - \Phi(y),$$

qui peut être résolue par la méthode de la section précédente. On déduit l'itération suivante

$$y_{k+1} := y_k + h_k,$$
$$h_k := \exp(\int \frac{\partial F}{\partial y}(X,y_k)) \int (F(X,y_k) - y_k') \exp(-\int \frac{\partial F}{\partial y}(X,y_k)) \bmod x^{2^{k+1}}.$$

Il reste à prouver la convergence quadratique. Comme pour le théorème sur l'itération de Newton sur les séries du cours 3 (p. 31), la preuve se fait par récurrence sur k en prouvant simultanément $\Phi(y_k) = O(X^{2^k})$ et $h_k = O(X^{2^k})$. Les calculs sont les mêmes et sont donc laissés en exercice.

Pour obtenir le résultat de complexité, il suffit d'observer que l'itération requiert à chaque étape la résolution d'une équation différentielle linéaire inhomogène du premier ordre et d'invoquer le lemme « Diviser pour régner ». □

2. Équations différentielles linéaires d'ordre supérieur et systèmes d'ordre 1

L'équation différentielle d'ordre r

$$a_r(X)y^{(r)}(X) + \cdots + a_1(X)y'(X) + a_0(X)y(X) = 0 \tag{2}$$

où les coefficients a_i sont des séries en X peut être récrite comme une équation d'ordre 1

$$Y' = A(X)Y \tag{3}$$

en prenant pour Y le vecteur de séries $(y(X), \ldots, y^{(r-1)}(X))$ et A une matrice (compagnon) de séries en X.

À l'inverse, un système (3) induit une relation de dépendance linéaire à coefficients des séries entre $Y, Y', Y'', \ldots, Y^{(r)}$ si r est la dimension de A (on a $r+1$ vecteurs en dimension r). Le vecteur Y et chacun de ses coefficients vérifient donc une équation de type (2).

Résoudre un problème est donc équivalent à résoudre l'autre. Dans certains cas, exploiter le caractère compagnon de la matrice induite par (2) mène à une meilleure complexité pour le cas des équations.

Une différence importante avec le cas des équations d'ordre 1 est que nous avons, pour les équations de type (2) comme pour les systèmes (3) deux problèmes à considérer :

- Calculer une base des séries solutions ;
- étant données des conditions initiales, calculer *la* série solution correspondante.

Les complexités de ces problèmes sont liées : si l'on sait résoudre le second pour un coût C, alors on sait résoudre le premier pour un coût borné par $r\mathsf{C}$. Si l'on sait résoudre le premier, alors une combinaison linéaire des solutions permet de résoudre le second en au plus rN opérations supplémentaires.

Il est cependant utile de considérer les quatre problèmes (système ou équation, base ou solution unique) car la structure de chacun des problèmes permet de concevoir des algorithmes plus efficaces que n'en donnent les conversions directes d'un problème à l'autre.

2.1. Une méthode par diviser pour régner. L'équation à résoudre à précision X^N est ici

$$Y' - AY = B, \quad Y(0) = v,$$

où A est une matrice de séries formelles. L'idée est de résoudre d'abord à précision moitié et de déterminer puis résoudre l'équation satisfaite par le reste. La condition initiale est d'abord traitée séparément en écrivant $Y = v + XU$ et en injectant dans l'équation, ce qui mène à

$$XU' + (I - XA(X))U = C, \quad C = B + A(X)v.$$

Soit maintenant $d < N$ et $U = U_0 + X^d U_1$ où U_0 est n polynôme de degré au plus $d-1$ en X, l'équation satisfaite par U donne :

$$XU_1' + ((d+1)I - XA(X))U_1 = -X^{-d}(XU_0' + (I - XA(X))U_0 - C).$$

Si U_0 est une solution à précision d, le membre droit de l'équation est bien un polynôme. L'application du paradigme « diviser pour régner » amène donc naturellement à considérer l'équation plus générale

$$XY' + (pI - XA)Y = R,$$

où R est une série, A une matrice de séries et $p \in \{1, \dots, N\}$.

L'algorithme récursif est donc le suivant :

DivideAndConquer(A, p, s)

Input : $A_0, \dots, A_{N-p}$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $s_0, \dots, s_{N-p}$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $s = \sum s_i X^i$, $p \in \{1, \dots, N\}$.

Output : $y = \sum_{i=0}^{N-p} y_i X^i$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})[X]$ tel que $Xy' + (pI - XA)y = s + O(X^{N-p+1})$.

If $m = 1$ then return $p^{-1}s(0)$
else
  $m \leftarrow N - p$
  $d \leftarrow \lfloor m/2 \rfloor$
  $y_0 \leftarrow$ DivideAndConquer$(A, p + d, s \bmod X^d)$
  $R \leftarrow [s - Xy_0' - (pI - XA)y_0]_d^m$
  $y_1 \leftarrow$ DivideAndConquer$(A, p + d, R)$
  return $y_0 + X^d y_1$

La notation $[s]_d^m$ désigne le quotient de la division euclidienne de $s \bmod X^m$ par X^d.

Cet algorithme est invoqué par l'algorithme de résolution suggéré ci-dessus et dont voici une version détaillée.

SolvebyDAC(A, N, B, v)

Input : $A_0, \dots, A_N$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $B_0, \dots, B_N$ et v dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $B = \sum B_i X^i$.

Output : $y = \sum_{i=0}^{N} y_i X^i$ dans $\mathcal{M}_{r\times\ell}(\mathbb{A})[X]$ tel que $y' - Ay = B$ et $y(0) = v$.

return $v + X$DivideAndConquer$(A, 1, B + Av)$

Les résultats de complexité se déduisent de cette méthode pour différentes valeurs de A et de ℓ. En particulier, le cas des équations a une meilleure complexité que le cas général d'un système d'ordre 1 car la matrice correspondante est une matrice compagnon. Le produit d'une telle matrice par un vecteur ne coûte que r opérations au lieu de r^2 dans le cas général.

THÉORÈME 1 (Diviser pour régner pour les équations différentielles). *Étant donnés les N premiers coefficients de $A \in \mathcal{M}_{r\times r}(\mathbb{A}[[X]])$, de $B \in \mathcal{M}_{r\times\ell}(\mathbb{A}[[X]])$ ainsi que des conditions initiales $v \in \mathcal{M}_{r\times\ell}(\mathbb{A})$, l'algorithme* SolvebyDAC *calcule l'unique solution de*

$$Y' - AY = B + O(X^N), \qquad Y(0) = v$$

en un nombre d'opérations dans $\mathbb{A}$ borné par

1. $O(r^2 \ell \mathsf{M}(N) \log N)$ *en général ;*
2. $O(r\ell \mathsf{M}(N) \log N)$ *si A est une matrice compagnon.*

Il est possible d'améliorer les estimations de complexité pour cet algorithme lorsque $\ell = r$, mais dans ce cas du calcul de toute une base des solutions, il vaut mieux employer les algorithmes de la section suivante, qui sont plus efficaces.

DÉMONSTRATION. Il suffit de prouver le résultat pour N une puissance de 2, et le cas général s'en déduit par les hypothèses habituelles sur la fonction de multiplication M.

Les opérations de troncature ne demandent pas de calcul dans $\mathbb{A}$. Outre les deux appels récursifs, le calcul demande une dérivation (linéaire), un produit par p fois l'identité (linéaire) et un produit par A. La complexité $C(N)$ vérifie donc

$$C(N) = 2C(N/2) + \lambda \ell N + \mathsf{Mult}(A, V, N),$$

où $\mathsf{Mult}(A, V, N)$ désigne le coût du produit de la matrice A par la matrice $r \times \ell$ de séries à précision N. La conclusion s'obtient par le lemme « Diviser pour régner » en observant les complexités de base pour $\mathsf{Mult}(A, V, N)$. □

2.2. L'itération de Newton matricielle. Comme dans la preuve de la Proposition 1, le point de départ de l'algorithme consiste à appliquer l'itération de Newton à l'opérateur dont on cherche les solutions, en linéarisant au voisinage d'une solution approchée. L'opérateur

$$Y \mapsto Y' - A(X)Y$$

étant linéaire, il est son propre linéarisé. Formellement, l'itération de Newton s'écrit donc

$$Y_{k+1} = Y_k - U_k, \quad U_k' - AU_k = Y_k' - AY_k.$$

Lorsque Y_k est une solution approchée, le membre droit de l'équation en U_k a une valuation strictement positive, et la solution U_k cherchée doit avoir aussi une

SolveHomDiffSys(A, N, Y_0)

Input : $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$.

Output : $Y = \sum_{i=0}^{N-1} Y_i X^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$, et $Z = Y^{-1} + O(X^{N/2})$.

$Y \leftarrow (I + XA_0)Y_0$
$Z \leftarrow Y_0^{-1}$
$m \leftarrow 2$
while $m \leq N/2$ do
    $Z \leftarrow Z + Z(I_r - YZ) \bmod X^m$
    $Y \leftarrow Y - Y\left(\int Z(Y' - A \bmod X^{2m-1}Y)\right) \bmod X^{2m}$
    $m \leftarrow 2m$
return Y, Z

FIG. 1. Résolution de $Y' = A(X)Y$, $Y(0) = Y_0$ par itération de Newton

telle valuation. Une telle solution est obtenue par la méthode de la variation de la constante *si l'on dispose d'une base des solutions*, c'est-à-dire dans notre cas si Y_k est une matrice $r \times r$ avec $\det Y_k(0) \neq 0$. Dans ce cas, la méthode de la variation de la constante suggère de poser $U_k = Y_k T$, ce qui donne

$$U_k' - AU_k = Y_k T' + \underbrace{AY_k T - AY_k T}_{0} = Y_k' - AY_k$$

d'où finalement

$$U_k = Y_k \int Y_k^{-1}(Y_k' - AY_k).$$

Cette méthode requiert le calcul de l'inverse d'une base de solution, inverse qui peut être calculé simultanément par l'itération de Newton :

$$Z_{k+1} = Z_k + Z_k(I - Y_k Z_k).$$

Une version précise de l'algorithme, avec les ordres de troncatures, est donnée en Figure 1.

LEMME 1 (Correction de l'algorithme). *Soit m un entier pair, soient y et z dans $\mathcal{M}_{r\times r}(\mathbb{A}[X])$ tels que*

$$I - yz = O(X^{m/2}), \quad y' - Ay = O(X^m).$$

Soient ensuite Y et Z définis par

$$Z := z(2I - yz) \bmod X^m, \quad Y := y(I - \int Z(y' - Ay)) \bmod X^{2m}.$$

Alors Y et Z vérifient

$$I - YZ = O(X^m), \quad Y' - AY = O(X^{2m-1}).$$

DÉMONSTRATION. D'après l'hypothèse sur $y' - Ay = O(X^m)$, $Y = y + O(X^m)$ et donc la convergence de l'inversion est donnée par

$$\begin{aligned} I - YZ &= I - y(z + z(I - yz)) + O(X^m), \\ &= (I - yz) - yz(I - yz) + O(X^m), \\ &= (I - yz)^2 + O(X^m) = O(X^m). \end{aligned}$$

La seconde propriété s'obtient en injectant la définition de Y dans $Y' - AY$ (on pose $Q = \int Z(y' - Ay)$) :

$$\begin{aligned} Y' - AY &= y' + y'Q - Ay - AyQ - yZ(y' - Ay) + O(X^{2m}), \\ &= (I - yZ)(y' - Ay) - (y' - Ay)Q + O(X^{2m}), \\ &= O(X^m)O(X^m) + O(X^m)O(X^{m-1}) + O(X^{2m}) = O(X^{2m-1}). \end{aligned}$$

□

Comme d'habitude, l'application du lemme « Diviser pour régner » mène au résultat de complexité.

THÉORÈME 2 (Newton pour les systèmes différentiels linéaires). *L'algorithme* SolveHomDiffSys *calcule les N premiers termes d'une base de solutions de $Y' = AY$ en $O(\mathsf{MM}(r, N))$ opérations dans $\mathbb{A}$.*

La notation $\mathsf{MM}(r, N)$ représente la complexité du produit de matrices $r \times r$ de polynômes de degré au plus N ; des bornes non triviales ont été données au cours 7 :

$$\mathsf{MM}(r, N) = O(r^\omega N + r^2 \mathsf{M}(N)).$$

EXERCICE 1. L'exponentielle d'une série est obtenue en $O(\mathsf{M}(N))$ opérations par cet algorithme lorsque $r = 1$. Vérifier que la constante est meilleure que celle de l'algorithme du cours 3.

EXERCICE 2. Le cas d'un système *inhomogène* $Y' = AY + B$ où B est une matrice de séries s'obtient à partir de l'algorithme précédent par variation de la constante. Étudier les précisions requises dans les troncatures, et donner le résultat de complexité.

3. Cas particuliers

3.1. Équations différentielles linéaires à coefficients polynomiaux. Pour ces équations, la méthode des coefficients indéterminés donne une complexité linéaire en N. L'explication tient à une propriété classique : les solutions séries de ces équations ont des coefficients qui vérifient des récurrences linéaires. En effet, si $A = a_0 + a_1 X + \cdots$ est une série solution de

$$q_0(X)A^{(r)} + \cdots + q_r(X)A = 0,$$

alors en notant $[X^n]f(X)$ le coefficient de X^n dans la série $f(X)$, on a les relations

$$[X^n]f'(X) = (n+1)[X^{n+1}]f(X), \qquad [X^n]X^k f(X) = [X^{n-k}]f(X).$$

Par conséquent, l'extraction du coefficient de X^n de l'équation différentielle fournit une récurrence linéaire sur les a_n valide dès lors que $n \geq n_0 := \max_{0 \leq i \leq r} \deg q_i$.

Les résultats du cours 6 sur les récurrences linéaires à coefficients polynomiaux s'appliquent alors, et en particulier les N premiers coefficients d'une solution s'obtiennent en $O(drN)$ opérations si d est une borne sur le degré des q_i.

Le cas matriciel se traite de la même façon, la récurrence ci-dessus étant alors à coefficients matriciels. Les complexités sont de même nature, avec en outre la possibilité d'utiliser un produit rapide de matrices dans le cas du calcul d'une base de solutions. Les résultats correspondants sont donnés dans la table 1.

3.2. Équations différentielles linéaires à coefficients constants. Dans le cas où les coefficients de l'équation ou du système sont constants, la formule (1) s'applique encore et devient :

$$y(X) = \exp(XA)y(0).$$

Pour étudier les propriétés de cette solution, il est usuel de faire intervenir la forme de Jordan de la matrice, mais cette forme ne se prête pas facilement au calcul.

Un algorithme efficace est obtenu en exploitant la *transformée de Laplace formelle*. Si $S = s_0 + s_1X + s_2X^2 + \dots$ est une série, cette transformée est définie par

$$\mathcal{L}S = s_0 + 1!s_1X + 2!s_2X^2 + \dots.$$

Les troncatures de $\mathcal{L}S + O(X^N)$ et de la transformée inverse $\mathcal{L}^{-1}S + O(X^N)$ se calculent en N opérations. Cette transformation simplifie les systèmes différentiels linéaires à coefficients constants en les rendant linéaires non-différentiels. Ceci découle de

$$\begin{aligned}\mathcal{L}(y) &= \mathcal{L}((I + XA + \frac{1}{2!}X^2A^2 + \dots)y(0)) \\ &= (I + XA + X^2A^2 + \dots)y(0) \\ &= (I - XA)^{-1}y(0).\end{aligned}$$

Les vecteurs solutions ont donc des transformées de Laplace dont les coordonnées sont rationnelles. En outre, d'après les formules de Cramer, le numérateur et le dénominateur de ces fractions ont des degrés bornés par l'ordre r du système. L'algorithme suivant s'en déduit :

ConstantCoefficients(A, v, N)

Input : A in $\mathcal{M}_{r\times r}(\mathbb{A})$, $v \in \mathcal{M}_{r\times 1}(\mathbb{A})$.

Output : $Y = \sum_{i=0}^{N-1} Y_iX^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$.

1. Calculer les vecteurs $v, Av, A^2v, A^3v, \dots, A^{2r}v$;
2. Pour tout $j = 1, \dots, r$:
   (a) reconstruire la fraction rationnelle ayant pour développement en série $\sum (A^iv)_jX^i$;
   (b) développer cette fraction en série à précision X^N en $O(N\,\mathsf{M}(r)/r)$ opérations ;
   (c) reconstruire le développement de y à partir de celui de z, en $O(N)$ opérations.

L'étape 1 est effectuée par la méthode naïve en $O(r^3)$ opérations. Une méthode plus efficace, en $O(r^\omega \log r)$ opérations, est à la base d'un algorithme de calcul du polynôme minimal d'une matrice et sera présentée au cours 26. L'étape 2 (a) se ramène à un calcul d'algèbre linéaire en posant des coefficients indéterminés pour les numérateurs et dénominateurs. Là aussi le coût peut-être diminué en faisant appel au calcul d'approximants de Padé qui sera vu au cours 10, mais cette partie du calcul ne dépend pas de N. L'étape 2 (b) utilise l'algorithme de développement

de fractions rationnelles vu au cours 4, et c'est là que se concentre le gain en complexité.

4. Extensions

4.1. Composer se ramène à résoudre. Le seul algorithme du cours sur les calculs rapides de séries dont la complexité n'est pas quasi-optimale est l'algorithme de composition.

Dans les applications où l'on connaît une équation $\Phi(x, f, f', \dots, f^{(m)}) = 0$ vérifiée par la série f il est parfois possible de calculer la série $h = f \circ g$ efficacement *sans passer par l'algorithme de composition* en remarquant qu'elle vérifie une équation du même ordre. Cette équation est obtenue en remplaçant x par g dans Φ et en composant les dérivées. Si Φ ne faisait intervenir que des polynômes ou des fractions rationnelles, le résultat de cette opération a pour coefficients des séries, ce qui mène assez généralement à une complexité en $O(\mathsf{M}(N))$ opérations en utilisant les algorithmes de ce cours.

Exemple 1. Pour calculer le développement de $h = \tan(f)$, il est possible de calculer en $O(\mathsf{M}(N))$ ceux de $\sin(f)$ et $\cos(f)$ (via l'exponentielle) et de diviser, mais il est aussi possible d'observer que h vérifie l'équation $h' = 1 + h^2 f'$, et d'utiliser les méthodes ci-dessus. Il faut ensuite comparer les constantes dans les $O()$ (ou deux implantations !) pour déterminer laquelle des deux méthodes est préférable.

4.2. Systèmes non-linéaires. La linéarisation effectuée dans le cas des équations d'ordre 1 se généralise aux systèmes. L'énoncé devient alors le suivant.

Théorème 3. *Soient $\phi_1, \dots, \phi_r$ r séries de $\mathbb{A}[[X, Y_1, \dots, Y_r]]$, telles que pour tout r-uplet de séries $(s_1(X), \dots, s_r(X)) \in \mathbb{A}[[X]]^r$, les N premiers termes des $\phi_i(X, s_1, \dots, s_r)$ et de $\mathrm{Jac}(\phi)(X, s_1(X), \dots, s_r(X))$ puissent être calculés en $O(\mathsf{L}(\mathsf{N}))$ opérations dans $\mathbb{A}$. On suppose en outre que $\mathsf{L}(\mathsf{n})/\mathsf{n}$ est une suite croissante. Alors, le système différentiel*

$$\begin{cases} y_1'(t) = \varphi_1(t, y_1(t), \dots, y_r(t)), \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ y_r'(t) = \varphi_r(t, y_1(t), \dots, y_r(t)), \end{cases}$$

avec conditions initiales $(y_1, \dots, y_r)(0) = v$ admet une solution série formelle, et ses N premiers termes peuvent être calculés en

$$O(\mathsf{L}(\mathsf{N}) + \min(\mathsf{MM}(\mathsf{r}, \mathsf{N}), \mathsf{r}^2\mathsf{M}(\mathsf{N}) \log \mathsf{N})).$$

Le symbole $\mathrm{Jac}(\phi)$ désigne la matrice jacobienne : son coefficient sur la ligne i et la colonne j vaut $\partial\phi_i/\partial y_j$.

L'intérêt d'énoncer le théorème à l'aide de la fonction L est le même que dans le cas des équations.

Démonstration. Il s'agit d'une généralisation de la Proposition 1. La linéarisation de $\Phi : Y \mapsto Y' - \varphi(Y)$ au voisinage d'une solution approchée y s'obtient en écrivant :

$$\Phi(y + h) = \Phi(y) + h' + \mathrm{Jac}(\phi)(y)h + O(h^2).$$

L'équation à résoudre pour une itération de Newton est donc le système linéaire inhomogène

$$h' = \mathrm{Jac}(\phi)(y)h - (y' - \phi(y)).$$

Si h est un vecteur solution de ce système à précision $2m$ et y une solution de Φ à précision m, ces équations montrent que $y + h$ est solution à précision $2m$. Le cas non-linéaire est ainsi ramené au cas linéaire. □

Notes

Les résultats de la Section 1 sont tirés de [2]. Une bonne partie des résultats de ce cours est tirée de [1]. La technique de la section 2.2 dans le cas particulier de l'exponentielle d'une série est due à [3]. L'algorithme diviser pour régner de la section 2.1 se trouve au moins implicitement dans [4].

Bibliographie

[1] Bostan (Alin), Chyzak (Frédéric), Ollivier (François), Salvy (Bruno), Schost (Éric), and Sedoglavic (Alexandre). – Fast computation of power series solutions of systems of differential equations. In *SODA'07*. – *Toappear*.

[2] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n° 4, 1978, pp. 581–595.

[3] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Applicable Algebra in Engineering, Communication and Computing*, vol. 14, n° 6, March 2004, pp. 415–438.

[4] van der Hoeven (Joris). – Relax, but don't be too lazy. *Journal of Symbolic Computation*, vol. 34, n° 6, 2002, pp. 479–542.

COURS 9

Principe de Tellegen

COURS 10

Pgcd, résultant, et approximants de Padé

Résumé

L'algorithme d'Euclide classique permet de calculer le pgcd et le pgcd étendu. Il est relié aux résultants qui permettent un calcul d'élimination. Les calculs efficaces de ces objets seront détaillés dans un cours ultérieur. Ils sont quasi-optimaux dans le cas univarié.

Les calculs de pgcd sont cruciaux pour la simplification des fractions, qu'il s'agisse de fractions d'entiers ou de fractions de polynômes. Les algorithmes efficaces de factorisation de polynômes reposent par ailleurs de manière essentielle sur le pgcd de polynômes. Comme pour la multiplication, les algorithmes efficaces de pgcd sont plus complexes dans le cas des entiers que dans le cas des polynômes à cause des retenues et nous ne détaillons que la version polynomiale. Dans les définitions et résultats de nature moins algorithmique, nous utilisons le cadre algébrique des anneaux euclidiens qui permet de traiter simultanément toutes les applications que nous avons en vue.

L'algorithme d'Euclide pour le calcul du pgcd est présenté en section 1. Dans le cas de polynômes de $\mathbb{K}[X]$, sa complexité est quadratique en nombre d'opérations dans le corps $\mathbb{K}$. Le résultant est également lié à l'algorithme d'Euclide, ses propriétés et son calcul sont présentés en section 2. En outre, une technique dite « des sous-résultants » permet de réduire l'explosion de la taille des coefficients intermédiaires dont souffre l'algorithme d'Euclide pour le pgcd dans $\mathbb{Q}[X]$. Le cours se termine en Section 3 sur un algorithme plus moderne, de complexité quasi-optimale, exploitant la multiplication rapide. Ces algorithmes, à base d'approximants de Padé, permettent aussi de calculer la *reconstruction rationnelle*, grâce à laquelle on peut retrouver une récurrence linéaire à coefficients constants d'ordre d à partir de $2d$ termes consécutifs d'une solution.

Dans ce cours, $\mathbb{A}$ désignera toujours un anneau intègre (commutatif et sans diviseurs de zéro) et unitaire.

1. Algorithme d'Euclide

1.1. Le pgcd. Des pgcd peuvent être définis dans tout anneau intègre $\mathbb{A}$: on appelle *plus grand diviseur commun* (pgcd) de A et B tout $G \in \mathbb{A}$ qui divise A et B et tel que tout diviseur de A et de B divise aussi G. Contrairement à l'usage, nous notons dans ce cours $\mathrm{pgcd}(A, B)$ tout pgcd de A et de B sans faire de choix de normalisation parmi les pgcd de A et de B.

L'algorithme d'Euclide présenté dans cette section permet le calcul du pgcd dans $\mathbb{Z}$ ou dans l'anneau $\mathbb{K}[X]$, où $\mathbb{K}$ est un corps. Il repose sur l'existence d'une *division euclidienne* dans ces anneaux. Un anneau intègre $\mathbb{A}$ est appelé *anneau euclidien* s'il existe une fonction de taille $d : \mathbb{A} \to \mathbb{N} \cup \{-\infty\}$ telle que pour tout $a \in$

Euclide(A,B)
Entrée : A et B dans $\mathbb{A}$.
Sortie : Un pgcd de A et B.

1. $R_0 := A$; $R_1 := B$; $i := 1$.
2. Tant que R_i est non nul, faire :
   $R_{i+1} := R_{i-1} \bmod R_i$
   $i := i + 1$
3. Renvoyer R_{i-1}.

FIG. 1. L'algorithme d'Euclide

$\mathbb{A}$, $b \in \mathbb{A} \setminus \{0\}$, il existe q et r dans $\mathbb{A}$ avec

$$a = qb + r, \qquad d(r) < d(b).$$

En particulier, $d(0) < d(b)$ dès que b est non nul. Dans le cas des entiers, la valeur absolue fournit une telle fonction d, et le degré remplit ce rôle pour les polynômes. La notation $r := a \bmod b$ sert dans les deux cas à définir r à partir de a et de b. L'entier ou le polynôme r s'appelle *le reste*.

EXERCICE 1. Un élément d'un anneau est dit irréductible si les produits qui lui sont égaux font tous intervenir un élément inversible. Tout anneau euclidien est *factoriel* (c'est-à-dire que tout élément non nul y a une factorisation unique en irréductibles, à l'ordre près de la factorisation et à des multiplications près des facteurs par des inversibles de l'anneau). Le pgcd se lit aussi sur les factorisations. Si A et B se factorisent sous la forme

$$A = am_1^{k_1} \cdots m_s^{k_s}, \qquad B = bm_1^{\ell_1} \cdots m_s^{\ell_s},$$

où a et b sont inversibles dans $\mathbb{A}$, les m_i sont irréductibles et les k_i et ℓ_i sont des entiers positifs ou nuls, montrer qu'alors un pgcd de A et B est

$$G = m_1^{\min(k_1,\ell_1)} \cdots m_s^{\min(k_s,\ell_s)}.$$

Autrement dit, on « met » dans G les facteurs irréductibles communs de A et B, avec la plus grande multiplicité possible. La factorisation dans $\mathbb{Z}$ ou dans $\mathbb{K}[X]$ est plus coûteuse que le pgcd et cette propriété n'est donc pas utilisée pour le calcul du pgcd.

1.2. Calcul du pgcd. Étant donnés A, B dans un anneau euclidien $\mathbb{A}$, l'algorithme d'Euclide (Fig. 1) calcule une suite de restes successifs dont la taille décroît, jusqu'à atteindre le pgcd.

La terminaison provient de la décroissance stricte de la taille à chaque étape. La correction de cet algorithme se déduit de la relation

$$\operatorname{pgcd}(F, G) = \operatorname{pgcd}(H, G) \qquad \text{pour} \qquad H := F \bmod G,$$

dont la preuve est laissée en exercice. Par récurrence, il s'ensuit que $\operatorname{pgcd}(F, G) = \operatorname{pgcd}(R_i, R_{i+1})$ pour tout i. Si en outre R_{i+1} est nul, alors $\operatorname{pgcd}(R_i, R_{i+1}) = R_i$, ce qui prouve la correction.

EXEMPLE 1. Soient $A = X^4 - 13X^3 + 2X^2 - X - 1$ et $B = X^2 - X - 1$ dans $\mathbb{Q}[X]$. La suite des restes est :

$$\begin{aligned} R_0 &= X^4 - 13X^3 + 2X^2 - X - 1, \\ R_1 &= X^2 - X - 1, \\ R_2 &= -22X - 10, \\ R_3 &= -41/121, \\ R_4 &= 0, \end{aligned}$$

de sorte que $-41/121$ est un pgcd de A et B et donc 1 aussi.

PROPOSITION 1. *L'algorithme d'Euclide calcule un pgcd de A et B dans $\mathbb{K}[X]$ en $O(\deg A \deg B)$ opérations dans $\mathbb{K}$.*

DÉMONSTRATION. La correction a été prouvée dans le cas général. Pour l'étude de complexité, nous supposons d'abord que $\deg A \geq \deg B$. D'après le cours 3 (p. 1.1), le calcul de $P \bmod Q$ peut être effectué en $2 \deg Q(\deg P - \deg Q + 1)$ opérations de $\mathbb{K}$. Il s'ensuit que le coût de l'algorithme d'Euclide est borné par la somme des $2\deg(R_i)(\deg R_{i-1} - \deg R_i + 1)$, pour $i \geq 1$. Tous les $\deg(R_i)$ sont majorés par $\deg A$, de sorte que le coût est borné par $2 \deg A \sum_{i \geq 1}(\deg R_{i-1} - \deg R_i + 1) = 2 \deg A(\deg R_0 + \deg B) = O(\deg A \deg B)$.

Si le degré de B et supérieur à celui de A, la première étape ne coûte pas d'opération arithmétique, et la borne ci-dessus s'applique ensuite au reste du calcul. □

La borne de complexité quadratique reflète bien le comportement de l'algorithme : dans une exécution typique de l'algorithme, les quotients ont degré 1 à chaque itération, les degrés des restes successifs diminuent de 1 à chaque itération et leur calcul est linéaire ; le nombre de coefficients intermédiaires calculés est quadratique et ils sont calculés aussi efficacement qu'il est possible par un algorithme qui les calcule tous.

1.3. Pgcd étendu et inversion modulaire.

Relation de Bézout. Une relation de la forme $G = UA + VB$ qui donne le pgcd G de deux polynômes A et B avec deux cofacteurs polynomiaux U et V est appelée une *relation de Bézout*. L'algorithme d'Euclide étendu est une modification légère de l'algorithme d'Euclide qui calcule non seulement le pgcd, mais aussi une relation de Bézout particulière,

$$UA + VB = G, \quad \text{avec } d(UG) < d(B) \text{ et } d(VG) < d(A). \tag{1}$$

Une fois le pgcd G choisi, les cofacteurs U et V sont rendus uniques par la contrainte sur les degrés. Dans de nombreuses applications, c'est davantage de ces *cofacteurs* U et V que du pgcd lui-même dont on a besoin. On parle alors de calcul de pgcd *étendu*. Ce calcul intervient par exemple de manière importante dans les algorithmes de factorisation de polynômes dans $\mathbb{Z}[X]$ ou $\mathbb{Q}[X]$ par des techniques de type « Hensel » (cours 29), et dans le développement rapide des séries algébriques (cours 12).

Les calculs de pgcd étendu permettent d'effectuer des *inversions modulaires*. Lorsque A et B sont premiers entre eux ($G \in \mathbb{A}$), alors l'élément V est un inverse de B modulo A.

Exemple 2. La relation de Bézout pour $a+bX$ et $1+X^2$ s'écrit :

$$(a-bX)(a+bX)+b^2(1+X^2)=a^2+b^2.$$

L'inverse de $B=a+bX$ modulo $A=1+X^2$ vaut donc

$$V=\frac{a-bX}{a^2+b^2}.$$

Puisque le corps des complexes s'obtient par le quotient $\mathbb{R}[X]/(X^2+1)$, cette relation n'est autre que la formule d'inversion familière

$$\frac{1}{a+ib}=\frac{a-ib}{a^2+b^2}.$$

Si $A\in\mathbb{A}$ est irréductible, $\mathbb{A}/(A)$ est un corps et le calcul de la relation de Bézout permet d'y effectuer la division : si $B\neq 0 \mod A$ alors $N/B=NV \bmod A$.

Exemple 3. La « simplification » de la fraction rationnelle

$$r=\frac{\phi^4-\phi+1}{\phi^7-1}=\frac{25\phi-34}{169}$$

où ϕ est le « nombre d'or », défini par $A(\phi)=0$ pour $A(X)=X^2-X-1$, est obtenue par les trois calculs suivants :

- calcul du reste $U=13X+7$ par $U:=X^7-1 \bmod A$,
- détermination de la relation de Bézout

$$\frac{13X-20}{169}(13X+7)-A=1,$$

- calcul du reste $V=25X-34$ par $V:=(13X-20)(X^4-X+1) \bmod A$.

Plus généralement, dans le cas où $A\in\mathbb{K}[X]$ est un polynôme irréductible, ces calculs montrent que le corps $\mathbb{K}(\alpha)$ des fractions rationnelles en α racine de A est un espace vectoriel dont une base est $1,\alpha,\alpha^2,\ldots,\alpha^{\deg A-1}$. Les algorithmes d'Euclide et d'Euclide étendu fournissent un moyen de calcul dans cette représentation. Il est ainsi possible de manipuler de manière exacte les racines d'un polynôme de degré arbitraire sans « résoudre ».

Exemple 4. Lorsque l'élément $A\in\mathbb{A}$ n'est pas irréductible, $\mathbb{A}/(A)$ n'est pas un corps. Cependant, les éléments inversibles peuvent y être inversés par le même calcul que ci-dessus. Lorsqu'un élément B non nul n'est pas inversible, la relation de Bézout se produit avec un G différent de 1. Il est alors possible de tirer parti de cette information (un facteur de A) en scindant le calcul d'une part sur G et d'autre part sur B/G.

Exemple 5. Le même calcul qu'à l'exemple 3 fonctionne sur des entiers :

$$r=\frac{25}{33}\equiv 5 \bmod 7$$

se déduit de

$$33 \bmod 7=5,\qquad 3\times 5-2\times 7=1,\qquad 3\times 25 \bmod 7=5.$$

Euclide étendu(A,B)

Entrée : A et B dans $\mathbb{A}$.
Sortie : Un pgcd de A et B et les cofacteurs U et V de l'identité de Bézout $UA + VB = G$.

1. $R_0 := A$; $U_0 := 1$; $V_0 := 0$; $R_1 := B$; $U_1 := 0$; $V_1 := 1$; $i := 1$.
2. Tant que R_i est non nul, faire :
   $R_{i+1} := QR_i + R_{i-1}$ la division euclidienne ;
   $U_{i+1} := U_{i-1} - QU_i$; $V_{i+1} := V_{i-1} - QV_i$;
   $i := i + 1$
3. Renvoyer $R_{i-1}, U_{i-1}, V_{i-1}$.

FIG. 2. L'algorithme d'Euclide étendu

Calcul des cofacteurs. L'idée-clé est de suivre pendant l'algorithme d'Euclide la décomposition de chacun des R_i sur A et B. Autrement dit, pour tout i, l'algorithme calcule des éléments U_i et V_i tels que

$$U_i A + V_i B = R_i,$$

dont les tailles sont bien contrôlées. Pour $i = 0$, il suffit de poser $U_0 = 1, V_0 = 0$, ce qui correspond à l'égalité

$$1 \cdot A + 0 \cdot B = A = R_0.$$

Pour $i = 1$, l'égalité

$$0 \cdot A + 1 \cdot B = B = R_1$$

est obtenue avec $U_1 = 0, V_1 = 1$. Ensuite, la division euclidienne

$$R_{i-1} = QR_i + R_{i+1}$$

donne la relation

$$R_{i+1} = R_{i-1} - QR_i = (U_{i-1} - QU_i)A + (V_{i-1} - QV_i)$$

qui pousse à définir U_{i+1} par $U_{i-1} - QU_i$ et V_{i+1} par $V_{i-1} - QV_i$, et à partir de laquelle une preuve par récurrence montre que les conditions de tailles de la relation de Bézout sont satisfaites.

L'algorithme est résumé en Figure 2. À nouveau, dans le cas des polynômes ou des entiers, la complexité est quadratique.

2. Résultant

2.1. Matrice de Sylvester. L'algorithme d'Euclide pour *deux* polynômes A, B à *une* variable est étroitement relié à la matrice de Sylvester. Si

$$A = a_m X^m + \cdots + a_0, \quad B = b_n X^n + \cdots + b_0,$$

appartiennent à $\mathbb{A}[X]$, cette matrice carrée s'écrit

$$\mathsf{Syl}(A,B) = \begin{pmatrix} a_m & a_{m-1} & \dots & a_0 & & \\ & a_m & a_{m-1} & \dots & a_0 & \\ & & \dots & \dots & & \\ & & a_m & a_{m-1} & \dots & a_0 \\ b_n & b_{n-1} & \dots & b_0 & & \\ & b_n & b_{n-1} & \dots & b_0 & \\ & & \dots & \dots & & \\ & & b_n & b_{n-1} & \dots & b_0 \end{pmatrix}$$

avec les n premières lignes contenant les coefficients de A et les m suivantes contenant les coefficients de B.

La transposée de cette matrice représente l'application linéaire

$$(U,V) \mapsto AU + BV,$$

en se limitant aux paires (U,V) tels que $\deg U < \deg B$ et $\deg V < \deg A$, et en utilisant la base des monômes $1, X, \dots, X^{m+n-1}$ à l'arrivée. Les combinaisons linéaires des lignes de la matrice de Sylvester donnent donc les coefficients des polynômes qui peuvent être obtenus comme image. Au vu de la relation de Bézout (1), le pgcd de A et B peut être obtenu ainsi et il est caractérisé comme l'élément de plus petit degré qui peut l'être. Ceci permet de le lire sur une forme *échelonnée en ligne* de la matrice.

Definition 1. Une matrice est en forme *échelonnée en ligne* lorsque

1. les lignes nulles sont toutes en dessous des lignes non nulles,
2. le premier coefficient non nul de chaque ligne non nulle est à droite du premier coefficient non nul de la ligne précédente, les coefficients des lignes suivantes dans sa colonne étant nuls.

L'algorithme classique d'élimination de Gauss sans pivot sur les colonnes calcule une forme échelonnée en ligne d'une matrice, dont les lignes non nulles sont une base de l'espace vectoriel engendré par les lignes de la matrice de départ. En conclusion, nous avons obtenu une relation entre la matrice de Sylvester et le pgcd.

Proposition 2. *Soient A, B deux polynômes de $\mathbb{K}[X]$. Alors la forme échelonnée en ligne de la matrice de Sylvester $\mathsf{Syl}(A,B)$ contient sur sa dernière ligne non nulle les coefficients d'un pgcd de A et B. En outre, la dimension du noyau de cette matrice est le degré du pgcd.*

Exemple 6. Pour les polynômes $A = X^4 - X^3 - 7X^2 + 2X + 3$ et $B = X^3 - 4X^2 + 2X + 3$, la matrice de Sylvester et sa forme échelonnée en ligne sont :

$$\begin{pmatrix} 1 & -1 & -7 & 2 & 3 & 0 & 0 \\ 0 & 1 & -1 & -7 & 2 & 3 & 0 \\ 0 & 0 & 1 & -1 & -7 & 2 & 3 \\ 1 & -4 & 2 & 3 & 0 & 0 & 0 \\ 0 & 1 & -4 & 2 & 3 & 0 & 0 \\ 0 & 0 & 1 & -4 & 2 & 3 & 0 \\ 0 & 0 & 0 & 1 & -4 & 2 & 3 \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} 1 & -1 & -7 & 2 & 3 & 0 & 0 \\ 0 & 1 & -1 & -7 & 2 & 3 & 0 \\ 0 & 0 & 1 & -1 & -7 & 2 & 3 \\ 0 & 0 & 0 & -14 & 45 & -3 & -18 \\ 0 & 0 & 0 & 0 & -\frac{5}{7} & \frac{12}{7} & \frac{9}{7} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{10} & -\frac{3}{10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

La dernière ligne non nulle donne le pgcd $X - 3$.

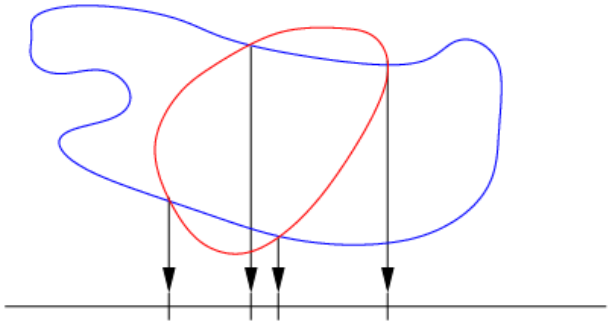


FIG. 3. Le résultant calcule des projections

DEFINITION 2. Le *résultant* de A et B est le déterminant de la matrice de Sylvester $\mathsf{Syl}(A, B)$. Il est noté $\mathrm{Res}(A, B)$ ou $\mathrm{Res}_X(A, B)$ si l'on veut insister sur l'élimination de la variable X.

Le résultant peut être vu comme une condition de cohérence pour le système formé par les deux polynômes A et B.

COROLLAIRE 1. *Soient A et B deux polynômes de $\mathbb{K}[X]$. Alors A et B sont premiers entre eux si et seulement si* $\mathrm{Res}(A, B) \neq 0$.

DÉMONSTRATION. Le déterminant d'une matrice carrée se lit sur la forme échelonnée en ligne en faisant le produit des éléments diagonaux. Il est non nul si et seulement si tous les éléments diagonaux le sont et dans ce cas un pgcd non nul est sur la dernière ligne. À l'inverse, s'il existe un pgcd G non nul, alors les polynômes $X^k G$ montrent l'existence de lignes avec un coefficient de tête non nul dans chaque colonne de la forme échelonnée. □

EXEMPLE 7. Le discriminant de A est par définition le résultant de A et sa dérivée A'. Il s'annule lorsque ces deux polynômes ont une racine commune dans une clôture algébrique de A, c'est-à-dire, en caractéristique nulle, lorsque A a une racine multiple.

EXERCICE 2. Calculer le discriminant de aX^2+bX+c en prenant le déterminant de la matrice de Sylvester.

2.2. Applications du résultant.

Calculs de projections. Algébriquement, la « résolution » d'un système polynomial se ramène souvent à une question d'*élimination*. Lorsqu'elle est possible, l'élimination successive des variables amène le système d'entrée sous une forme triangulaire. De proche en proche, la résolution d'un tel système se réduit alors à la manipulation de polynômes à une variable, pour lesquels compter, isoler, ... , les solutions est bien plus facile.

Géométriquement, l'élimination correspond à une projection. Cette section montre comment le résultant de deux polynômes permet de traiter ce genre de problèmes, pour les systèmes de deux équations en deux inconnues. Le schéma général est représenté en Figure 3. Cette opération est à la base d'une technique

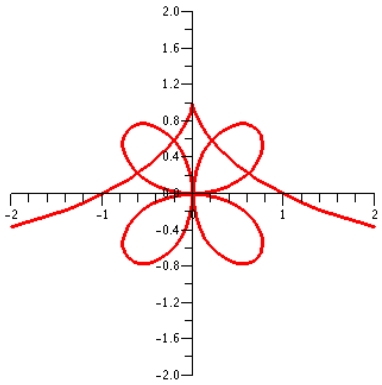


FIG. 4.

plus générale pour un nombre arbitraire d'équations et d'inconnues, la *résolution géométrique*, qui sera présentée au cours 20.

EXEMPLE 8. Les deux courbes de la Figure 4 ont pour équations

$$A = (X^2 + Y^2)^3 - 4X^2Y^2 = 0, \qquad B = X^2(1+Y) - (1-Y)^3 = 0.$$

Ces deux polynômes sont irréductibles et leur pgcd qui vaut 1 ne renseigne pas sur leurs racines communes. Les résultants par rapport à X et Y donnent en revanche des polynômes qui s'annulent sur les coordonnées de ces points d'intersection :

$$\mathrm{Res}_X(A,B) = (4Y^7 + 60Y^6 - 152Y^5 + 164Y^4 - 95Y^3 + 35Y^2 - 9Y + 1)^2,$$
$$\mathrm{Res}_Y(A,B) = 16X^{14} + 6032X^{12} - 1624X^{10} + 4192X^8 - 815X^6 - 301X^4 - 9X^2 + 1.$$

Il n'y a que 4 points d'intersection visibles sur la figure, les 10 autres ont au moins une coordonnée qui n'est pas réelle. Le caractère bicarré du résultant en Y provient de la symétrie de la figure par rapport à l'axe des Y. C'est pour cette même raison que le premier résultant est un carré.

Le résultat général est le suivant.

PROPOSITION 3. *Soit $A = a_m Y^m + \cdots$ et $B = b_n Y^n + \cdots$ où les coefficients a_i et b_j sont dans $\mathbb{K}[X]$. Alors les racines du polynôme $\mathrm{Res}_Y(A,B) \in \mathbb{K}[X]$ sont d'une part les abscisses des solutions du système $A = B = 0$, d'autre part les racines communes de a_m et b_n.*

DÉMONSTRATION. Ce résultat est une conséquence immédiate du théorème 1 de la section 2.3 ci-dessous. □

Graphiquement, les racines « dégénérées » du second cas se traduisent par la présence d'asymptotes verticales.

EXEMPLE 9. Avec $A = X^2Y + X + 1$, $B = XY - 1$, on obtient $\mathrm{Res}_Y(A,B) = -X(2X+1)$ (asymptote en $X = 0$, « vraie » solution en $X = -\frac{1}{2}$). Les courbes correspondantes sont représentées en Figure 5.

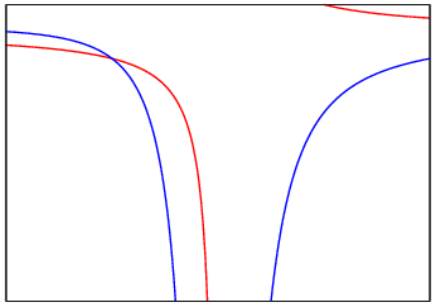


FIG. 5.

Une fois calculé le résultant donnant les coordonnées des abscisses des intersections des deux courbes, il est souhaitable d'obtenir les ordonnées correspondantes. Dans le cas simple où l'avant dernier reste de l'algorithme d'Euclide est de degré 1, il donne une telle paramétrisation.

EXEMPLE 10. Sur les deux polynômes A et B de l'exemple 8, vus comme polynômes dans $\mathbb{Q}(X)[Y]$, la suite des restes est

$$(X^4 - 13X^2 + 15)Y^2 - 4(X^2+2)(X^2+3)Y + (X^6 - 2X^4 - 8X^2 + 10),$$
$$(4X^8 - 344X^6 - 1243X^4 - 301X^2 - 21)Y + (84X^8 - 372X^6 + 169X^4 + 143X^2 + 15),$$
$$1.$$

Un calcul de pgcd du coefficient de Y dans l'avant dernier reste avec $\mathrm{Res}_Y(A,B)$ montre que ces deux polynômes sont premiers entre eux. Pour chaque racine X de $\mathrm{Res}_Y(A,B)$ il y a donc un unique point d'intersection aux deux courbes, d'ordonnée donnée par

$$Y = -\frac{84X^8 - 372X^6 + 169X^4 + 143X^2 + 15}{4X^8 - 344X^6 - 1243X^4 - 301X^2 - 21}.$$

En utilisant un calcul de pgcd étendu, ceci peut être récrit comme expliqué plus haut en un polynôme de degré au plus 13 en X.

Le même calcul sur A et B vus comme polynômes dans $\mathbb{Q}(Y)[X]$ donne B comme dernier reste avant le pgcd. Ceci correspond aux deux points ayant la même projection sur l'axe des Y.

Implicitation. Une autre application géométrique du résultant est l'*implicitation* de courbes du plan. Étant donnée une courbe paramétrée

$$X = A(T), \quad Y = B(T), \qquad A, B \in \mathbb{K}(T),$$

il s'agit de calculer un polynôme non trivial en X et Y qui s'annule sur la courbe. Il suffit pour cela de prendre le résultant en T du numérateur de $X - A(T)$ et $Y - B(T)$.

EXEMPLE 11. La courbe « en fleur » de la Figure 4 peut aussi être donnée sous la forme

$$X = \frac{4t(1-t^2)^2}{(1+t^2)^3}, \quad Y = \frac{8t^2(1-t^2)}{(1+t^2)^3}.$$

Il suffit d'effectuer le calcul du résultant

$$\mathrm{Res}_t((1+t^2)^3X - 4t(1-t^2)^2, (1+t^2)^3Y - 8t^2(1-t^2))$$

pour retrouver (à un facteur constant près) le polynôme A de l'exemple 8.

2.3. Propriétés. L'essentiel des propriétés du résultant est contenu dans le théorème suivant.

THÉORÈME 1. *Si les polynômes A et B de $\mathbb{A}[X]$ s'écrivent*

$$A = a(X-\alpha_1)\cdots(X-\alpha_m), \quad B = b(X-\beta_1)\cdots(X-\beta_n),$$

alors, le résultant de A et B vaut

$$\mathrm{Res}(A,B) = a^n b^m \prod_{i,j}(\alpha_i - \beta_j) = (-1)^{mn} b^m \prod_{1\le i\le n} A(\beta_i) = a^n \prod_{1\le i\le m} B(\alpha_i) = (-1)^{mn}\,\mathrm{Res}(B,A).$$

DÉMONSTRATION. Il suffit de prouver la première égalité. Les deux suivantes en sont des conséquences immédiates.

Le facteur $a^n b^m$ est une conséquence de la multilinéarité du déterminant. Nous considérons maintenant le cas où $a = b = 1$. Il est commode de considérer le cas générique où les α_i et β_j sont des indéterminées et où l'anneau $\mathbb{A}$ est $\mathbb{Z}[\alpha_1,\ldots,\alpha_m,\beta_1,\ldots,\beta_n]$. Si le résultat est vrai dans cet anneau, il l'est aussi pour des valeurs arbitraires des α_i et β_j. Le corollaire 1 dans $\mathbb{Q}(\alpha_1,\ldots,\alpha_m,\beta_1,\ldots,\beta_n)$ montre que le produit des $\alpha_i - \beta_j$ divise le résultant. Par ailleurs, le degré en α_i de chacune des n premières lignes de la matrice de Sylvester est 1 et ce degré est nul pour les m autres lignes. Ceci donne une borne n pour le degré en chaque α_i du résultant et de la même manière une borne m pour son degré en chaque β_j. Il s'ensuit que le résultant est égal au produit des $\alpha_i - \beta_j$ à un facteur constant près. Ce facteur est indépendant des α_i et β_j. En choisissant $\beta_1 = \cdots = \beta_n = 0$, c'est-à-dire $B = X^n$, et en développant le déterminant par rapport à ses m dernières lignes, on obtient que le résultant vaut $(-1)^{mn}A(0)^n$, ce qui donne le facteur 1 et conclut la preuve. □

COROLLAIRE 2 (Multiplicativité du résultant). *Pour tous polynômes A, B, C de $\mathbb{A}[X]$,* $\mathrm{Res}(AB,C) = \mathrm{Res}(A,C)\,\mathrm{Res}(B,C)$.

DÉMONSTRATION. L'anneau $\mathbb{A}$ est intègre, il possède donc un corps de fraction qui a lui-même une clôture algébrique dans laquelle les polynômes A, B et C peuvent s'écrire sous la forme utilisée dans le théorème précédent. L'identité sur les résultants (qui appartiennent à $\mathbb{A}$ par définition) est alors vérifiée en considérant les produits deux à deux de racines. □

Une dernière propriété utile du résultant est la suivante.

PROPOSITION 4. *Il existe U et V dans $\mathbb{A}[X]$ tels que le résultant s'écrive $S = UA + VB$.*

DÉMONSTRATION. En ajoutant à la dernière colonne de la matrice de Sylvester le produit des colonnes précédentes par des puissances adaptées de X, on fait apparaître dans la dernière colonne les polynômes A et B, sans avoir changé le déterminant. Le développement du déterminant par rapport à la dernière colonne permet alors de conclure. □

L'algorithme d'Euclide étendu montre que S (qui est un multiple par un élément du corps des fractions $\mathbb{K}$ de l'un des restes euclidiens « standards »), peut s'écrire comme une combinaison polynomiale de A et B, à coefficients dans $\mathbb{K}[X]$. Cette proposition montre que cette combinaison se fait « sans division », c'est-à-dire avec des coefficients dans $\mathbb{A}[X]$.

2.4. Calcul avec des nombres algébriques. L'idée que les polynômes sont des bonnes structures de données pour représenter leur racines amène à chercher des algorithmes pour effectuer les opérations de base sur ces racines, comme la somme ou le produit. Le résultant répond à cette attente.

PROPOSITION 5. *Soient $A = \prod_i (X - \alpha_i)$ et $B = \prod_j (X - \beta_j)$ des polynômes de $\mathbb{K}[X]$. Alors*

$$\begin{aligned}
\mathrm{Res}_X(A(X), B(T - X)) &= \prod_{i,j} (T - (\alpha_i + \beta_j)),\\
\mathrm{Res}_X(A(X), B(T + X)) &= \prod_{i,j} (T - (\beta_j - \alpha_i)),\\
\mathrm{Res}_X(A(X), X^{\deg B} B(T/X)) &= \prod_{i,j} (T - \alpha_i \beta_j),\\
\mathrm{Res}_X(A(X), T - B(X)) &= \prod_{i} (T - G(f_i)).
\end{aligned}$$

DÉMONSTRATION. C'est une application directe du Théorème 1. □

EXEMPLE 12. On sait que $\sqrt{2}$ est racine de $X^2 - 2$, tout comme $\sqrt{3}$ est racine de $X^2 - 3$. Un polynôme de degré minimal ayant pour racine $\sqrt{2} + \sqrt{3}$ est donné par

$$\mathrm{Res}_X(X^2 - 2, (T - X)^2 - 3) = T^4 - 10T^2 + 1.$$

Ces opérations ne distinguent pas les racines de A et B, les quatre racines du résultant sont donc $\pm\sqrt{2} \pm \sqrt{3}$.

Calcul du résultant bivarié. On dispose à l'heure actuelle d'un algorithme rapide (quasi-optimal) pour le calcul du résultant univarié. Le meilleur algorithme connu actuellement pour le calcul du résultant bivarié est une méthode d'évaluation-interpolation qui se ramène au résultant univarié. Cet algorithme n'est pas quasi-optimal. En revanche, pour les trois premières opérations de la Proposition 5, des algorithmes quasi-optimaux à base de multiplication rapide de séries existent, ils ont été présentés dans le cours 3.

2.5. ⋆ Sous-résultants ⋆.

La croissance des coefficients dans l'algorithme d'Euclide. Le nombre d'opérations dans le corps des coefficients n'est pas une mesure suffisante de la complexité des calculs lorsque les opérations elles-mêmes ont une complexité variable. C'est le cas pour le calcul de pgcd dans $\mathbb{Q}[X]$ et dans $\mathbb{K}(Y)[X]$. Dans ces corps de coefficients, on constate empiriquement les phénomènes suivants :

- l'algorithme d'Euclide amène à faire des divisions, et introduit des dénominateurs au cours du calcul ;

– la taille des coefficients croît rapidement ;
– ces coefficients peuvent souvent se « simplifier ».

Exemple 13. L'exécution de l'algorithme d'Euclide sur

$$A = 115X^5 + 7X^4 + 117X^3 + 30X^2 + 87X + 44,$$
$$B = 91X^4 + 155X^3 + 3X^2 + 143X + 115.$$

produit les restes successifs suivants :

$$\frac{3601622}{8281}X^3 - \frac{1196501}{8281}X^2 + \frac{151912}{637}X + \frac{2340984}{8281}$$
$$\frac{189886027626841}{12971681030884}X^2 - \frac{57448278681703}{3242920257721}X - \frac{17501090665331}{3242920257721}$$
$$\frac{3748556212578804983806085060}{435414847094570987735 1001}X + \frac{1418333609151239693280148 92}{33493449776505460595007 7}.$$

En simplifiant ces restes par des multiplications par des constantes bien choisies, on obtient les restes :

$$3601622X^3 - 1196501X^2 + 1974856X + 2340984,$$
$$22930325761X^2 - 27749440252X - 8453612204,$$
$$288979986761465X + 142143002707719.$$

Il est souhaitable d'éviter le calcul sur les rationnels ou les fractions rationnelles pour lequel l'addition requiert des calculs de multiplications et éventuellement de pgcd. Une première idée consiste alors à éviter totalement les divisions en utilisant des *pseudo-restes*.

Definition 3. Si A et B sont des polynômes de $\mathbb{A}[X]$ et b est le coefficient de tête de B, le *pseudo-reste* $\overline{R}$ de A et B est défini par

$$b^{\deg A - \deg B + 1} A = \overline{Q}B + \overline{R}.$$

Remplacer les calculs de restes de l'algorithme d'Euclide par des calculs de pseudo-restes évite d'introduire des dénominateurs. Cette idée seule n'est pas suffisante : les coefficients de ces pseudo-restes croissent trop.

Exemple 14. Sur le même exemple, la modification de l'algorithme d'Euclide produit la suite de pseudo-restes

$$3601622X^3 - 1196501X^2 + 1974856X + 2340984,$$
$$189886027626841X^2 - 229793114726812X - 70004362661324,$$
$$257057096083899261191107914552182660X$$
$$+126440823512296156588839626542149756.$$

Une possibilité pour éviter cette croissance est de diviser ces polynômes par le pgcd de leurs coefficients à chaque étape. C'est ce que nous avons fait dans l'exemple ci-dessus. On parle alors de suite de pseudo-restes primitifs. Cependant, ces calculs de pgcd de coefficients sont trop coûteux.

L'algorithme des sous-résultants donné en Figure 6 est une modification de l'algorithme d'Euclide qui évite les divisions, tout en *prévoyant* des facteurs communs qui apparaissent dans les coefficients de la suite des pseudo-restes de sorte à limiter leur croissance. Ces pseudo-restes sont appelés des sous-résultants. Dans cet algorithme, on se contente de renvoyer le dernier sous-résultant non nul ; on conçoit

Sous-résultants(A,B)
Entrée : A et B dans $\mathbb{A}[X]$ avec $\deg B < \deg A$.
Sortie : Le dernier sous-résultant non nul.

1. $f = g = s = 1$
2. Tant que $\deg B > 0$
   (a) $d = \deg A - \deg B$
   (b) $R =$pseudoReste(A, B)
   (c) Si les degrés de A et B sont impairs, faire $s = -s$.
   (d) Faire $A := B$ et mettre dans B le quotient de la division de R par fg^d
   (e) Prendre $f :=$coefficientDominant(A)
   (f) Mettre à jour g en lui affectant le quotient de la division de f^d par g^{d-1}
3. $d = \deg A$;
4. Renvoyer le quotient de la division de sB^d par g^{d-1}.

FIG. 6.

aisément comment modifier l'algorithme pour obtenir *n'importe* quel sous-résultant (spécifié par des conditions de degré, par exemple).

EXEMPLE 15. Toujours sur les même polynômes, la suite des sous-résultants redonne les polynômes que nous avions déjà calculés par simplification :

$$3601622X^3 - 1196501X^2 + 1974856X + 2340984,$$
$$22930325761X^2 - 27749440252X - 8453612204,$$
$$288979986761465X + 142143002707719.$$

La suite des sous-résultant de cet exemple est donc primitive : les facteurs prédits par l'algorithme du sous-résultant ont suffi à éliminer tout facteur commun entre les coefficients des pseudo-restes.

Comme le résultant, les sous-résultants sont liés à la matrice de Sylvester et à l'algorithme d'Euclide. Une formule de Cramer sur une sous-matrice de la matrice de Sylvester donne le résultat suivant.

PROPOSITION 6. *Toutes les divisions effectuées au cours de l'algorithme des sous-résultants sont exactes.*

La preuve de ce résultat est technique et omise ici.

En corollaire, on remarque en outre qu'il est assez facile de déduire de ce résultat des estimations sur la « taille » des coefficients qui apparaissent au cours de l'algorithme. Un cas particulier intéressant est celui où $\mathbb{A}$ est l'anneau de polynômes $\mathbb{K}[Y]$. Alors, si A et B ont des degrés totaux m et n, tous les sous-résultants ont des degrés bornés par mn. Ces bornes permettent un calcul du résultant par évaluation-interpolation dès que l'on dispose d'un algorithme efficace dans le cas univarié.

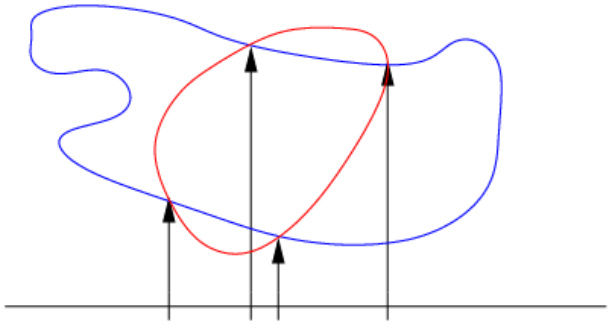


FIG. 7.

Calcul de la paramétrisation. Les sous-résultants permettent de finir la « résolution » des systèmes de deux polynômes bivariés. Pour simplifier, nous faisons l'hypothèse que pour tout x racine du résultant $R = \mathrm{Res}_Y(A, B)$, il n'y a qu'un seul y tel que $A(x, y) = B(x, y) = 0$. Dans ce cas, on peut montrer que le sous-résultant S_1 (de degré 1 en Y) est non nul ; écrivons-le sous la forme $S_1 = P_0(X)Y + Q_0(X)$. Comme pour le résultant, il existe des polynômes U et V dans $\mathbb{K}[X, Y]$ tels qu'on ait l'égalité

$$AU + BV = P_0(X)Y + Q_0(X).$$

On en déduit donc que toutes les solutions (x, y) du système $A = B = 0$ satisfont l'équation $P_0(x)y = Q_0$. Autrement dit, en écartant les solutions dégénérées où $P_0(x) = 0$, on obtient l'ordonnée des points solutions en évaluant la fraction rationnelle $-Q_0/P_0$ sur les racines du résultant R. Autrement dit, cette procédure permet de décrire les solutions du système sous la forme

$$\begin{cases} y = Q(x) \\ R(x) = 0 \end{cases}$$

Géométriquement, ce polynôme Q permet de retrouver y à partir de x, il permet donc d'effectuer l'opération symbolisée sur la Figure 7, la paramétrisation des solutions du système par les racines de R.

EXEMPLE 16. Sur les mêmes polynômes qu'à l'exemple 8, le sous-résultant redonne la paramétrisation calculée à partir de l'algorithme d'Euclide, mais en étant plus économe dans les opérations sur les coefficients.

3. Approximants de Padé-Hermite et algorithmes efficaces

3.1. Définitions.

DEFINITION 4. Un approximant de Padé de type (m, n) d'une série $S \in \mathbb{K}[[X]]$ est une fraction rationnelle $R \in \mathbb{K}(X)$ dont le numérateur a degré borné par m le dénominateur a degré borné par n et telle que

$$R - S = O(X^{m+n+1}).$$

DEFINITION 5. Un approximant de Padé-Hermite de type $m_1, \ldots, m_k$ d'un k-uplet de séries $S_1, \ldots, S_k$ de $\mathbb{K}(X)$ est un k-uplet de polynômes $p_1, \ldots, p_k$ de $\mathbb{K}[X]$ tels que $\deg p_i \leq m_i$, $i = 1, \ldots, k$ et

$$p_1 S_1 + \cdots + p_k S_k = O(X^{m_1+\cdots+m_k+k-1}).$$

Les approximants de Padé sont obtenus comme cas particulier des approximants de Padé-Hermite en prenant le couple de séries $(1, S)$.

3.2. Applications.

Reconstruction rationnelle. Si la série S provient du développement d'une fraction rationnelle P/Q, alors le calcul d'approximant de Padé à partir de suffisamment de termes de la série reconstruit cette fraction. En effet, si A/B est un approximant de type $(\deg P, \deg Q)$, l'identité

$$\frac{A}{B} = S + O(X^{\deg P+\deg Q+1}) = \frac{P}{Q} + O(X^{\deg P+\deg Q+1})$$

entraîne

$$AQ = BP + O(X^{\deg P+\deg Q+1}).$$

Le degré des polynômes intervenant dans cette égalité est borné par $\deg P + \deg Q$. Il s'ensuit que cette identité de séries est une identité de polynômes :

$$AQ = BP.$$

Les applications de cette propriété sont nombreuses : pour la résolution de systèmes différentiels à coefficients constants (cours 8), pour reconnaître une suite récurrente linéaire à partir de ses premiers termes, pour le calcul de polynômes minimaux de matrices creuses par l'algorithme de Wiedemann (cours 26), pour la résolution de systèmes linéaires à coefficients polynomiaux (cours 27), pour le décodage des codes BCH en théorie des codes, etc.

Pgcd. Si A et B sont des polynômes de $\mathbb{K}[X]$ de degrés m et n, il existe un approximant de Padé non nul de A/B type $(n-1, m-1)$ si et seulement si A et B ont un pgcd non trivial. Dans ce cas, l'élément de degré minimal de la base renvoyée par l'algorithme du Théorème 2 fournit deux polynômes U et V tels que

$$UA + VB = 0,$$

où l'égalité à 0 provient à nouveau de considérations de degré.

Il s'ensuit que

$$\frac{A}{B} = -\frac{V}{U}$$

et par conséquent le pgcd vaut $A/V = B/U$.

Pgcd étendu. Une fois trouvé le pgcd, l'identité de Bézout

$$UA + VB - G = 0$$

s'obtient par un approximant de Padé-Hermite de type $(n-1, m-1, 0)$ de (A, B, G), à nouveau dans la même complexité.

À l'inverse, il est possible de calculer des approximants de Padé à partir de l'algorithme d'Euclide étendu.

PROPOSITION 7. *Soit R_i la suite des restes associés à X^n et $\overline{F} = F \mod X^n$. Soit i le premier indice tel que $\deg R_i \leq m-1$, et A_i, B_i les cofacteurs correspondant :*

$$A_i X^n + B_i \overline{F} = R_i.$$

Il existe une solution au problème de Padé $(m, n-m)$ *si et seulement si* $\operatorname{pgcd}(R_i, B_i) = 1$. *Si c'est le cas, tous les approximants* $(m, n-m)$ *sont proportionnels au couple* (R_i, B_i).

3.3. Calcul. Le problème du calcul d'approximants de Padé-Hermite est un problème d'algèbre linéaire : les coefficients de $1, X, X^2, \ldots, X^{m_1+\cdots+m_k+k-2}$ fournissent un système linéaire (homogène) en les coefficients de $p_1, \ldots, p_k$ — c'est d'ailleurs cette réécriture qui sous-tend le choix de l'ordre de troncature : il y a autant d'équations que d'inconnues (à la constante d'homogénéité près : $p_1, \ldots, p_k$ ne peuvent être déterminés qu'à une constante près).

Davantage que de savoir si le problème a une solution, la question est donc plutôt ici de trouver cette solution rapidement : on veut faire mieux que la résolution brutale d'un système linéaire.

Il se trouve que l'on dispose d'un algorithme quasi-optimal, dont l'efficacité est utilisée dans ce cours comme la base de nombreux algorithmes efficaces. La constante dans le $O(\cdot)$ peut être améliorée dans le cas du pgcd et du pgcd étendu, mais il est plus simple pédagogiquement de se reposer sur une seule construction.

THÉORÈME 2. *Soit* $N = m_1 + \cdots + m_k + k - 1$. *Il est possible de calculer une base des approximants de Padé-Hermite sur* $\mathbb{K}[X]$ *en* $O(k^\omega \mathsf{M}(N))$ *opérations dans* $\mathbb{K}$.

L'algorithme repose sur un diviser pour régner assez subtil.

En corollaire, on obtient des algorithmes quasi-optimaux pour le calcul de pgcd et de pgcd étendus de polynômes.

Notes

Pour le résultant de deux polynômes (et les sous-résultants), on peut consulter [**7**, **1**, **2**] ou [**3**] pour une approche un peu plus complète. De nombreuses propriétés des résultants et de l'algorithme d'Euclide sont établies de manière élémentaire à l'aide des fonctions symétriques dans [**4**].

Le pgcd et le pgcd étendu peuvent aussi être définis dans un contexte non-commutatif. Ils sont alors utiles au calcul avec des opérateurs différentiels ou de récurrence. Cette généralisation sera présentée au cours 23.

Bibliographie

[1] Geddes (Keith O.), Czapor (Stephen R.), and Labahn (George). – *Algorithms for Computer Algebra*. – Kluwer Academic Publishers, 1992.

[2] Knuth (Donald E.). – *The Art of Computer Programming*. – Addison-Wesley Publishing Co., Reading, Mass., 1997, 3rd edition, *Computer Science and Information Processing*, vol. 2 : Seminumerical Algorithms, xiv+762p.

[3] Lang (Serge). – *Algebra*. – Springer-Verlag, New York, 2002, third edition, *Graduate Texts in Mathematics*, vol. 211, xvi+914p.

[4] Lascoux (Alain). – *Symmetric functions and combinatorial operators on polynomials*. – Published for the Conference Board of the Mathematical Sciences, Washington, DC, 2003, *CBMS Regional Conference Series in Mathematics*, vol. 99, xii+268p.

[5] Pan (V. Y.) and Wang (X.). – Acceleration of Euclidean algorithm and extensions. In Mora (Teo) (editor), *ISSAC'2002*. pp. 207–213. – ACM, New York, 2002. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, July 07–10, 2002, Université de Lille, France.

[6] Stehlé (D.) and Zimmermann (P.). – A binary recursive Gcd algorithm. In *ANTS-VI. Lecture Notes in Computer Science*, vol. 3076, pp. 411–425. – Springer, 2004.

[7] von zur Gathen (Joachim) and Gerhard (Jürgen). – *Modern computer algebra*. – Cambridge University Press, New York, 2003, 2nd edition, xiv+785p.

[8] Yap (Chee). – *Fundamental Problems in Algorithmic Algebra*. – Oxford University Press, 2000.

COURS 11

Algorithme rapide pour le pgcd

1. Algorithme d'Euclide rapide

Les méthodes à base d'approximants de Padé-Hermite ne peuvent remplacer l'algorithme d'Euclide que dans le cas du calcul de pgcd de polynômes. Historiquement, c'est d'abord un algorithme rapide pour le pgcd d'entiers (le « demi-pgcd ») qui est apparu, il a ensuite été étendu aux polynômes, et les algorithmes rapides pour les approximants de Padé-Hermite sont de conception plus récente. Nous décrivons maintenant l'algorithme de « demi-pgcd ». Pour simplifier, nous nous plaçons dans le cas du pgcd de polynômes, mais cette approche s'étend au cas des entiers.

Soient donc A et B dans $\mathbb{K}[X]$, avec $n = \deg A$ et $\deg B < n$ (ce qui n'est pas une forte restriction). Posons comme précédemment $R_0 = A$, $R_1 = B$. Soient ensuite R_i les restes successifs de l'algorithme d'Euclide et soit en particulier $R_N = \mathrm{pgcd}(A, B)$ le dernier non nul d'entre eux. On sait qu'il existe une matrice 2×2 $M_{A,B}$ de cofacteurs telle que :

$$M_{A,B} \begin{bmatrix} R_0 \\ R_1 \end{bmatrix} = \begin{bmatrix} U_N & V_N \\ U_{N+1} & V_{N+1} \end{bmatrix} \begin{bmatrix} R_0 \\ R_1 \end{bmatrix} = \begin{bmatrix} R_N \\ 0 \end{bmatrix}.$$

L'algorithme de pgcd rapide calcule d'abord cette matrice ; à partir de là, on en déduit aisément le pgcd, pour $O(\mathsf{M}(n))$ opérations supplémentaires.

À titre préparatoire, notons qu'une étape de l'algorithme d'Euclide classique peut elle aussi s'écrire sous une forme matricielle. En effet, si Q_{i+1} est le quotient de la division de R_{i-1} par R_i, on peut écrire :

$$\begin{bmatrix} 0 & 1 \\ 1 & -Q_{i+1} \end{bmatrix} \begin{bmatrix} R_{i-1} \\ R_i \end{bmatrix} = \begin{bmatrix} R_i \\ R_{i+1} \end{bmatrix}.$$

Ainsi, la matrice $M_{A,B}$ est un produit de matrices élémentaires de ce type. Elle est inversible.

Demi-pgcd : définition. Comme étape intermédiaire pour obtenir une division euclidienne rapide, on utilise l'algorithme dit du « demi-pgcd » (*half-gcd* en anglais, d'où la notation *HGCD*), qui permet de faire des « pas de géant » dans la liste des restes successifs.

Le demi-pgcd est défini comme suit. Les degrés des restes successifs R_i décroissent strictement, donc il existe un unique indice j tel que :

- $\deg R_j \geq \frac{n}{2}$;
- $\deg R_{j+1} < \frac{n}{2}$.

Euclide rapide via demi-pgcd

Entrée : $R_0 = A$ et $R_1 B$ dans $\mathbb{K}[X]$ avec $\deg B < \deg A = n$.
Sortie : La matrice $M_{A,B}$.

1. Soit $M_{\text{hgcd}} = \mathsf{HGCD}(A, B)$.
2. Calculer R_j et R_{j+1}.
3. Si $R_{j+1} = 0$, renvoyer M_{hgcd}.
4. Soit M la matrice de la division euclidienne de R_j par R_{j+1}, et R_{j+2} le reste correspondant.
5. Si $R_{j+2} = 0$, renvoyer MM_{hgcd}.
6. Calculer $M_{R_{j+1},R_{j+2}}$ par un appel récursif, et renvoyer $M_{R_{j+1},R_{j+2}} M M_{\text{hgcd}}$.

FIG. 1.

On a vu plus haut (définition de l'algorithme étendu) qu'il existe $U_j, V_j, U_{j+1}, V_{j+1}$ tels que :

$$U_j R_0 + V_j R_1 = R_j \quad \text{et} \quad U_{j+1} R_0 + V_{j+1} R_1 = R_{j+1},$$

ces polynômes étant en outre uniques si on rajoute les conditions de degré voulue sur la relation de Bézout. Ceci peut se réécrire sous la forme matricielle suivante :

$$\begin{bmatrix} U_j & V_j \\ U_{j+1} & V_{j+1} \end{bmatrix} \begin{bmatrix} R_0 \\ R_1 \end{bmatrix} = \begin{bmatrix} R_j \\ R_{j+1} \end{bmatrix}.$$

Pour fixer les idées, en première approximation, on peut estimer que les polynômes $U_j, V_j, U_{j+1}, V_{j+1}$ ont des degrés de l'ordre de $\frac{n}{2}$ (attention, ce n'est qu'une estimation, pas une égalité).

Notons M_{hgcd} la matrice ci-dessus donnant les restes R_j et R_{j+1}. L'algorithme du demi-pgcd (noté HGCD ci-dessous) a pour vocation de calculer cette matrice. Avant d'en étudier le fonctionnement, voyons comment il permet d'obtenir le calcul du pgcd étendu. L'idée est qu'un appel à HGCD en degré n permet d'obtenir les restes de degré approximativement $\frac{n}{2}$; alors, un appel en degré $\frac{n}{2}$ permet d'obtenir les restes de degré approximativement $\frac{n}{4}$, et ainsi de suite jusqu'à trouver le pgcd. L'algorithme est détaillé en Figure 1.

À l'étape 4, R_j et R_{j+1} ont par définition des degrés qui encadrent $\frac{n}{2}$, mais on n'a pas de borne supérieure sur le degré de R_j, qui peut être proche de n. Aussi, pour obtenir deux polynômes de degré au plus $\frac{n}{2}$ pour l'appel récursif, il est nécessaire d'effectuer cette étape de division euclidienne.

Soit $\mathsf{H}(n)$ la complexité de l'algorithme de demi-pgcd sur des entrées de degré au plus n. Pour estimer la complexité de l'algorithme de pgcd rapide, on fait l'hypothèse que $\mathsf{H}(n+n') \geq \mathsf{H}(n) + \mathsf{H}(n')$ et que $\mathsf{M}(n)$ est négligeable devant $\mathsf{H}(n)$. Ces hypothèses sont vérifiées pour l'algorithme proposé plus loin.

PROPOSITION 1. *L'algorithme ci-dessus calcule $M_{A,B}$ en $O(\mathsf{H}(n))$ opérations.*

DÉMONSTRATION. La validité de l'algorithme est immédiate, puisque toutes les matrices calculées ne sont au fond que des matrices de passage, qui font passer de deux restes successifs à deux autres restes successifs. Multiplier ces matrices permet de composer ces opérations de transition.

Estimons la complexité. Le coût de l'appel à HGCD est $\mathsf{H}(n)$. Ensuite, toutes les opérations des étapes 2 à 5 ont une complexité en $O(\mathsf{M}(n))$, en utilisant une division euclidienne rapide pour l'étape 4. On effectue ensuite un appel récursif, puis de nouveau des opérations dont le coût est en $O(\mathsf{M}(n))$. Ainsi, la complexité $\mathsf{B}(n)$ de l'algorithme de pgcd rapide satisfait la récurrence :

$$\mathsf{B}(n) \leq \mathsf{B}(n/2) + \mathsf{H}(n) + C\mathsf{M}(n),$$

C étant une constante. Le lemme « diviser pour régner » permet de conclure. □

Autrement dit, le coût du pgcd est, à une constante près, le même que celui du demi-pgcd. Il devient donc légitime de se consacrer à ce dernier uniquement.

Demi-pgcd : algorithme. Pour mettre en place une stratégie « diviser pour régner » pour le demi-pgcd, on utilise un moyen de « couper les polynômes » en deux. L'idée qui fonctionne est de ne garder que les coefficients de poids forts des polynômes d'entrée et d'utiliser le fait que *le quotient de la division euclidienne de deux polynômes ne dépend que de leurs coefficients de poids fort*, d'une façon tout à fait quantifiée par la suite.

Remarquons qu'on accède aux coefficients de poids fort d'un polynôme en prenant son quotient par un polynôme de la forme X^k : par exemple, si A est

$$X^{10} + 2X^9 + 5X^8 + 3X^7 + 11X^6 + \cdots,$$

le quotient de A par X^6 est

$$X^4 + 2X^3 + 5X^2 + 3X + 11.$$

Voyons comment cette idée permet d'obtenir notre algorithme. Partant de deux polynômes A et B de degrés de l'ordre de n, on leur associe f et g de degré approximativement $\frac{n}{2}$, en nommant f et g les quotients respectifs de A et B par $X^{\frac{n}{2}}$: on a jeté les coefficients de petits degrés.

On calcule la matrice M du demi-pgcd de f et g (moralement, les entrées de cette matrice ont degré $\frac{n}{4}$). La remarque ci-dessus permet de montrer qu'on obtient ainsi une matrice de passage pour les restes de A et B eux-mêmes

On applique donc cette matrice aux polynômes initiaux ; on obtient des restes H et I dont les degrés sont de l'ordre de $\frac{3n}{4}$. On effectue alors une seconde fois le même type d'opération, avec un appel récursif en degré $\frac{n}{2}$, pour gagner à nouveau $\frac{n}{4}$, et finalement arriver en degré $\frac{n}{2}$.

Les choix exacts des degrés de troncature sont subtils, et on admettra que les valeurs données en Figure 2 permettent d'assurer la correction de l'algorithme.

PROPOSITION 2. *L'algorithme ci-dessus calcule la matrice du demi-pgcd de A et B en $O(\mathsf{M}(n)\log(n))$ opérations dans $\mathbb{K}$.*

DÉMONSTRATION. Comme indiqué plus haut, on admet que les choix des degrés de troncature permettent d'assurer la validité de l'algorithme. Il est plus facile d'en effectuer l'analyse de complexité. Le coût $\mathsf{H}(n)$ peut être majoré par deux fois le coût en degré au plus $\frac{n}{2}$ (les deux appels ont lieu aux lignes 3 et 7), plus un certain nombre de multiplications de polynômes et une division euclidienne. En remarquant que tous les produits se font en degré au pire n, on en déduit la récurrence

$$\mathsf{H}(n) \leq 2\mathsf{H}(\frac{n}{2}) + C\mathsf{M}(n),$$

HGCD

Entrée : A et B dans $\mathbb{K}[X]$, avec $\deg B < \deg A = n$.
Sortie : La matrice M_{hgcd} de A et B.

1. Soit $m = \lceil \frac{n}{2} \rceil$. Si $\deg B < m$, renvoyer la matrice identité.
   C'est terminé.
2. Soient f et g les quotients respectifs de A et B par X^m.
   f et g ont des degrés approximativement $n/2$.
3. Soit $M = \mathsf{HGCD}(f, g)$, et A', B' définis par
$$M \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} A' \\ B' \end{bmatrix}.$$
4. Si $\deg B' < m$, renvoyer M.
   C'est terminé.
5. Soient $H = B', I = A' \bmod B'$, et M' la matrice de la division euclidienne correspondante.
   Moralement, B' a un degré de l'ordre au pire de $\frac{3n}{4}$. Par contre, A' peut avoir un degré plus grand. L'étape de division euclidienne permet de se ramener à des polynômes de degrés de l'ordre de $\frac{3n}{4}$.
6. Soit $\ell = 2m - \deg H$, h et i les quotients respectifs de H et I par X^ℓ.
   ℓ est de l'ordre de $\frac{n}{4}$, de sorte de h et i on des degrés de l'ordre $\frac{n}{2}$.
7. Soit $M'' = \mathsf{HGCD}(h, i)$. Renvoyer la matrice $M''M'M$.

FIG. 2.

où C est une constante. La conclusion découle comme d'habitude du lemme « diviser pour régner ». □

Complément : calcul d'un reste choisi. On peut en tirer un raffinement fort utile de l'algorithme de demi-pgcd : le calcul d'un reste particulier (sélectionné par une condition de degré), ainsi que des cofacteurs associés.

PROPOSITION 3. *Soient $R_0 = A$ et $R_1 = B$, avec $\deg A = n$ et $\deg B < \deg A$, et soient R_i les restes successifs de l'algorithme d'Euclide appliqué à A et B.*

Soit $\ell < n$, et R_j le premier reste de degré inférieur à ℓ. On peut calculer R_j et les cofacteurs associés U_j et V_j en $O(\mathsf{M}(n)\log(n))$ opérations de $\mathbb{K}$.

DÉMONSTRATION (SUCCINTE). Si ℓ est plus petit que $\frac{n}{2}$, on fait un appel à HGCD pour se ramener en degré $\frac{n}{2}$, et on effectue un appel récursif. Si ℓ est plus grand que $\frac{n}{2}$, on fait un appel à HGCD sur les polynômes divisés par $X^{n-2\ell}$. □

Algorithmes sur les entiers. Les mêmes idées algorithmiques s'étendent au calcul sur les entiers, mais il est beaucoup plus difficile d'écrire un algorithme correct dans ce cas. On se contentera d'énoncer les résultats de complexité suivants :

THÉORÈME 1. *Soient $R_0 = A \in \mathbb{N}$, $R_1 = B \in \mathbb{N}$, avec $B \leq A$ et R_i les restes de l'algorithme d'Euclide appliqué à A et B. On peut calculer :*
– *le pgcd de A et B,*
– *le premier reste inférieur à un entier $\ell < A$,*
ainsi que les cofacteurs associés en $O(\mathsf{M}_{\mathbb{Z}}(\log A)\log(\log(A)))$ opérations binaires.

L'analogue de la reconstruction rationnelle est également calculable dans la même complexité. Si n est un entier positif, et A un entier compris entre 0 et $n-1$, et étant donné $m \leq n$, il s'agit de calculer des entiers U et V, avec $|U| < m$ et $0 \leq V \leq \frac{n}{m}$ tels que

$$\text{pgcd}(n, V) = 1 \quad \text{et} \quad A = \frac{U}{V} \mod n. \tag{1}$$

Notes

Il est très difficile de trouver une référence complète, lisible et sans erreur sur les algorithmes de pgcd rapide. Le chapitre 11 de [3] fournit une bonne partie des résultats techniques implicitement utilisés dans la section 1, mais l'algorithme qu'il propose est erroné. Les notes du cours [4] nous ont beaucoup inspiré, mais elles contiennent des erreurs pour le pgcd entier. On pourra également consulter des articles plus récents [1, 2], mais rien ne garantit qu'ils soient libres d'erreurs.

Ces algorithmes de pgcd rapide sont aussi assez délicats à implanter de manière efficace. Par exemple, pour le pgcd de polynômes, une bonne implantation doit prendre en compte les algorithmes de multiplication utilisés. Si on utilise la FFT, il est possible d'économiser des transformées de Fourier directes et inverses ; il faut alors régler le nombre de points d'évaluation en fonction des degrés du résultat final, et pas des résultats intermédiaires, etc. Si l'on travaille sur un corps fini « raisonnable » (les coefficients faisant de quelques bits jusqu'à quelques dizaines voire centaines de bits), on peut estimer que le seuil au-delà duquel utiliser l'algorithme rapide se situe autour des degrés 200 ou 300. Il faut noter que très peu de systèmes disposent de telles implantations (c'est le cas pour Magma et la bibliothèque NTL). En ce qui concerne le pgcd des entiers, la situation est sensiblement plus délicate, toujours à cause du problème des retenues (une bonne partie des algorithmes présentés dans les ouvrages de référence sont incorrects, à cause de ce problème). Pour donner une idée, dans les systèmes Magma, Mathematica, ou des bibliothèques telles que GMP (code toujours en développement et utilisé entre autres par les entiers de Maple), le seuil se situe autour de nombres de 30 000 bits (10 000 chiffres décimaux).

Bibliographie

[1] Pan (V. Y.) and Wang (X.). – Acceleration of Euclidean algorithm and extensions. In Mora (Teo) (editor), *ISSAC'2002*. pp. 207–213. – ACM, New York, 2002. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, July 07–10, 2002, Université de Lille, France.

[2] Stehlé (D.) and Zimmermann (P.). – A binary recursive Gcd algorithm. In *ANTS-VI. Lecture Notes in Computer Science*, vol. 3076, pp. 411–425. – Springer, 2004.

[3] von zur Gathen (Joachim) and Gerhard (Jürgen). – *Modern computer algebra*. – Cambridge University Press, New York, 2003, 2nd edition, xiv+785p.

[4] Yap (Chee). – *Fundamental Problems in Algorithmic Algebra*. – Oxford University Press, 2000.

COURS 12

Algorithmique des séries D-finies

Résumé

Les séries D-finies se calculent rapidement. L'équation différentielle linéaire les définissant fournit une structure de données adaptée sur laquelle plusieurs opérations utiles sont algorithmiques.

Les séries D-finies (c'est-à-dire solutions d'équations différentielles linéaires à coefficients polynomiaux) ont des coefficients qui satisfont une récurrence linéaire, ce qui permet d'en calculer les N premiers en $O(N)$ opérations, donc plus vite que la plupart des autres séries. Il est de ce fait crucial de reconnaître les séries qui sont D-finies et de disposer des équations différentielles les définissant. De plus, les coefficients des séries D-finies forment des suites qui sont appelées P-récursives, dont l'algorithmique est évidemment étroitement liée à celle des séries D-finies.

L'importance de ces séries et suites provient d'une part de leur algorithmique spécifique, et d'autre part de leur omniprésence dans les applications. Ainsi, le *Handbook of Mathematical Functions*, référence importante en physique, chimie et mathématiques appliquées, comporte environ 60% de fonctions solutions d'équations différentielles linéaires ; de même, les suites P-récursives forment environ un quart des plus de 100 000 suites référencées dans la version en ligne de l'*Encyclopedia of Integer Sequences* de N. Sloane.

1. Équations différentielles et récurrences

Definition 1. Une série formelle $A(X)$ à coefficients dans un corps $\mathbb{K}$ est dite *différentiellement finie* (ou D-finie) lorsque ses dérivées successives A, A', …, engendrent un espace vectoriel de dimension finie sur le corps $\mathbb{K}(X)$ des fractions rationnelles.

De manière équivalente, cette série est solution d'une équation différentielle linéaire à coefficients dans $\mathbb{K}(X)$: si c'est le cas alors l'équation différentielle permet de récrire toute dérivée d'ordre supérieur à celui de l'équation en termes des dérivées d'ordre moindre (en nombre borné par l'ordre), à l'inverse, si l'espace est de dimension finie, alors pour d suffisamment grand, $A, A', \ldots, A^{(d)}$ sont liées et une relation de liaison entre ces dérivées est une équation différentielle linéaire.

Definition 2. Une suite $(a_n)_{n\geq 0}$ d'éléments d'un corps $\mathbb{K}$ est appelée suite *polynomialement récursive* (ou P-récursive) si elle satisfait une récurrence de la forme

$$p_d(n)a_{n+d} + p_{d-1}(n)a_{n+d-1} + \cdots + p_0(n)a_n = 0, \qquad n \geq 0, \tag{1}$$

où les p_i sont des polynômes de $\mathbb{K}[X]$.

Dans la suite, $\mathbb{K}$ aura toujours caractéristique nulle. On peut donc penser sans rien perdre aux idées à $\mathbb{K} = \mathbb{Q}$.

1.1. Méthode naïve. Le résultat qu'il s'agit de considérer d'un point de vue algorithmique est le suivant.

THÉORÈME 1. *Une série formelle est D-finie si et seulement si la suite de ses coefficients est P-récursive.*

DÉMONSTRATION. Soit $A(X) = a_0 + a_1 X + \cdots$ une série D-finie et

$$q_0(X)A^{(d)}(X) + \cdots + q_d(X)A(X) = 0 \tag{2}$$

une équation différentielle qui l'annule. En notant $[X^n]f(X)$ le coefficient de X^n dans la série $f(X)$, on a les relations

$$\begin{aligned} [X^n]f'(X) &= (n+1)[X^{n+1}]f(X) \quad (n \geq 0), \\ [X^n]X^k f(X) &= [X^{n-k}]f(X) \quad (n \geq k). \end{aligned} \tag{3}$$

Par conséquent, l'extraction du coefficient de X^n de (2) fournit une récurrence linéaire sur les a_n valide dès lors que $n \geq n_0 := \max_{0\leq i\leq d} \deg q_i$. Pour obtenir une récurrence valide pour tout $n \geq 0$, il suffit de multiplier cette récurrence par le polynôme $n(n-1)\cdots(n-n_0+1)$.

À l'inverse, soit (a_n) une suite vérifiant la récurrence (1). Les identités analogues à (3) sont maintenant

$$\sum_{n\geq 0} n^k a_n X^n = \left(X\frac{d}{dX}\right)^k A(X), \quad \sum_{n\geq 0} a_{n+k}X^n = (A(X) - a_0 - \cdots - a_{k-1}X^{k-1})/X^k,$$

où A est la série génératrice des coefficients a_n et la notation $(Xd/dX)^k$ signifie que l'opérateur Xd/dX est appliqué k fois. En multipliant (1) par X^n et en sommant pour n allant de 0 à ∞, puis en multipliant par une puissance de X on obtient donc une équation différentielle linéaire de la forme

$$q_0(X)A^{(d)}(X) + \cdots + q_d(X)A(X) = p(X),$$

où le membre droit provient des conditions initiales. Il est alors possible, quitte à augmenter l'ordre de l'équation de 1, de faire disparaître ce membre droit, par une dérivation et une combinaison linéaire. □

EXERCICE 1. Estimer la complexité d'un algorithme direct utilisant cette idée en nombre d'opérations dans $\mathbb{K}$.

Un algorithme plus efficace pour passer d'une équation différentielle d'ordre élevé à la récurrence linéaire satisfaite par les coefficients des solutions est donnée en §1.3.

1.2. Exemples d'applications.

EXEMPLE 1. Pour calculer une racine du polynôme $P_N(x)$ défini par la série génératrice

$$\sum_{n\geq 0} P_n(x)\frac{z^n}{n!} = \left(\frac{1+z}{1+z^2}\right)^x,$$

lorsque N est grand, il n'est pas nécessaire de calculer ce polynôme. Il suffit d'observer que cette série vérifie une équation différentielle linéaire d'ordre 1 (avec x en paramètre), ainsi que la série génératrice des dérivées des P_n, et d'utiliser les récurrences que l'on en déduit sur ces polynômes pour en calculer des valeurs. Ces valeurs permettent alors d'appliquer une méthode de Newton par exemple pour

résoudre le polynôme. Cette idée peut aussi être combinée avec la méthode de scindage binaire.

EXEMPLE 2. Le cas particulier des récurrences d'ordre 1 donne lieu aux suites *hypergéométriques*, qui jouent un rôle important dans la sommation symbolique abordée dans un cours ultérieur.

1.3. ⋆ Algorithme rapide ⋆. Vue l'efficacité obtenue grâce au passage de l'équation différentielle à la récurrence, il est souhaitable de maîtriser le coût de cette conversion. Il est possible d'obtenir la récurrence plus efficacement que par la méthode naïve, en mettant en œuvre des techniques abordées dans des cours précédents.

1.3.1. *Cas d'un opérateur donné en $\theta = Xd/dX$.* Si l'équation différentielle linéaire de départ est donnée non pas comme un polynôme en X et d/dX, mais comme un polynôme en X et $\theta = Xd/dX$ (θ est parfois appelé l'opérateur d'Euler), alors la conversion en récurrence est assez facile : partant de

$$\sum_{\substack{0\le j\le m\\ 0\le i\le d}} a_{ij} X^j \theta^i,$$

les relations (3) donnent l'opérateur de récurrence

$$\sum a_{ij} S_n^{-j} n^i = \sum a_{ij} (n-j)^i S_n^{-j}.$$

De cette conversion découle le résultat de complexité suivant.

PROPOSITION 1. *La récurrence satisfaite par les coefficients des séries solutions d'une équation différentielle linéaire de degré d en θ et m en X se calcule en $O(mM(d))$ opérations sur les coefficients.*

Par rapport au nombre dm de coefficients du résultat, cette complexité est quasi-optimale.

La preuve utilise la formule ci-dessus et l'observation que calculer les coefficients du polynôme $P(X-j)$ connaissant ceux du polynôme $P(X)$ de degré d ne requiert que $O(M(d))$ opérations, par exemple en utilisant la somme composée vue au cours 6.

1.3.2. *Cas général.* Quitte à le multiplier au préalable par une puissance de X égale au plus à son degré en d/dX, il est toujours possible de récrire un polynôme en X et ∂ en un polynôme en X et θ. Cette récriture peut elle-même être effectuée assez rapidement.

Une première observation est que de la commutation

$$(\theta - i)X^i = X^i\theta$$

se déduit en multipliant à droite par ∂^i la relation

$$X^{i+1}\partial^{i+1} = (\theta - i)X^i\partial^i = (\theta - i)(\theta - i + 1)\cdots\theta.$$

Étant donnés des polynômes $a_i(X)$ de degré au plus $m+d$, il s'agit donc maintenant de calculer des polynômes $b_i(X)$ tels que

$$\sum_{i=0}^{d} a_i(X)X^i\partial^i = \sum_{i=0}^{d} a_i(X)(\theta - i + 1)\cdots\theta = \sum_{i=0}^{d} b_i(X)\theta^i.$$

Récrire le polynôme sous la forme

$$\sum_{j=0}^{m+d} X^j \sum_{i=0}^{d} a_{ij} X^i \partial^i$$

s'effectue en nombre linéaire d'opérations et montre qu'il suffit de savoir traiter efficacement le cas où les a_i (et donc aussi les b_i) sont constants. La transition des uns vers les autres se calcule alors par évaluation-interpolation sur $\theta = 0, 1, 2, \ldots$. Soit P le polynôme à calculer. Les premières identités obtenues par évaluation sont

$$a_0 = b_0, \quad a_0 + a_1 = \sum b_i, \quad a_0 + 2a_1 + 2a_2 = \sum 2^i b_i,$$

et plus généralement

$$e^X \sum a_i X^i = \sum \frac{P(i)}{i!} X^i,$$

ce qui montre que les valeurs de P en $0, \ldots, d$ peuvent être obtenues en $O(M(d))$ opérations à partir des coefficients a_i, et par suite les coefficients de P en $O(M(d) \log d)$ opérations par interpolation en utilisant les techniques du cours sur l'évaluation-interpolation.

THÉORÈME 2. *Le calcul des N premiers termes d'une série solution d'une équation différentielle linéaire d'ordre d à coefficients des polynômes de degré au plus m requiert un nombre d'opérations arithmétiques borné par*

$$O\left((m+d)M(d)\left(\log d + \frac{N}{d}\right)\right).$$

La première partie de l'estimation provient des estimations ci-dessus, la seconde de la complexité du calcul des N premiers termes d'une suite solution d'une récurrence linéaire d'ordre au plus $m + d$ avec des coefficients de degré au plus d, vue au cours 2.

2. Somme et produit

THÉORÈME 3. *L'ensemble des séries D-finies à coefficients dans un corps $\mathbb{K}$ est une algèbre sur $\mathbb{K}$. L'ensemble des suites P-récursives d'éléments de $\mathbb{K}$ est aussi une algèbre sur $\mathbb{K}$.*

DÉMONSTRATION. Les preuves pour les suites et les séries sont similaires. Les preuves pour les sommes sont plus faciles que pour les produits, mais dans le même esprit. Nous ne donnons donc que la preuve pour le produit $h = fg$ de deux séries D-finies f et g. Par la formule de Leibniz, toutes les dérivées de h s'écrivent comme combinaisons linéaires de produits entre une dérivée $f^{(i)}$ de f et une dérivée $g^{(j)}$ de g. Les dérivées de f et de g étant engendrées par un nombre fini d'entre elles, il en va de même pour les produits $f^{(i)}g^{(j)}$, ce qui prouve la D-finitude de h. □

EXERCICE 2. Faire la preuve pour le cas du produit de suites P-récursives.

En outre, cette preuve permet de borner l'ordre des équations : l'ordre de l'équation satisfaite par une somme est borné par la somme des ordres des équations satisfaites par les sommants, et l'ordre de l'équation satisfaite par un produit est borné par le produit des ordres.

Cette preuve donne également un algorithme pour trouver l'équation différentielle (resp. la récurrence) cherchée : il suffit de calculer les dérivées (resp.

les décalées) successives en les récrivant sur un ensemble fini de générateurs. Une fois leur nombre suffisant (c'est-à-dire au pire égal à la dimension plus 1), il existe une relation linéaire entre elles. À partir de la matrice dont les lignes contiennent les coordonnées des dérivées successives (resp. des décalés successifs) sur cet ensemble fini de générateurs, la détermination de cette relation se réduit alors à celle du noyau de la transposée.

Exemple 3. L'identité de Cassini sur les nombres de Fibonacci s'écrit

$$F_{n+2}F_n - F_{n+1}^2 = (-1)^n.$$

Pour calculer le membre droit de cette égalité, le point de départ est simplement la récurrence définissant les nombres de Fibonacci :

$$F_{n+2} = F_{n+1} + F_n,$$

qui exprime que tous les décalés de F_n sont des combinaisons linéaires de F_n et F_{n+1}. Les produits qui interviennent dans l'identité de Cassini s'expriment donc *a priori* comme combinaison linéaire de F_n^2, F_nF_{n+1} et F_{n+1}^2 et donc le membre de gauche vérifie une récurrence d'ordre borné par 4. Le calcul est assez simple et donne une récurrence d'ordre 2 :

$$\begin{aligned} u_n &= F_{n+2}F_n - F_{n+1}^2 = F_{n+1}F_n + F_n^2 - F_{n+1}^2, \\ u_{n+1} &= F_{n+2}F_{n+1} + F_{n+1}^2 - F_{n+2}^2 = F_{n+1}^2 - F_n^2 - F_nF_{n+1} \\ &= -u_n. \end{aligned}$$

La preuve de l'identité est alors conclue en observant que $u_0 = 1$.

En réalité, ce calcul donne plus que la preuve de l'identité : il détermine le membre droit à partir du membre gauche. Si le membre droit est donné, le calcul est bien plus simple : comme le membre gauche vérifie une récurrence d'ordre au plus 4 et le membre droit une récurrence d'ordre 1, leur différence vérifie une récurrence d'ordre au plus 5. Il n'est pas nécessaire de calculer cette récurrence. Il suffit de vérifier que ses 5 conditions initiales sont nulles. Autrement dit, vérifier l'identité pour $n = 0, \dots, 4$ la prouve !

Exercice 3. De la même manière, montrer que $\sin^2 x + \cos^2 x = 1$, avec et sans calcul.

3. Produit d'Hadamard

Corollaire 1. *Si $f = \sum_{n\geq 0} a_n X^n$ et $g = \sum_{n\geq 0} b_n X^n$ sont deux séries D-finies, alors leur produit d'Hadamard*

$$f \odot g = \sum_{n\geq 0} a_n b_n X^n$$

l'est aussi.

La preuve est également un algorithme : des deux équations différentielles se déduisent deux récurrences satisfaites par les suites (a_n) et (b_n) ; d'après la section précédente, le produit (a_nb_n) vérifie alors une récurrence linéaire, dont se déduit enfin l'équation différentielle satisfaite par sa série génératrice.

Exemple 4. Les polynômes de Hermite ont pour série génératrice

$$\sum_{n\geq 0} H_n(x)\frac{z^n}{n!} = \exp(z(2x-z)).$$

À partir de là, la détermination du membre droit de l'identité suivante due à Mehler est entièrement algorithmique :

$$\sum_{n\geq 0} H_n(x)H_n(y)\frac{z^n}{n!} = \frac{\exp\left(\frac{4z(xy-z(x^2+y^2))}{1-4z^2}\right)}{\sqrt{1-4z^2}}.$$

4. Séries algébriques

Théorème 4. *Si la série $Y(X)$ annule un polynôme $P(X,Y)$ de degré d en Y, alors elle est solution d'une équation différentielle linéaire d'ordre au plus d.*

Démonstration. La preuve est algorithmique. Quitte à diviser d'abord P par son pgcd avec sa dérivée P_Y par rapport à Y, il est possible de le supposer premier avec P_Y (car la caractéristique est nulle !). En dérivant $P(X,Y)=0$ et en isolant Y', il vient

$$Y' = -\frac{P_X}{P_Y}.$$

Par *inversion modulaire* de P_Y (voir Cours 10), cette identité se récrit via un calcul de pgcd étendu en

$$Y' = R_1(Y) \bmod P,$$

où R_1 est un polynôme en Y de degré au plus d et à coefficients dans $\mathbb{K}(X)$. Ceci signifie que Y' s'écrit comme combinaison linéaire de $1, Y, Y^2, \ldots, Y^{d-1}$ à coefficients dans $\mathbb{K}(X)$. Dériver à nouveau cette équation, puis récrire Y' et prendre le reste de la division par P mène à nouveau à une telle combinaison linéaire pour Y'' et plus généralement pour les dérivées successives de Y. Les $d+1$ vecteurs $Y, Y', \ldots, Y^{(d)}$ sont donc linéairement dépendants et la relation de liaison est l'équation cherchée. □

Exemple 5. Les dénombrements d'arbres mènent naturellement à des équations algébriques sur les séries génératrices. Ainsi, la série génératrice des nombres de Catalan (nombre d'arbres binaires à n sommets internes) vérifie

$$y = 1 + zy^2;$$

la série génératrice des nombres de Motzkin (nombre d'arbres unaires-binaires à n sommets internes) vérifie

$$y = 1 + zy + zy^2.$$

Dans les deux cas, il est aisé d'obtenir d'abord une équation différentielle puis une récurrence qui permet de calculer efficacement ces nombres par scindage binaire. Dans le cas des nombres de Catalan, la récurrence est d'ordre 1, la suite est donc hypergéométrique et s'exprime aisément.

Exercice 4. Trouver une formule explicite des nombre de Catalan. Récrire les fonctions Γ qui pourraient apparaître dans le résultat en termes de factorielles, puis de coefficient binomial.

Exercice 5. À l'aide d'un système de calcul formel, calculer une récurrence linéaire satisfaite par les coefficients de la série y solution de

$$y = 1 + zy + zy^7.$$

Les mêmes arguments que ci-dessus mènent à une autre propriété de clôture des séries D-finies.

Corollaire 2. *Si f est une série D-finie et y une série algébrique sans terme constant, alors $f \circ y$ est D-finie.*

La preuve consiste à observer que les dérivées successives de $f \circ y$ s'expriment comme combinaisons linéaires des $f^{(i)}(y)y^j$ pour un nombre fini de dérivées de f (par D-finitude) et de puissances de y (par la même preuve que pour le théorème 4). Cette preuve fournit encore un algorithme.

Exemple 6. À l'aide d'un système de calcul formel, calculer une récurrence linéaire satisfaite par les coefficients du développement en série de Taylor de

$$\exp\left(\frac{1-\sqrt{1-4z}}{2}\right).$$

5. ⋆ Limitations ⋆

En général, la composition de deux séries D-finies n'est pas D-finie. Voici trois résultats plus forts, dont la preuve repose sur la théorie de Galois différentielle et dépasse le cadre de ce cours.

Théorème 5.
1. *Les séries f et $1/f$ sont simultanément D-finies si et seulement si f'/f est algébrique.*
2. *Les séries f et $\exp(\int f)$ sont simultanément D-finies si et seulement si f est algébrique.*
3. *Soit g algébrique de genre supérieur ou égal à 1, alors f et $g \circ f$ sont D-finies si et seulement si f est algébrique.*

Exercice 6. Prouver le sens "si" de ces trois propriétés.

Exercices

Exercice 7 (Opérations de clôture pour les équations différentielles linéaires à coefficients constants). Soient $f(X)$ et $g(X)$ solutions des équations différentielles linéaires homogènes

$$a_m f^{(m)}(X) + \cdots + a_0 f(X) = 0, \qquad b_n g^{(n)}(X) + \cdots + b_0 g(X) = 0,$$

avec $a_0, \ldots, a_m, b_0, \ldots, b_n$ des coefficients rationnels.

1. Montrer que $f(X)g(X)$ et $f(X) + g(X)$ sont solutions d'équations différentielles du même type.

Si le polynôme $a_m X^m + \cdots + a_0$ se factorise sur $\mathbb{C}$ en

$$a_m(X-\alpha_1)^{d_1} \cdots (X-\alpha_k)^{d_k},$$

on rappelle qu'une base de l'espace des solutions de

$$a_m f^{(m)}(X) + \cdots + a_0 f(X) = 0$$

est donnée par $\left\{e^{\alpha_1 X}, Xe^{\alpha_1 X}, \ldots, X^{d_1-1}e^{\alpha_1 X}, \ldots, e^{\alpha_k X}, Xe^{\alpha_k X}, \ldots, X^{d_k-1}e^{\alpha_k X}\right\}$.

2. Montrer qu'une équation satisfaite par $f(X) + g(X)$ peut être calculée à l'aide de l'algorithme d'Euclide.
3. Montrer qu'une équation satisfaite par $f(X)g(X)$ peut être obtenue par un calcul de résultant.

Exercice 8 (Calcul efficace de coefficients trinomiaux centraux). Soit $\mathsf{C} : \mathbb{N} \to \mathbb{N}$ la fonction définie par $\mathsf{C}(n) = \lfloor n \log(n+1) \log(\log(n+3)) \rfloor$ pour tout $n \geq 1$. On rappelle qu'il est possible de multiplier des entiers de n chiffres binaires en $O(\mathsf{C}(n))$ opérations binaires et des polynômes de degré n à coefficients dans un anneau arbitraire en $O(\mathsf{C}(n))$ opérations arithmétiques dans l'anneau.

Soit $N \in \mathbb{N}$ et soit $P = \sum_{i=0}^{2N} p_i X^i \in \mathbb{Z}[X]$ le polynôme $P(X) = (1+X+X^2)^N$.

1. Montrer que $O(\mathsf{C}(N))$ opérations binaires suffisent pour déterminer la parité de tous les coefficients de P.
   Indication : un entier n est pair si et seulement si $n = 0$ dans $\mathbb{Z}/2\mathbb{Z}$.
2. Montrer que P vérifie une équation différentielle linéaire d'ordre 1 à coefficients polynomiaux. En déduire que les p_i suivent une récurrence d'ordre 2 que l'on précisera.
3. Donner un algorithme qui calcule p_N en $O(\mathsf{C}(N \log N) \log N)$ opérations binaires.

Notes

Les propriétés de clôture des séries D-finies ont été décrites avec leurs applications par Stanley dans [12] ainsi que dans son livre [13], et par Lipshitz dans [8].

L'utilisation des séries D-finies pour les séries algébriques en combinatoire est exploitée à de nombreuses reprises par Comtet dans son livre [4], où il utilise l'algorithme décrit dans la preuve du Théorème 4. L'histoire de cet algorithme est compliquée. Il était connu d'Abel qui l'avait rédigé dans un manuscrit de 1827 qui n'a pas été publié. Ce manuscrit est décrit (p. 287) dans les œuvres complètes d'Abel [1]. Ensuite, ce résultat a été retrouvé par Sir James Cockle en 1860 et popularisé par le révérend Harley en 1862 [6]. Quelques années plus tard, il est encore retrouvé par Tannery [14] dans sa thèse, dont le manuscrit est remarquablement clair et disponible sur le web (à l'url `http://gallica.bnf.fr`).

Les limitations de la dernière section ne sont pas très connues. Elles sont dues à Harris et Sibuya pour la première [7] et à Singer pour les deux autres [10].

En ce qui concerne les algorithmes et les implantations, la plupart des algorithmes présentés dans ce cours sont implantés dans le package `gfun` de Maple [9]. L'algorithme rapide de la section 1.3 provient essentiellement de [3], qui donne une variante légèrement plus efficace (d'un facteur constant). L'exercice 8 est tiré d'une réponse à un "défi" [5].

Bibliographie

[1] Abel (Niels Henrik). – *Œuvres complètes. Tome II.* – Éditions Jacques Gabay, Sceaux, 1992, vi+716p. Edited and with notes by L. Sylow and S. Lie, Reprint of the second (1881) edition. Disponible en ligne à `http://gallica.bnf.fr`.

[2] Abramowitz (Milton) and Stegun (Irene A.) (editors). – *Handbook of mathematical functions with formulas, graphs, and mathematical tables.* – Dover Publications Inc., New York, 1992, xiv+1046p. Reprint of the 1972 edition.

[3] Bostan (Alin). – *Algorithmique efficace pour des opérations de base en Calcul formel*. – PhD thesis, École polytechnique, December 2003.

[4] Comtet (L.). – *Analyse Combinatoire*. – PUF, Paris, 1970. 2 volumes.

[5] Flajolet (Philippe) and Salvy (Bruno). – The Sigsam challenges : Symbolic asymptotics in practice. *SIGSAM Bulletin*, vol. 31, n° 4, December 1997, pp. 36–47.

[6] Harley (Rev. Robert). – On the theory of the transcendental solution of algebraic equations. *Quarterly Journal of Pure and Applied Mathematics*, vol. 5, 1862, pp. 337–360.

[7] Harris (William A.) and Sibuya (Yasutaka). – The reciprocals of solutions of linear ordinary differential equations. *Advances in Mathematics*, vol. 58, n° 2, 1985, pp. 119–132.

[8] Lipshitz (L.). – *D*-finite power series. *Journal of Algebra*, vol. 122, n° 2, 1989, pp. 353–373.

[9] Salvy (Bruno) and Zimmermann (Paul). – Gfun : a Maple package for the manipulation of generating and holonomic functions in one variable. *ACM Transactions on Mathematical Software*, vol. 20, n° 2, 1994, pp. 163–177.

[10] Singer (Michael F.). – Algebraic relations among solutions of linear differential equations. *Transactions of the American Mathematical Society*, vol. 295, n° 2, 1986, pp. 753–763.

[11] Sloane (N. J. A.). – *The On-Line Encyclopedia of Integer Sequences*. – 2006. Published electronically at `http ://www.research.att.com/~njas/sequences/`.

[12] Stanley (R. P.). – Differentiably finite power series. *European Journal of Combinatorics*, vol. 1, n° 2, 1980, pp. 175–188.

[13] Stanley (Richard P.). – *Enumerative combinatorics*. – Cambridge University Press, 1999, vol. 2, xii+581p.

[14] Tannery (Jules). – *Propriétés des intégrales des équations différentielles linéaires à coefficients variables*. – Thèse de doctorat ès sciences mathématiques, Faculté des Sciences de Paris, 1874. Disponible en ligne à `http://gallica.bnf.fr`.

Deuxième partie

Systèmes Polynomiaux

COURS 13

Bases standard

Résumé

Dans ce chapitre, nous allons généraliser à plusieurs variables à la fois l'algorithme d'Euclide et l'algorithme de Gauss.

Lien PGCD/Euclide/Gauss/Résultant. Géométrie et idéaux. Bases standard. Divisions à plusieurs variables. Historique. Bornes de complexité.

Exemple introductif : essayons de faire prouver à une machine le simplissime théorème d'Appollonius, qui affirme que les milieux des côtés d'un triangle rectangle, le sommet de l'angle droit et le pied de la hauteur relative à l'hypothénuse sont sur un même cercle. Ce n'est qu'une version simplifiée du cercle des 9 points d'un triangle.

Soit donc OAB un triangle rectangle de sommet O. Nous pouvons toujours choisir un système de coordonnées tel que l'origine soit en O, et les deux autres sommets les points respectivement $(2a, 0)$ et $(0, 2b)$. Le cercle passant par le sommet et les milieux $(a, 0)$ et $(0, b)$ des côtés a pour équation $X^2 + Y^2 - aX - bY = 0$. Vérifier que le milieu (a, b) de l'hypothénuse est sur ce cercle n'est qu'une évalution du polynôme équation. Le pied de la hauteur $M = (x, y)$ est définie par les relations linéaires de son appartenance à AB et l'orthogonalité entre OM et AB, soit le système linéaire $ay + bx - 2ab = by - ax = 0$. Résolution immédiate :

$$y = \frac{a}{b}x \; x = \frac{2ab^2}{a^2 + b^2} \; y = \frac{2a^2b}{a^2 + b^2}$$

Polynômes hypothèses (ici linéaires) $\Rightarrow$ polynôme conclusion, modulo une petite non-dégénérescence.

Exemple à 1 variable / PGCD / Euclide. Exemple de générateurs, Gauss.

Interprétation en termes de monômes privilégiés, réécriture, idéal, appartenance à l'idéal, anneau quotient.

Calcul dans $k(a, b)[x, y]$. On essaie d'écrire le polynôme conclusion (l'équation du cercle) comme « conséquence du système d'équations hypothèses », c'est-à-dire combinaison linéaire à coefficients polynomiaux des polynômes hypothèses.

Notion d'idéal engendré.

Monde du calcul (réécriture) vs. monde de la géométrie.

1. Introduction

Rappelons la notion d'anneau (resp. de corps) effectif. Tout d'abord que tout élément possède une représentation en machine via une **structure de données**. Puis les opérations de base : addition, opposé, multiplication, (resp. plus l'inversion d'un élément non nul) sont réalisables par un algorithme. Enfin, il ne faut pas

oublier le test d'égalité (qui se réduit au test à zéro) qui doit être algorithmique (cf. *** CEX Richardson cours 1 ***).

EXERCICE 1. L'anneau des entiers $\mathbb{Z}$ et le corps $\mathbb{Q}$ des rationnels sont effectifs. Imaginer une représentation et décrire les algorithmes.

EXERCICE 2. Un anneau de polynômes sur un corps effectif est effectif. Une extension algébrique ou transcendante de $\mathbb{Q}$ est un corps effectif.

EXERCICE 3. Un anneau quotient d'un anneau de polynômes sur un corps effectif par un idéal est effectif est-il effectif ?

Relation d'équivalence associée.

L'introduction nous suggère l'importance d'un ordre sur les monômes permettant de privilégier l'un d'entre eux.

Cas linéaire, cas à une variable.

2. Ordres totaux admissibles sur le monoïde des monômes

À un élément $a = (a_1, \dots, a_n)$ de $\mathbb{N}$ est associé le monôme ou produit de puissances $X = ({X_1}^{a_1} \cdots {X_n}^{a_n})$. Ainsi les monômes de l'anneau R forment un monoïde multiplicatif $\mathcal{M}_n$ isomorphe au monoïde additif $\mathbb{N}^n$ (l'élément neutre 1 correspond au point de coordonnées toutes nulles). Nous nous autoriserons à utiliser indifféremment la notation additive ou multiplicative.

DÉFINITION 1. *Un ordre total $<$ sur les monômes est dit* compatible*s'il est compatible avec la structure de monoïde. Tous les éléments distincts de l'élément neutre sont d'un même côté que celui-ci, et la multiplication par un monôme est croissante.*

Il est dit de plus admissible *si l'élément neutre est plus petit que tous les autres :*

(i) $1 < m$ *si* m *n'est pas l'élément neutre ;*

(ii) $m' < m''$ *implique pour tout* m $mm' < mm''$.

EXEMPLE 1. L'ordre lexicographique : pour ordonner deux points différents de $\mathbb{N}^n$, on regarde la première coordonnée où elles diffèrent. En fait, on devrait plutôt parler de l'ordre lexicographique *induit* par un ordre total sur les variables. C'est ainsi que l'ordre des lettres de l'alphabet induit un ordre sur les mots du dictionnaire.

EXEMPLE 2. Il existe aussi pour les rimailleurs des dictionnaires de rimes : les mots sont ordonnés d'après les dernières lettres ; et pour les amateurs de *Scrabble*, des listes de mots rangés par longueur croissante. *** ordres lexicographique inverse ; ordres diagonaux

EXEMPLE 3. *** Matrice à coefficients entiers/réels ; irrationnels.

THÉORÈME 1. **** th. de structure de Robbiano pour mention (référence ? Hahn–Banach ?)*

LEMME 1. *Tout ordre total admissible est un* bon *ordre, cést-à-dire que toute chaîne descendante (c'cest-à-dire une suite décroissante) stationne.*

DÉMONSTRATION. Exemple du dictionnaire (même infini) : si on l'ouvre au milieu et qu'on pique un mot, il n'y a qu'un nombre fini de pages précédentes. Par le théorème de Robbiano, nous pouvons nous ramener au cas particulier de l'ordre lexicographique. Celui-ci se traite par récurrence sur le nombre de lettres. □

Remarquons que la propiété est triviale pour les ordres diagonaux (finitude à degré constant).

3. Exposants privilégiés et escaliers

DÉFINITION 2. *En présence d'un ordre total admissible sur les monômes, nous appelerons* exposant privilégié *d'un polynôme* non nul *le plus grand de ses monômes.*

Notons bien que cette notion n'est pas définie pour le polynôme nul. On peut la voir comme l'analogue d'une linéarisation. ***

DÉFINITION 3. *Un* idéal de monômes *est une partie de $\mathcal{M}_n$ stable par multiplication externe : tout multiple d'un élément de cet idéal appartient encore à cet idéal. De manière isomorphe, une* partie stable *de $\mathbb{N}^n$ est stable par addition de quadrant : tout translaté d'un point de cette partie stable par un élément de $\mathbb{N}^n$ est encore dans cette partie stable.*

On parle de manière imagée d'un *escalier* (dessin).

$E(0) =$ DOUTEUX ; néanmoins cela a plus de sens que pour un seul polynôme, tant qu'on ne parle pas de générateurs. $E(A) = \mathbb{N}^n$

4. noethérianité du monoïde des monômes

[Lemme de Dickson]

1. Tout ensemble stable de $\mathbb{N}^n$ est engendré par un nombre fini d'éléments, c'est-à-dire :

$$E = \bigcup_{i=1}^{q} (a_i + \mathbb{N}^n)$$

2. Toute suite croissante de parties stables stationne.

Ces deux propriétés sont équivalentes.

1. implique 2. Soit E_i $i = 1, ldots$ une suite croissante de parties stables. Leur réunion E est encore stable, donc finiment engendré. Chacun des générateurs appartient à un membre de la suite, on prend le dernier qui contient tous les autres précedents et la suite s'arrête dessus.

2. implique 1. Soit donc une partie stable E non finiment engendrée, et a_1 un de ses éléments, qui ne peut donc pas l'engendrer. C'est donc qu'il existe un autre élément a_2 dans E, etc ...

DÉFINITION 4. *Une* base standard *d'un idéal de l'anneau de polynômes est un ensemble de polynômes de l'idéal dont les exposants privilégiés engendrent l'escalier de l'idéal.*

Notons bien que les éléments doivent appartenir à l'idéal mais que rien pour l'instant hormis la terminologie ne prouve qu'ils l'engendrent. C'est le paragraphe suivant qui va nous le démontrer, grâce à une généralisation à plusieurs variables de la notion de division.

5. Divisions

Pour ce faire, commençons par définir une *division élémentaire faible* (resp. *forte* d'un polynôme dividende par un seul diviseur. Ce diviseur s'écrit $exp(f) - reste(f)$, et nous lui associons la règle de réécriture $exp(f) \rightarrow reste(f)$. Elle consiste

à remplacer une occurence éventuelle de $exp(f)$ comme facteur du monôme privilégié (resp. de tout monôme) du dividende par le polynôme $reste(f)$.

L'itération du processus de division faible (resp. forte) s'arrête (noethérianité) sur un reste dont l'exposant privégié n'est plus divisible par $exp(f)$ (resp. dont aucun monôme n'est divisible par $exp(f)$).

Définition 5 (Divisions).

Division faible et forte par une famille (processus non déterministe).

Si un reste d'une division faible d'un dividende par une BS est nul, tout autre reste aussi. Sinon, l'exposant privilégié du reste est unique.

Le reste d'une division forte d'un dividende par une BS est unique. En particulier, la division réalise la projection canonique de l'anneau de polynômes sur son anneau quotient par l'idéal (opération k-linéaire.

Unicité des quotients. Décomposition en somme directe de l'anneau ambiant comme EV en un idéal et l'anneau quotient.

Noethérianité de l'anneau des polynômes (Théorème de Hilbert).

Exercices

Exercice 4.

1. Calculer une base standard de l'idéal engendré par $(x + y - z, x^2 - 2t^2, y^2 - 5t^2)$ pour l'ordre lexicographique induit par $x > y > z > t$, le monôme dominant étant le plus grand.
2. Déduire des calculs précédents que $\sqrt{2} + \sqrt{5}$ est un nombre algébrique sur le corps des rationnels $\mathbb{Q}$, en exhibant un polynôme à une variable à coefficients rationnels dont il est racine.
3. Quel est le résultant de $(y - z)^2 - 2$ et $y^2 - 5$ par rapport à y ?
4. Déduire des calculs précédents que $\mathbb{Q}(\sqrt{2}, \sqrt{5}) = \mathbb{Q}(\sqrt{2} + \sqrt{5})$. Exprimer $\sqrt{2}$ et $\sqrt{5}$ en fonction de $\sqrt{2} + \sqrt{5}$.

Notes

Historique rapide BS/GB, Macaulay/Hironaka/Groebner/Buchberger. Majorant doublement exponentiel du degré d'une BS. Mention et comparaison Hermann/Moreno. Borne inférieure de Mayr-Meyer. Annonce du cas général.

COURS 14

Syzygies et construction des bases standard pour des idéaux homogènes

COURS 15

Fonction et polynôme de Hilbert. Dimension, degré

COURS 16

Triangularisation des idéaux, Nullstellensatz, mise en position de Noether

COURS 17

Théorie de la dimension

COURS 18

Géométrie affine/géométrie projective. Clôture projective

COURS 19

Quête d'une meilleure complexité

Cas particulier des intersections complètes projectives. Rôle crucial des structures de données.

COURS 20

Résolution géométrique I : algorithme

COURS 21

Résolution géométrique II : complexité

Troisième partie

Équations Différentielles et Récurrences Linéaires

COURS 22

Résolution d'équations différentielles linéaires

Notes de Bruno Salvy

Résumé

Les solutions polynomiales ou rationnelles d'équations différentielles linéaires s'obtiennent en utilisant des développements en série et la structure des ensembles de séries solutions.

L'objectif de ce cours est de décrire des algorithmes permettant de calculer les solutions rationnelles d'une équation de la forme

$$Ly(x) = \sum_{k=0}^{n} a_k(x) y^{(k)}(x) = 0, \tag{1}$$

où les coefficients a_k, $k = 0, \dots, n$ sont des polynômes à coefficients dans un corps $\mathbb{K}$. De manière équivalente (voir §1), ces algorithmes permettront de résoudre le système

$$Y'(x) = A(x)Y(x), \tag{2}$$

où $A(x)$ est une matrice de fractions rationnelles de $\mathbb{K}(x)$ et Y un vecteur.

Ces algorithmes seront utilisés dans un cours ultérieur pour le calcul d'intégrales définies.

1. Système et équation

L'équivalence entre équation linéaire d'ordre n et système linéaire d'ordre 1 sur des vecteurs de taille n est classique. Nous détaillons les calculs en jeu.

L'équation (1) est transformée en une équation de la forme (2), en posant $Y = (y_0, \dots, y_{n-1})$ où $y_i = y^{(i)}$ pour $i = 0, \dots, n-1$. La matrice A est alors une matrice compagnon

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \\ -\frac{a_0}{a_n} & \dots & \dots & \dots & -\frac{a_{n-1}}{a_n} \end{pmatrix}. \tag{3}$$

À l'inverse, pour toute matrice A intervenant dans un système de type (2), une équation différentielle de type (1) peut être obtenue pour n'importe quelle combinaison linéaire à coefficients dans $\mathbb{K}$ des coordonnées d'une solution Y. En effet, les dérivées successives $Y, Y', Y'', \dots$ sont des vecteurs dans un espace de dimension n où n est la dimension de la matrice. Il existe donc un $k \leq n$ tel

que $Y, \dots, Y^{(k)}$ soient liés sur $\mathbb{K}$. Multiplier à gauche par un vecteur constant donne l'équation cherchée.

EXERCICE 1. Trouver une équation différentielle linéaire satisfaite par y_1 solution de

$$y_1' = xy_1 - y_2, \quad y_2' = y_1 - xy_2.$$

2. Solutions séries et singularités

Avant de rechercher le développement en série de solutions de l'équation (1) ou du système (2), il est utile de localiser les singularités. Le point de départ est une version du théorème de Cauchy sur les équations différentielles :

THÉORÈME 1. *Si $A(x)$ est une fonction de $\mathbb{C}$ dans $\mathbb{C}^{n\times n}$ analytique dans une région simplement connexe R du plan complexe, alors l'équation*

$$Y'(x) = A(x)Y(x)$$

possède une unique solution telle que $Y(\alpha) = U$ pour tout $\alpha \in R$ et $U \in \mathbb{C}^n$. Cette solution est analytique dans R.

L'application de ce résultat à l'équation (2) montre qu'en tout point où A est analytique (développable en série entière), il existe une base de solutions séries entières convergentes. Au vu de la matrice compagnon (3), il en va de même pour les solutions de l'équation (1) en tout point où le coefficient de tête a_n est non-nul. *A contrario*, les seuls points où les solutions peuvent ne pas admettre de solution série sont les racines de a_n.

DEFINITION 1. On dit que $\alpha \in \mathbb{C}$ est un point *ordinaire* de l'équation (1) si $a_n(a) \neq 0$. On dit qu'il est *singulier* dans le cas contraire.

EXEMPLE 1. La fraction rationnelle $y = 1/(1-x)$ est solution de l'équation

$$(1-x)y'(x) - y(x) = 0.$$

Le complexe 1 est singularité de la solution et donc nécessairement point singulier de l'équation, ce qui s'y traduit par l'annulation du coefficient de tête.

EXEMPLE 2. Le polynôme x^{10} est solution de l'équation

$$10xy'(x) - y(x) = 0.$$

La solution n'a pas de point singulier à distance finie, mais le complexe 0 est point singulier de l'équation ; le théorème de Cauchy ne s'y applique pas.

Une autre conséquence utile de ce théorème est que les fonctions D-finies ne peuvent avoir qu'un nombre fini de singularités (les racines du coefficient de tête). Il s'en déduit des résultats négatifs.

EXEMPLE 3. La fonction $1/\sin x$ ne satisfait pas d'équation différentielle linéaire à coefficients polynomiaux.

EXEMPLE 4. La suite des nombres de Bernoulli, qui interviennent en particulier dans la formule d'Euler-Maclaurin, ne peut être solution d'une récurrence linéaire à coefficients polynomiaux, puisque la série génératrice exponentielle

$$\sum_{n=0}^{\infty} B_n \frac{z^n}{n!} = \frac{z}{\exp(z)-1}$$

a une infinité de pôles (aux $2ik\pi$, $k \in \mathbb{Z}^\star$).

Le théorème ci-dessus entraîne aussi la possibilité de développer les solutions en série. Soit

$$\lambda := \min_k(\mathrm{val}(a_k) - k), \quad \mu := \max_k(\deg(a_k) - k),$$

où $\mathrm{val}(p(x))$ désigne le plus grand entier m tel que x^m divise $p(x)$, que l'on appelle la *valuation* de p. Alors les coefficients de l'équation (1) se récrivent

$$a_k(x) = \sum_{i=\lambda}^{\mu} a_{k,i} x^{i+k}.$$

Ceci permet de définir les polynômes

$$u_i(x) = \sum_{k=0}^{n} a_{k,i} x(x-1)\cdots(x-k+1),$$

de sorte que si $y = \sum_{m=-K}^{\infty} y_i x^i$ ($K \in \mathbb{Z}$) est une série de Laurent solution de (1), ses coefficients satisfont la récurrence

$$u_\lambda(i-\lambda)y_{i-\lambda} + \cdots + u_\mu(i-\mu)y_{i-\mu} = 0 \tag{4}$$

pour tout $i \in \mathbb{Z}$ (les y_i d'indice inférieur à $-K$ sont supposés nuls).

Definition 2. Le polynôme u_λ s'appelle *polynôme indiciel* de l'équation (1) à l'origine ; le polynôme u_μ est son *polynôme indiciel à l'infini*.

En changeant la variable x en $x + \alpha$ dans l'équation, le même calcul fournit deux polynômes. C'est un exercice de montrer que le polynôme indiciel à l'infini est inchangé. L'autre s'appelle le polynôme indiciel en α.

Proposition 1. *Si 0 est un point ordinaire pour l'équation (1), alors pour tout $U = (u_0, \ldots, u_{n-1}) \in \mathbb{C}^n$, les coefficients du développement en série de la solution y de (1) telle que $y^{(k)}(0) = u_k$, $k = 0, \ldots, n-1$ sont donnés par la récurrence linéaire (4).*

Démonstration. Il suffit de prouver que le coefficient de tête de la récurrence, à savoir $u_\lambda(i-\lambda)$, ne s'annule pas pour $i - \lambda = n, n+1, \ldots$, ce qui permet alors le calcul de $y_{i+\lambda}$ à partir des précédents. En effet, lorsque l'origine est ordinaire, le coefficient a_n est tel que $a_n(0) \neq 0$ et donc $\mathrm{val}(a_n) - n = -n$ est minimal. Donc $\lambda = n$ et $u_\lambda(x) = a_n(0)x(x-1)\cdots(x-n+1)$, ce qui permet de conclure. □

En effectuant le changement de variable $x \mapsto x + \alpha$, le même raisonnement s'applique à tout point α ordinaire.

3. Solutions polynomiales

S'il existe une solution polynomiale de degré N, l'équation (4) avec $i - \mu = N$ montre que $u_\mu(N) = 0$. Ceci fournit un procédé pour trouver les degrés possibles des solutions polynomiales.

Supposons d'abord que l'origine est un point ordinaire de l'équation. Alors une solution polynomiale est une solution dont le développement en série n'a que des coefficients nuls à partir du degré N. D'après la récurrence (4), il suffit que les coefficients des degrés $N+1$ à $N + \mu - \lambda$ le soient. L'idée est alors de constater que cette observation se ramène à un calcul d'algèbre linéaire. Voici le détail de l'algorithme :

1. Calculer la récurrence (4) ;
2. Calculer la plus grande racine entière positive N de u_μ. Si N n'existe pas, il n'existe pas de solution polynomiale non-nulle ;
3. Pour $0 \leq i \leq n-1$, utiliser la récurrence pour calculer une série solution
$$y_i = x^i + \sum_{j=n}^{N+\mu-\lambda} y_{i,j} x^j + O(x^{N+\mu-\lambda+1});$$
4. Former la matrice $M = [y_{i,j}]$, $0 \leq i < n$, $N+1 \leq j \leq N+\mu-\lambda$;
5. Calculer une base B du noyau de la transposée M^t ;
6. L'ensemble des $c_0 y_0 + \cdots + c_{n-1} y_{n-1}$ pour $(c_0, \ldots, c_{n-1})$ dans B forme une base de l'espace des solutions polynomiales.

Si l'origine n'est pas un point ordinaire de l'équation, il n'est pas garanti que la récurrence (4) permette de calculer les séries solutions. Deux approches sont alors possibles : soit on étend les calculs précédents pour s'adapter au cas singulier, soit, plus simplement, on trouve un point ordinaire (il y en a au moins un parmi $0, 1, \ldots, \deg(a_n)$ et on effectue les calculs en ce point.

4. Solutions rationnelles

Les solutions rationnelles ne peuvent avoir de pôle (zéro de leur dénominateur) qu'en une singularité de l'équation. De la même manière que pour les degrés des solutions polynomailes, les multiplicités possibles des pôles sont données par les racines entières négatives du polynôme indiciel en ces singularités. Ceci conduit à un algorithme simple, essentiellement dû à Liouville, pour calculer les solutions rationnelles de (1).

1. En toute racine α de a_n :
   - calculer le polynôme indiciel $p_\alpha(n)$;
   - calculer la plus petite racine entière négative N_α de p_α, s'il n'en existe pas, faire $N_\alpha := 0$;
2. Former le polynôme $P = \prod_{a_n(\alpha)=0} (x-\alpha)^{-N_\alpha}$;
3. Effectuer le changement de fonction inconnue $y = Y/P$ et réduire au même dénominateur ;
4. Chercher une base B des solutions polynomiales de cette nouvelle équation. Une base des solutions rationnelles est formée des fractions $\{b/P, b \in B\}$.

La preuve de l'algorithme se réduit à observer que P est un multiple du dénominateur de toute solution rationnelle.

Cet algorithme permet également de trouver les solutions rationnelles du système (2), en se ramenant à une équation. Il existe aussi d'autres algorithmes plus directs.

Notes

Les idées de base de l'algorithme de recherche de solutions rationnelles sont dues à Liouville [3] qui donne également une méthode par coefficients indéterminés pour trouver les solutions polynomiales. La présentation qui utilise les récurrences donne un algorithme de même complexité mais fait mieux ressortir la structure du calcul [1].

La recherche de solutions d'équations différentielles linéaires ne s'arrête pas aux solutions rationnelles. En utilisant la théorie de Galois différentielle, il existe une algorithmique sophistiquée de recherche de solutions *liouvilliennes* (c'est-à-dire formées par l'application répétée d'exponentielles, d'intégrales et de prise de racines de polynômes). Les calculs se ramènent à la recherche présentée ici de solutions rationnelles pour des équations (les puissances symétriques) formées à partir de l'équation de départ [**4**, **5**, **6**].

Bibliographie

[1] Abramov (Sergei A.), Bronstein (Manuel), and Petkovšek (Marko). – On polynomial solutions of linear operator equations. In Levelt (A. H. M.) (editor), *Symbolic and Algebraic Computation*. pp. 290–296. – ACM Press, New York, 1995. Proceedings of ISSAC'95, July 1995, Montreal, Canada.

[2] Ince (E. L.). – *Ordinary differential equations*. – Dover Publications, New York, 1956, viii+558p. Reprint of the 1926 edition.

[3] Liouville (Joseph). – Second mémoire sur la détermination des intégrales dont la valeur est algébrique. *Journal de l'École polytechnique*, vol. 14, 1833, pp. 149–193.

[4] Marotte (F. M.). – *Les équations différentielles linéaires et la théorie des groupes*. – PhD thesis, Faculté des Sciences de Paris, 1898.

[5] Singer (Michael F.). – Liouvillian solutions of n-th order homogeneous linear differential equations. *American Journal of Mathematics*, vol. 103, n° 4, 1981, pp. 661–682.

[6] Singer (Michael F.) and Ulmer (Felix). – Linear differential equations and products of linear forms. *Journal of Pure and Applied Algebra*, vol. 117/118, 1997, pp. 549–563.

COURS 23

Équations fonctionnelles linéaires et polynômes tordus

Notes de Frédéric Chyzak

Résumé

Une certaine variété de polynômes non-commutatifs fournit une représentation unifiée pour une large classe d'équations fonctionnelles linéaires. Celle-ci s'avère bien adaptée pour les calculs. Nous réinterprétons nombre des algorithmes vus dans ce cours dans ce point de vue.

1. Des polynômes non-commutatifs pour calculer avec des opérateurs linéaires

Dans les années 1930, le mathématicien Oystein Ore (1899–1968) s'est intéressé à la résolution de systèmes linéaires reliant des dérivées $f_i^{(j)}(x)$, des décalées $f_i(x+j)$, ou les subtitutions $f_i(q^j x)$ de fonctions inconnues $f_i(x)$. À cette fin, il a introduit de nouvelles familles de polynômes en une variable ayant la propriété que cette variable ne commute pas avec les coefficients des polynômes.

Rappelons la relation de Leibniz pour deux fonctions quelconques f et g :

$$(fg)'(x) = f'(x)g(x) + f(x)g'(x).$$

En notant D l'opérateur de dérivation, M celui qui à une fonction f associe la fonction donnée par $M(f)(x) = xf(x)$, I l'opérateur identité sur les fonctions, et $\circ$ la composition d'opérateurs, la règle de Leibniz donne, pour $f(x) = x$ et g quelconque,

$$(D \circ M)(g) = D\big(M(g)\big) = M\big(D(g)\big) + g = (M \circ D + I)(g).$$

L'identité étant vérifiée par toute g, on obtient l'égalité $D \circ M = M \circ D + I$ entre opérateurs linéaires différentiels. D'autres opérateurs vérifient des analogues de la règle de Leibniz : l'opérateur Δ de différence finie, donné par $\Delta(f)(x) = f(x+1) - f(x)$; l'opérateur $S = \Delta + I$ de décalage, donné par $S(f)(x) = f(x+1)$; pour une constante q fixée autre que 0 et 1, l'opérateur H de dilatation, donné par $\Delta(f)(x) = f(qx)$. On a les relations :

$$\Delta(fg)(x) = f(x+1)\Delta(g)(x) + \Delta(f)(x)g(x),$$
$$(fg)(x+1) = f(x+1)g(x+1), \qquad (fg)(qx) = f(qx)g(qx),$$

qui mènent aux relations $\Delta \circ M = (M + I) \circ \Delta + I$, $S \circ M = (M + I) \circ S$, $H \circ M = Q \circ M \circ H$ entre opérateurs linéaires, après avoir introduit un nouvel opérateur Q donné par $Q(f)(x) = qf(x)$.

Le point de vue d'Ore est d'abstraire ces différents contextes d'opérateurs dans un même moule algébrique.

DÉFINITION. *Soit A un anneau commutatif unitaire de caractéristique zéro, que nous supposons muni d'un endomorphisme injectif σ et d'une σ-dérivation δ, au sens où pour tout a et tout b de A,*

$$\sigma(a+b)=\sigma(a)+\sigma(b),\qquad \sigma(ab)=\sigma(a)\sigma(b),\qquad \delta(ab)=\sigma(a)\delta(b)+\delta(a)b.$$

Pour une nouvelle variable ∂, on appelle anneau de polynômes tordus *l'algèbre sur A engendrée par ∂ et les relations, pour tout a de A,*

$$\partial a=\sigma(a)\partial+\delta(a).$$

On note cet anneau $A\langle\partial;\sigma,\delta\rangle$.

(La terminologie « polynôme tordu » est la traduction de l'anglais « *skew polynomial* », où « skew » signifie « de biais », « oblique ». Certains auteurs ont proposé la traduction « polynôme gauche », où « gauche » a le sens de « voilé », par opposition à « plan ». Mais nous voulons éviter ici toute confusion avec des notions algébriques de multiple, module, fraction, etc, pour lesquelles « gauche » a le sens opposé de « droite ».)

Des choix adéquats de σ et δ nous font retrouver les quelques exemples donnés plus haut. Pour simplifier la notation, nous supposons que A peut s'identifier à un bon espace de fonctions. On a alors, en notant 0 l'application qui a toute fonction associe la fonction constante nulle :

- $\mathbb{Q}(x)\langle\partial;I,D\rangle$ représente l'algèbre des opérateurs différentiels linéaires ;
- $\mathbb{Q}(x)\langle\partial;S,0\rangle$ représente l'algèbre des opérateurs de récurrence ;
- $\mathbb{Q}(x)\langle\partial;S,\Delta\rangle$ représente l'algèbre des opérateurs de différence finie ;
- $\mathbb{Q}(x)\langle\partial;H,0\rangle$ pour $q\in\mathbb{Q}(x)\setminus\{0,1\}$ représente l'algèbre des opérateurs de q-dilatation ;
- $\mathbb{Q}(x)\langle\partial;I,0\rangle$ n'est autre que l'anneau commutatif $\mathbb{Q}(x)[\partial]$ des polynômes usuels.

Toutes ces algèbres d'opérateurs sont à coefficients dans $\mathbb{Q}(x)$; on dispose aussi d'analogues pour $A=\mathbb{Q}[x]$ et $A=\mathbb{Q}[x,x^{-1}]$.

On fera attention à la notation. Si la composition entre opérateurs est notée par $\circ$, nous ne ferons qu'une simple juxtaposition pour le produit de polynômes tordus, et nous noterons 1 l'élément neutre pour le produit de polynômes tordus. Néanmoins, fera l'abus de notation de noter de la même façon, D_x, la dérivation par rapport à x quel que soit l'anneau A, et I, sans indice, pour l'identité de n'importe quel A. De plus, nous noterons simplement x pour M. Ainsi, on a :

- $\partial x=x\partial+1$ dans $\mathbb{Q}(x)\langle\partial;I,D\rangle$;
- $\partial x=(x+1)\partial$ dans $\mathbb{Q}(x)\langle\partial;S,0\rangle$;
- $\partial x=(x+1)\partial+1$ dans $\mathbb{Q}(x)\langle\partial;S,\Delta\rangle$;
- $\partial x=qx\partial$ dans $\mathbb{Q}(x)\langle\partial;H,0\rangle$;
- $\partial x=x\partial$ dans $\mathbb{Q}(x)\langle\partial;I,0\rangle=\mathbb{Q}(x)[\partial]$.

Le cas $\delta=0$ est fréquent, et on écrit alors $A\langle\partial;\sigma\rangle$, sans référence au 0. De même que dans le cas commutatif on définit les polynômes de Laurent, dont l'algèbre est notée $A[X,X^{-1}]$, et dans laquelle $XX^{-1}=X^{-1}X=1$, le cas où σ est inversible et autre que l'identité permet de représenter des opérateurs qui possèdent un inverse. Dans ce cas, on notera $A\langle\partial,\partial^{-1};\sigma\rangle$ l'algèbre où $\partial a=\sigma(a)\partial$, $\partial^{-1}a=\sigma^{-1}(a)\partial^{-1}$, $\partial\partial^{-1}=\partial^{-1}\partial=1$.

Pour finir de se détacher de la notation en termes d'opérateurs, on fait agir les anneaux de polynômes tordus sur les espaces de fonctions, au sens de l'action d'un anneau sur un module. Rappelons qu'un module M sur un anneau A est un ensemble non-vide, muni d'une loi $+$ en faisant un groupe additif, stable sous l'action d'une produit externe par les éléments de A, tel que l'action par produit externe par 1 soit l'identité, et vérifiant les formules $(PQ)\cdot f = P\cdot(Q\cdot f)$ et $(P+Q)\cdot f = (P\cdot f)+(Q\cdot f)$. Un anneau de polynômes tordus n'a pas d'action unique sur un espace de fonctions donné, mais dans la suite de ce texte, nous adoptons les conventions qu'un anneau de la forme $A\langle\partial;\sigma\rangle$ agit par $\partial\cdot f = \sigma(f)$, qu'un anneau de la forme $A\langle\partial;\sigma\rangle$ agit par $\partial\cdot f = \sigma(f)$, et que les coefficients dans A agissent par simple multiplication, $a\cdot f = af$.

Munis de cette notation générique, nous allons maintenant réexprimer des algorithmes déjà vus et petit à petit introduire de nouveaux calculs.

2. Clôtures par morphismes entre anneaux de polynômes tordus

Dans cette section, nous sommes amenés à considérer simulanément des fonctions de x et des fonctions d'une autre variable. Aussi indiquerons-nous en indice de D, S, ∂, σ, δ, etc, la variable à laquelle ces objets font référence. De plus, les anneaux A qui servent à construire les anneaux de polynômes tordus sont de la forme $\mathbb{Q}[x]$ ou $\mathbb{Q}[x,x^{-1}]$.

2.1. Récurrence sur les coefficients extraits d'une série D-finie et série génératrice d'une suite P-récursive. On a déjà vu que lorsqu'une série $f=\sum_{n\geq 0}u_n x^n$ est D-finie, ses coefficients vérifient une relation de récurrence finie, autrement dit, que la suite $u=(u_n)_{c\geq 0}$ est P-récursive. La preuve repose sur les identités

$$xf = \sum_{n\geq 1} u_{n-1}x^n = \sum_{n\geq 1}(\partial_n^{-1}\cdot c)(n)\,x^n$$

et

$$D_x(f) = f' = \sum_{n\geq 0}(n+1)u_{n+1}x^n = \sum_{n\geq 0}\big((n+1)\partial_n\cdot c\big)(n)\,x^n,$$

où nous avons introduit l'anneau $\mathbb{Q}[n]\langle\partial_n,\partial_n^{-1};S_n\rangle$. Par récurrence, ceci donne

$$\begin{aligned} x^\alpha D_x^\beta(f) &= \sum_{n\geq\alpha}\Big(\partial_n^{-\alpha}\big((n+1)\partial_n\big)^\beta\cdot c\Big)(n)\,x^n \\ &= \sum_{n\geq\alpha}\big((n+1-\alpha)\cdots(n+\beta-\alpha)\partial_n^{\beta-\alpha}\cdot c\big)(n)\,x^n. \end{aligned}$$

Pour une série f solution de l'équation différentielle

$$a_r(x)f^{(r)}(x)+\cdots+a_0(x)f(x)=0$$

où les a_i sont dans $\mathbb{Q}[x]$, nous obtenons ainsi une récurrence sur c, valable pour des n assez grands. Cette récurrence s'exprime en termes de polynômes tordus de la façon suivante. On représente l'opérateur différentiel associé à l'équation par le polynôme tordu $L=a_r(x)\partial_x^r+\cdots+a_0(x)$ dans $\mathbb{Q}[x]\langle\partial_x;I,D_x\rangle$ sur $\mathbb{Q}$. De la sorte, l'équation différentielle s'écrit $L\cdot f=0$. On introduit aussi l'algèbre $\mathbb{Q}[n]\langle\partial_n,\partial_n^{-1};S_n\rangle$ et le morphisme d'algèbres μ défini par $\mu(x)=\partial_n^{-1}$ et $\mu(\partial_x)=(n+1)\partial_n$. Alors, la

suite c des coefficients satisfait à la récurrence représentée par l'image $\mu(L)$. Pour comprendre pour quels n cette récurrence est valide, écrivons

$$\mu(L) = b_p(n)\partial_n^p + \cdots + b_q(n)\partial_n^q$$

pour $p \leq q$ et $b_p b_q \neq 0$. Alors, la récurrence prend la forme

$$\big(\mu(L)\cdot u\big)(n) = b_p(n)u_{n+p} + \cdots + b_q(n)u_{n+q} = 0$$

et est vérifiée pour tout n si $p \geq 0$ et pour tout $n \geq -p$ si $p < 0$.

De façon duale, une suite P-récursive u a une série génératrice $f = \sum_{n\geq 0} u_n x^n$ D-finie, ce que nous allons retrouver en termes de polynômes tordus. En effet, les formules

$$\sum_{n\geq 0} n u_n x^n = x\partial_x \cdot f \qquad \text{et} \qquad \sum_{n\geq 0} u_{n+1} x^n = x^{-1} f$$

donnent par récurrence

$$\sum_{n\geq 0} n^\alpha u_{n+\beta} x^n = (x\partial_x)^\alpha x^{-\beta}\cdot f = x^{-\beta}(x\partial_x - \beta)^\alpha \cdot f,$$

et fournissent un autre morphisme, ν, de $\mathbb{Q}[n]\langle \partial_n; S_n\rangle$ dans $\mathbb{Q}[x, x^{-1}]\langle \partial_x; I, D_x\rangle$, donné par $\nu(n) = x\partial_x$ et par $\nu(\partial_n) = x^{-1}$. Pour une suite u solution de l'équation de récurrence

$$b_p(n)u_{n+r} + \cdots + a_0(n)u_n = 0$$

où les b_i sont dans $\mathbb{Q}[n]$, nous introduisons le polynôme tordu $P = b_p(n)\partial_n^r + \cdots + b_0(n)$ de $\mathbb{Q}[n]\langle \partial_n; S_n\rangle$. Pour obtenons une relation différentielle sur la série génératrice f, nous considérons $\nu(P)$ que nous écrivons

$$\nu(P) = a_0(x) + \cdots + a_r(x)\partial_x^r.$$

Alors la série f satisfait à la relation différentielle

$$a_0(x)f(x) + \cdots + a_r(x)f^{(r)}(x) = 0.$$

Algébriquement, les propriétés précédentes s'expriment par le fait que μ s'étend en un isomorphisme d'algèbres entre $\mathbb{Q}[x, x^{-1}]\langle \partial_x; I, D_x\rangle$ et $\mathbb{Q}[n]\langle \partial_n, \partial_n^{-1}; S_n\rangle$, dont l'inverse étend ν.

2.2. Séries binomiales.

Exercice 1. Une série binomiale est une série de la forme $\sum_{n\geq 0} u_n \binom{x}{n}$. Montrer que les solutions en série binomiale d'une équation fonctionnelle à différence

$$a_r(x)f(x+r) + \cdots + a_0(x)f(x) = 0$$

ont des coefficients u_n qui vérifient une récurrence et expliciter le morphisme entre algèbres de polynômes tordus correspondant.

2.3. Changements de variables. Lorsqu'une série D-finie $f(x)$ est solution d'une équation différentielle $L\cdot f = 0$ donnée par un polynôme tordu

$$L = L(x, \partial_x) = a_r(x)\partial_x^r + \cdots + a_0(x),$$

la série $f(\lambda x)$ est solution de l'équation différentielle associée à

$$L(\lambda x, \lambda^{-1}\partial_x) = a_r(\lambda x)\lambda^{-r}\partial_x^r + \cdots + a_0(\lambda x),$$

ce qui est encore le résultat d'un morphisme d'algèbres.

Lorsque f est une fonction D-finie, la fonction $z \mapsto f(1/z)$ est elle aussi D-finie, en z cette fois, pour autant que la fonction composée ait un sens. En effet,

pour toute fonction g, notons $\tilde{g}(z) = g(1/z)$ (avec la même réserve de définition). Puisque $g(x) = \tilde{g}(1/x)$, par dérivation on a $g'(x) = -\tilde{g}'(1/x)/x^2$, ce qui est l'évaluation en $z = 1/x$ de $-z^2\partial_z \cdot \tilde{g}$. Autrement dit, on a $\widetilde{g'} = -z^2\partial_z \cdot \tilde{g}$, d'où par récurrence $\widetilde{g^{(\beta)}} = (-z^2\partial_z)^\beta \cdot \tilde{g}$. Ainsi, $\tilde{f}$ est D-finie, donnée comme vérifiant l'équation différentielle associée à l'image de L par le morphisme de $\mathbb{Q}[x]\langle\partial_x; I, D_x\rangle$ dans $\mathbb{Q}[z, z^{-1}]\langle\partial_z; I, D_z\rangle$ qui envoie x sur z^{-1} et ∂_x sur $-z^2\partial_z$.

Exercice 2. Plus généralement, la fonction obtenue par substitution rationnelle de la variable, donnée par $h(u) = f\big(r(u)\big)$, est encore D-finie. Nous laissons en exercice le soin de montrer ce résultat par la même approche dans le cas où la dérivée r' s'exprime comme une fraction rationnelle en r.

3. Division euclidienne

Dans cette section et les suivantes, nous nous appuyons sur des propriétés particulières des anneaux de polynômes tordus quand l'anneau A de la construction est un corps, que nous prendrons de la forme $\mathbb{Q}(x)$.

La commutation $\partial a = \sigma(a)\partial + \delta(a)$ dans $\mathbb{Q}(x)\langle\partial; \sigma, \delta\rangle$ permet d'écrire tout polynôme tordu sous la forme $a_0(x)+\cdots+a_r(x)\partial^r$, pour des fractions rationnelles a_i de $\mathbb{Q}(x)$. Une conséquence de l'injectivité de σ est l'existence d'un degré en ∂ bien défini, étant l'entier r de l'écriture précédente lorsque a_r est non-nulle. En particulier, le degré d'un produit L_1L_2 de polynômes tordus est la somme des degrés des L_i. Il s'ensuit que la division euclidienne du cas commutatif, et toute la théorie qui en découle, se transpose avec peu d'altérations dans le cas tordu.

La différence principale avec le cas commutatif est qu'on distingue division euclidienne à gauche et division euclidienne à droite. Vu notre interprétation en termes d'opérateurs linéaires, nous ne considèrerons que la division à droite, qui se fait en retranchant des multiples à gauche. Soit à diviser $A = a_r(x)\partial^r + \cdots + a_0(x)$ de degré r par $B = b_s(x)\partial^s + \cdots + b_0(x)$ de degré s. On suppose $s \leq r$. Alors,

$$\partial^{r-s}B = \sigma^{r-s}\big(b_s(x)\big)\partial^r + \text{termes d'ordre inférieur},$$

où la puissance de σ représente une itération (par composition), et ainsi

$$A - a_r(x)\sigma^{r-s}\big(b_s(x)\big)^{-1}\partial^{r-s}B$$

est de degré strictement inférieur à r. Cette étape de réduction est l'étape élémentaire de la division euclidienne. En itérant le procédé, on aboutit à un reste R de degré strictement inférieur à s. En regroupant les facteurs gauches, on obtient un quotient à gauche Q tel que $A = QB + R$.

Exemple 1. On considère l'anneau $\mathbb{Q}(n)\langle\partial_n; S_n\rangle$ des polynômes tordus représentant les opérateurs de décalage. La division de $A = (n^2-1)\partial_n^2 - (n^3+3n^2+n-2)\partial_n + (n^3+3n^2+2n)$, qui annule les combinaisons linéaires de $n!$ et n, par $B = n\partial_n^2 - (n^2+3n+1)\partial_n + (n^2+2n+1)$, qui annule les combinaisons linéaires $n!$ et 1, s'écrit

$$A = n^{-1}(n^2-1)B - n^{-1}(n^2+n+1)\big(\partial_n - (n+1)\big).$$

Le reste est multiple de $\partial_n - (n+1)$, qui représente la récurrence $u_{n+1} = (n+1)u_n$, vérifiée par la factorielle.

Notons une propriété de cette division : si A est multiplié à gauche par un facteur $m(x)$ sans que B ne soit changé, alors Q et R sont multipliés à gauche

par le même facteur $m(x)$. Ceci ne vaut plus (en général) pour un facteur faisant intervenir ∂.

La division euclidienne nous donne une nouvelle interprétation du calcul du N-ième terme d'une suite P-récursive $u = (u_n)$ relativement à $\mathbb{Q}(n)\langle\partial_n; S_n\rangle$. Supposons que u soit solution de l'équation de récurrence

$$a_r(n)u_{n+r} + \cdots + a_r(n)u_n = 0.$$

En déroulant la récurrence, on voit que u_N peut, pour tout N sauf annulation malvenue de a_r, se mettre sous la forme $\alpha_{r-1,N}u_{r-1} + \cdots + \alpha_{0,N}u_0$. Plus généralement, on a une relation qui récrit u_{n+N} en terme de u_{n+r-1}, …, u_n. Pour l'obtenir, associons à la récurrence sur u le polynôme tordu $P = a_r(n)\partial_n^r + \cdots + a_r(n)$. Pour un N donné, la division euclidienne de ∂_n^N par P s'écrit

$$\partial_n^N = Q_N(n)P + \alpha_{r-1,N}(n)\partial_n^{r-1} + \cdots + \alpha_{0,N}(n)$$

pour des fractions rationnelles $\alpha_{i,N}(n)$. Après application sur u et évaluation en n, nous obtenons

$$u_{n+N} = 0 + \alpha_{r-1,N}(n)u_{n+r-1} + \cdots + \alpha_{0,N}(n)u_n,$$

d'où le résultat annoncé pour $\alpha_{i,N} = \alpha_{i,N}(0)$.

Exercice 3. Nous laissons le lecteur se convaincre que la récriture d'une dérivée $f^{(N)}$ d'une fonction D-finie f décrite par une équation différentielle d'ordre r en terme de ses dérivées d'ordre strictement inférieur à r s'interprète de façon analogue comme le calcul d'un reste de division euclidienne.

Le même ingrédient se retrouve dans l'algorithme donnant la clôture par addition de deux fonctions D-finies ou de deux suites P-récursives : pour deux objets f et g à additionner, décrits comme solutions des équations respectives $L_f \cdot f = 0$ et $L_g \cdot g = 0$ pour des polynômes tordus de degrés respectifs r et s d'un anneau adéquat $A\langle\partial; \sigma, \delta\rangle$, l'algorithme exprime pour des i successifs $\partial^i \cdot (f + g)$ sous la forme $(\partial^i \bmod L_f) \cdot f + (\partial^i \bmod L_g) \cdot g$, où la notation $A \bmod B$ note le reste de la division euclidienne à droite de A par B. Lorsque suffisamment de i ont été considérés, l'algorithme qui a jusqu'à présent été donné calcule par de l'algèbre linéaire des cofacteurs a_0, …, a_{r+s} tels que

$$\sum_{i=0}^{r+s} a_i(\partial^i \bmod L_f) = 0 \qquad \text{et} \qquad \sum_{i=0}^{r+s} a_i(\partial^i \bmod L_g) = 0.$$

Notons que $P = \sum_{i=0}^{r+s} a_i\partial^i$ est un multiple commun à gauche de L_f et de L_g, puisque $P \bmod L_f = P \bmod L_g = 0$.

4. Recherche de solutions et factorisation d'opérateurs

Comme pour les anneaux de polynômes commutatifs usuels, une notion de factorisation est présente pour les anneaux de polynômes tordus. Une nuance importante réside dans le lien entre les « zéros » des polynômes tordus et la position des facteurs. Nous allons voir que la factorisation de polynômes tordus se relie aux algorithmes vus en cours pour la recherche de solutions polynomiales, rationnelles, et hypergéométriques dans le cas de récurrences.

Dans le cas d'un polynôme commutatif h se factorisant sous la forme fg pour des facteurs polynomiaux de degré au moins 2, tout zéro de f et tout zéro de g est zéro de h ; à l'inverse, quitte à se placer dans une clôture algébrique, tout zéro α de h

en fournit un facteur $x - \alpha$ et un quotient exact $f(x)$ tel que $h(x) = f(x)(x - \alpha)$. Dans le cas tordu, une factorisation $L = PQ$ dans $A\langle\partial; \sigma, \delta\rangle$ (où A est un corps) a des propriétés différentes selon le facteur : une solution f de l'équation $Q \cdot f = 0$ est encore solution de $L \cdot f = 0$, car $L \cdot f = P \cdot (Q \cdot f) = P \cdot 0 = 0$; mais une solution g de P ne donne lieu à des solutions f de L que par la relation $Q \cdot f = g$. Inversement, une solution f de L donne lieu à un facteur droit d'ordre 1 de L, quitte à étendre A par f et tous ses itérés par σ et δ. Ce facteur est de la forme $\partial - (\partial \cdot f)/f$, c'est-à-dire $\partial - \delta(f)/f$ ou $\partial - \sigma(f)/f$ selon l'action de l'anneau de polynômes tordus sur les fonctions.

Les algorithmes de recherche de solutions dans des classes particulières fournissent donc implicitement, pour chaque solution trouvée, un facteur droit d'ordre 1. Plus précisément, dans le cas différentiel, une solution polynomiale ou rationnelle f d'une équation $L \cdot f = 0$ pour L dans l'anneau $\mathbb{Q}(x)\langle\partial_x; I, D_x\rangle$ fournit un facteur droit $D = \partial_x - D_x(f)/f$ où $D_x(f)/f$ est rationnel ; dans le cas à récurrence, une solution polynomiale, rationnelle ou hypergéométrique f d'une équation $L \cdot f = 0$ pour L dans l'anneau $\mathbb{Q}(x)\langle\partial_x; S_x\rangle$ fournit un facteur droit $D = \partial_x - S_x(f)/f$ où $S_x(f)/f$ est rationnel. Dans les deux cas, le quotient Q tel que $L = QD$ est aussi à coefficients rationnels.

Exercice 4. Un antimorphisme ϕ entre anneaux est une application $\mathbb{Q}$-linéaire qui renverse les produits : $\phi(PQ) = \phi(Q)\phi(P)$. Montrer l'existence d'antimorphismes $\mu : \mathbb{Q}(x)\langle\partial_x; I, D_x\rangle \to \mathbb{Q}(u)\langle\partial_u; I, D_u\rangle$ et $\nu : \mathbb{Q}(x)\langle\partial_x; S_x\rangle \to \mathbb{Q}(u)\langle\partial_u; S_u\rangle$, définis par les relations

$$\mu(x) = u, \quad \mu(\partial_x) = -\partial_u, \qquad \text{et} \qquad \nu(x) = -u, \quad \nu(\partial_x) = \partial_u.$$

Expliquer comment ces antimorphismes fournissent des facteurs gauches d'ordre 1 de polynômes tordus.

5. Algorithme d'Euclide

Rappelons qu'un idéal d'un anneau commutatif unitaire A est un sous-groupe additif de A clos par multiplication par les éléments de A. Il est classique que les anneaux commutatifs euclidiens — ceux dans lesquels l'existence d'un degré permet une division euclidienne — sont principaux — tout idéal peut être engendré par un unique générateur. C'est le cas des anneaux de polynômes commutatifs à coefficients dans un corps. Le p. g. c. d. p de deux polynômes f et g est alors l'unique polynôme unitaire engendrant l'idéal (f, g), somme des idéaux (f) et (g). Il se calcule comme dernier reste non-nul par l'algorithme d'Euclide.

Pour un anneau de polynômes tordus $A = K\langle\partial; \sigma, \delta\rangle$ sur un corps K, la situation est la même si on prend soin de ne considérer que des idéaux à gauche, c'est-à-dire avec la clôture par multiplication à gauche par les éléments de A. Les notions qui en découlent sont celles de divisions euclidiennes à droite et de plus grands communs diviseurs à droite (p. g. c. d. d.). Soient P_0 et P_1 deux polynômes de A. Si P_1 n'est pas nul, on écrit la division euclidienne de P_0 par P_1, sous la forme $P_0 = Q_0 P_1 + P_2$. Tant que P_{i+2} n'est pas nul, on itère en divisant P_{i+1} par P_{i+2}. Soit j la valeur finale de i, telle que $P_{j+1} \neq 0$ et $P_{j+2} = 0$. Alors :

$$\begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = \begin{bmatrix} Q_0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \cdots = \begin{bmatrix} Q_0 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} Q_j & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_{j+1} \\ 0 \end{bmatrix},$$

d'où on déduit que P_{j+1} divise P_0 et P_1 à droite : $P_0 = FP_{j+1}$ et $P_1 = GP_{j+1}$ pour des polynômes tordus F et G adéquats. Puis en inversant les matrices

$$\begin{bmatrix} P_{j+1} \\ 0 \end{bmatrix} = \begin{bmatrix} U & V \\ R & S \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} \qquad \text{pour} \qquad \begin{bmatrix} U & V \\ R & S \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q_j \end{bmatrix} \cdots \begin{bmatrix} 0 & 1 \\ 1 & -Q_0 \end{bmatrix}.$$

En particulier, $P_{j+1} = UP_0 + VP_1$ est élément de l'idéal à gauche $AP_0 + AP_1$. Un élément quelconque $L = MP_0 + NP_1$ de cet idéal est aussi multiple de P_{j+1} : $L = (MF+NG)P_{j+1}$. En normalisant P_{j+1} pour le rendre unitaire, on obtient donc un p. g. c. d. d. distingué de P_0 et P_1. Par ailleurs, le polynôme tordu $RP_0 = -SP_1$ est un plus petit multiple commun à gauche (p. p. c. m. g.) de P_0 et P_1, par le même argument que dans le cas commutatif, en suivant de près les degrés tout au long de l'algorithme.

On a vu que les algorithmes de clôture par addition entre fonctions D-finies ou entre suites P-récursives renvoient un multiple commun à gauche des polynômes tordus L_f et L_g décrivant les deux objets f et g additionner. Comme ces algorithmes opèrent par degrés croissants, le polynôme renvoyé est de degré minimal en ∂, parmi ceux qui annulent la somme $f+g$. Le polynôme annulateur de la somme $f+g$ renvoyé par ces algorithmes est donc le p. p. c. m. g. de L_f et de L_g.

Exemple 2. Nous repartons des polynômes A et B de l'exemple 1 pour en calculer un p. g. c. d. d. et un p. p. c. m. g. On pose $P_0 = A$, $P_1 = B$; on a déjà calculé $P_2 = -n^{-1}(n^2+n+1)\big(\partial_n - (n+1)\big)$ avec $P_0 = n^{-1}(n^2-1)P_1 + P_2$. Il vient ensuite

$$P_1 = \left(-\frac{n(n+1)}{n^2+3n+3}\partial_n + \frac{n(n+1)}{n^2+n+1} \right) P_2 + 0.$$

Ainsi, le p. g. c. d. d. unitaire est $\partial_n - (n+1)$. Remarquons qu'il annule les solutions communes de A et B, à savoir les multiples de $n!$. Le p. p. c. m. g unitaire s'obtient par renormalisation de $Q_1P_0 = (Q_1Q_0 + 1)P_1$:

$$\partial_n^3 - \frac{n^3+6n^2+8n+5}{n^2+n+1}\partial_n^2 + \frac{2n^3+9n^2+13n+7}{n^2+n+1}\partial_n - \frac{(n^2+3n+3)(n+1)}{n^2+n+1}.$$

Notons que ses solutions sont toutes les solutions de A et B : les combinaisons linéaires de $n!$, n et 1.

6. Relations de contiguïté

Un grand nombre de fonctions spéciales sont en fait des fonctions $f_n(x)$ d'une variable continue x et d'une variable discrète n. Les familles de telles fonctions dont l'intérêt a été relevé, par exemple par la physique mathématique, sont telles que la dépendance en x est liée à la dépendance en n. Il apparaît que très fréquemment, la fonction $f_{n+\alpha}(x)$, pour $\alpha = \pm 1$, est reliée à la fonction $f_n(x)$ et à ses dérivées. C'est le cas pour la classe importante des *fonctions hypergéométriques*, c'est-à-dire, essentiellement, pour les séries génératrices de suites hypergéométriques, et pour des fonctions limites de fonctions hypergéométriques, dont un certain nombre de familles de polynômes orthogonaux classiques, et pour des généralisations.

Dans cette section, nous considérons des fonctions D-finies paramétrées et résolvons algorithmiquement des problèmes tels que la détermination d'une relation de la forme

$$f_{n+1}(x) = \sum_{i=0}^{r-1} a_i(x) f_n^{(i)}(x)$$

pour la suite de polynômes

$$f_n(x) = \sum_{k=0}^{n} (-1)^k \binom{n}{k}^2 \binom{n+k}{k}^2 x^k.$$

Ici, la relation explicite est

$$f_{n+1}(x) = 12\frac{x^3(x+1)}{(n+1)^3} f_n'''(x) + 4\frac{x^2(2n+2xn+11+14x)}{(n+1)^3} f_n''(x) \\ - 4\frac{x(5xn^2-n^2-4n-6+2xn-9x)}{(n+1)^3} f_n'(x) - \frac{16xn-n-1+4x}{n+1} f_n(x).$$

L'opérateur différentiel linéaire implicitement au membre droit s'appelle un *opérateur de montée* ; un opérateur qui donnerait $f_{n-1}(x)$ s'appelle un *opérateur de descente*. Une relation linéaire entre les decalées $f_{n+i}(x)$ et ne faisant intervenir aucune dérivation s'appelle une *relation de contiguïté*.

6.1. Fonction hypergéométrique de Gauss. La plus simple des fonctions hypergéométriques est la fonction hypergéométrique de Gauss, définie par

$$F(a,b;c;x) = \sum_{k\geq 0} \frac{(a)_k (b)_k}{(c)_k} \frac{x^k}{k!} \qquad \text{avec} \qquad (s)_n = s(s+1)\cdots(s+n-1) = \frac{\Gamma(s+n)}{\Gamma(s)}.$$

(Ici, la fonction $\Gamma(s)$ est la fonction classique qui interpole la factorielle.)

Oublions la dépendance en b et c du coefficient de x^k dans cette somme, coefficient que nous notons $u_{a,k}$. Nous avons

$$u_{a,k+1} = \frac{(a+k)(b+k)}{(c+k)(k+1)} u_{a,k} \qquad \text{et} \qquad u_{a+1,k} = \left(\frac{k}{a}+1\right) u_{a,k}.$$

La première de ces identités nous fournit le polynôme tordu

$$(c+k)(k+1)\partial_k - (a+k)(b+k)$$

qui annule u. Par le morphisme qui effectue le passage à la série génératrice, nous obtenons

$$(c+x\partial_x)(x\partial_x+1)x^{-1} - (a+x\partial_x)(b+x\partial_x) \\ = \big((x\partial_x)^2 + (c+1)x\partial_x + c\big)x^{-1} - \big((x\partial_x)^2 + (a+b)x\partial_x + ab\big) \\ = x(1-x)\partial_x^2 + \big(c-(a+b+1)x\big)\partial_x - ab.$$

Notons L ce polynôme tordu en ∂_x. La deuxième identité sur u donne, après sommation, $F(a+1,b;c;x) = (a^{-1}x\partial_x + 1)\cdot F(a,b;c;x)$, c'est-à-dire qu'un opérateur de montée est donné par le polynôme tordu $L_\uparrow = a^{-1}x\partial_x + 1$.

Définissons $G(a,b;c;x)$ comme étant $F(a+1,b;c;x)$. Supposons qu'il existe un inverse V de $L_\uparrow$ modulo L à droite. Alors $VL_\uparrow - 1$ est un multiple à gauche de L, qui annule donc F. Ainsi, $F = VL_\uparrow \cdot F = V\cdot G$, autrement dit, V représente un opérateur de descente de G. On obtient un opérateur de descente pour F par un simple décalage arrière de a dans V. La division euclidienne

$$L = Q(a^{-1}\partial_x + 1) - (c-a-1)ax^{-1} \qquad \text{où} \qquad Q = a(1-x)\partial_x + (c-a-1)ax^{-1} - ab$$

donne l'opérateur de descente après avoir décalé a dans $(c-a-1)^{-1}a^{-1}xQ$ par

$$L_\downarrow = \frac{x(1-x)}{a-c}\partial_x - \frac{bx}{a-c} - 1.$$

Notre objectif est maintenant de calculer une relation de contiguïté pour F. Nous avons obtenu $L_\uparrow(a) \cdot F = \partial_a \cdot F$, où nous avons noté explicitement la dépendance en a du polynôme tordu $L_\uparrow$. Il s'ensuit la relation

$$\partial_a^i \cdot F = L_{\uparrow,i}(a) \cdot F \qquad \text{où} \qquad L_\uparrow(a)L_\uparrow(a+1)\cdots L_\uparrow(a+i-1) \cdot F,$$

dans laquelle nous pouvons, comme toujours, remplacer un polynôme agissant sur la fonction F par le reste de la division euclidienne de ce polynôme par L, qui annule F. Ainsi, une relation de contiguïté s'obtient en recherchant une combinaison linéaire, à coefficients dans $\mathbb{Q}(a,b,c,x)$ des restes modulo L à droite des $L_{\uparrow,i}(a)$ pour $i = 0, 1, 2$.

EXERCICE 5. Terminer ce calcul pour retrouver :

$$(a+1)(1-x)F(a+2,b;c;x) + \big(c - xb + (a+1)(x-2)\big)F(a+1,b;c;x) \\ + (a-c+1)F(a,b;c;x) = 0.$$

6.2. Extension aux séries partiellement hypergéométriques. Nous considérons maintenant des sommes

$$f_n(x) = \sum_{k\geq 0} u_{n,k} x^k$$

dans lesquelles u n'est hypergéométrique qu'en n, mais est seulement P-récursive en k, et satisfait à la relation

$$a_p(n,k)u_{n,k+p} + \cdots + a_0(n,k)u_{n,k} = 0.$$

Comme dans le cas doublement hypergéométrique, cette relation fournit une relation purement différentielle sur f. Comme précédemment, aussi, la relation de récurrence du premier order en n sur u donne une expression de $f_{n+1}(x)$ comme combinaison linéaire de dérivées. On procède donc comme dans la section précédente pour calculer opérateurs de montée, de descente, et relations de contiguïté.

EXERCICE 6 (Assez calculatoire). Calculer l'opérateur de montée annoncé dans l'introduction de cette section.

Bibliographie

[1] Bronstein (M.) and Petkovšek (M.). – On Ore rings, linear operators and factorisation. *Programmirovanie*, vol. 1, 1994, pp. 27–44. – Also available as Research Report 200, Informatik, ETH Zürich.

[2] Bronstein (Manuel) and Petkovšek (Marko). – An introduction to pseudo-linear algebra. *Theoretical Computer Science*, vol. 157, 1996, pp. 3–33.

[3] Chyzak (Frédéric) and Salvy (Bruno). – Non-commutative elimination in Ore algebras proves multivariate holonomic identities. *Journal of Symbolic Computation*, vol. 26, n° 2, August 1998, pp. 187–227.

[4] Ore (Oystein). – Linear equations in non-commutative fields. *Annals of Mathematics*, vol. 32, 1931, pp. 463–477.

[5] Ore (Oystein). – Theory of non-commutative polynomials. *Annals of Mathematics*, vol. 34, 1933, pp. 480–508.

[6] Takayama (Nobuki). – Gröbner basis and the problem of contiguous relations. *Japan Journal of Applied Mathematics*, vol. 6, n° 1, 1989, pp. 147–160.

COURS 24

Algorithmes pour les fonctions spéciales dans les algèbres de Ore

Notes de Frédéric Chyzak

1. Algèbres de Ore rationnelles

Une généralisation à plusieurs dérivations et décalages de la notion d'anneau de polynômes tordus du chapitre précédent est donnée par la définition qui suit.

Définition (Algèbre de Ore). *Étant donnés*
- *un corps $k(x) = k(x_1, \dots, x_r)$ de fractions rationnelles,*
- *r morphismes σ_i de ce corps commutant deux à deux,*
- *pour chaque i une σ-dérivation δ_i relative à σ_i, c'est-à-dire pour chaque i un endomorphisme linéaire pour lequel $\delta_i(ab) = \sigma_i(a)\delta_i(b) + \delta_i(a)b$ dès que a et b sont dans $k(x)$, toutes ces σ-dérivations commutant deux à deux et δ_i commutant avec σ_j chaque fois que $i \neq j$,*
- *r indéterminées ∂_i,*

l'algèbre de Ore (rationnelle) notée $k(x_1, \dots, x_r)\langle \partial_1, \dots, \partial_r; \sigma_1, \dots, \sigma_r, \delta_1, \dots, \delta_r\rangle$ ou plus simplement $k(x)\langle \partial; \sigma, \delta\rangle$ est la $k(x)$-algèbre associative engendrée par les ∂_i modulo les relations

$$\partial_i a = \sigma_i(a)\partial_i + \delta_i(a), \qquad \partial_i\partial_j = \partial_j\partial_i,$$

quand a est dans $k(x)$. On note plus simplement $k(x_1, \dots, x_r)\langle \partial_1, \dots, \partial_r; \sigma_1, \dots, \sigma_r\rangle$ ou encore $k(x)\langle \partial; \sigma\rangle$ le cas où tous les δ_i sont nuls.

Donnons un exemple : en notant x pour m_x et n pour m_n, on vérifie l'existence d'une algèbre de Ore $A = \mathbb{C}(n, x)\langle \partial_n, \partial_x; S_n, I, 0, D_x\rangle$ avec $S_n(n) = n + 1$, $S_n(x) = x$, $D_x(n) = 0$, $D_x(x) = 1$, et plus généralement $S_n(a) = a(n + 1, x)$ et $D_x(a) = da/dx$ quand $a = a(n, x) \in \mathbb{C}(n, x)$.

2. Idéal annulateur

Les éléments des algèbres de Ore représentent des opérateurs linéaires, différentiels, de récurrence, ou autres, et agissent donc sur des fonctions, suites, suites de fonctions, etc. Donnons un exemple qui montre que ces objets sont une bonne représentation polynomiale des opérateurs linéaires, l'exemple de la famille des polynômes orthogonaux de Laguerre qui va nous servir pour toute la suite du chapitre.

Pour chaque paramètre strictement positif α, l'intégrale

$$\langle f, g\rangle = \int_0^\infty f(x)g(x)x^\alpha e^{-x}\,dx$$

définit un produit scalaire sur les fonctions polynomiales réelles. Par la théorie, on déduit l'existence de bases orthogonales échelonnées en degré. On a par exemple la base des polynômes orthogonaux de Laguerre, donnée par

$$L_n^{(\alpha)}(x) = \frac{1}{n!} x^{-\alpha} e^x \left(\frac{d}{dx}\right)^n \left(e^{-x} x^{n+\alpha}\right) = \frac{1}{n!} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (\alpha+k+1)\cdots(\alpha+n) x^k.$$

On vérifie que ces polynômes vérifient les relations (linéaires)

$$\begin{aligned}(n+2)L_{n+2}^{(\alpha)} - (2n+\alpha+3-x)L_{n+1}^{(\alpha)} + (n+\alpha+1)L_n^{(\alpha)} &= 0,\\ xL_n^{(\alpha)\prime} - (n+1)L_{n+1}^{(\alpha)} + (n+\alpha+1-x)L_n^{(\alpha)} &= 0,\\ xL_n^{(\alpha)\prime\prime} + (\alpha+1-x)L_n^{(\alpha)\prime} + nL_n^{(\alpha)} &= 0,\end{aligned}$$

avec les conditions initiales $L_0^{(\alpha)} = 1$ et $L_1^{(\alpha)} = \alpha + 1 - x$. Dans l'algèbre de Ore A ci-dessus, ces équations se recodent en les polynômes tordus suivant, qui annulent la suite de fonctions polynomiales $L^{(\alpha)}$:

$$\begin{aligned}p_1 &= (n+2)\partial_n^2 - (2n+\alpha+3-x)\partial_n + (n+\alpha+1),\\ p_2 &= x\partial_x - (n+1)\partial_n + (n+\alpha+1-x),\\ p_3 &= x\partial_x^2 + (\alpha+1-x)\partial_x + n.\end{aligned}$$

Ces trois polynômes engendrent un idéal à gauche dans A, l'idéal annulateur de $L^{(\alpha)}$ dans A. Pour mémoire, un idéal à gauche I d'un anneau R est un sous-ensemble non vide de R stable par addition et par multiplication à gauche par tout élément de R. Cette stabilité reflète le fait que l'addition terme à terme de deux relations linéaires vérifiées par $L^{(\alpha)}$ est une nouvelle relation linéaire vérifiée par $L^{(\alpha)}$, de même qu'en appliquant un opérateur linéaire sur une relation linéaire vérifiée par $L^{(\alpha)}$, on retrouve une relation linéaire vérifiée par $L^{(\alpha)}$.

Toute autre famille de polynômes orthogonaux classique se traiterait de la même manière et aurait pu servir de support au cours. La même nature de système linéaire avec une dérivation sur une variable et un décalage sur une autre permet de traiter de la même façon nombre de famille de fonctions spéciales paramétrées, telles les fonctions de Bessel, de Hankel, etc.

3. Bases de Gröbner pour les idéaux à gauche

La propriété essentielle qui fait fonctionner toute la théorie des bases de Gröbner et l'algorithme de Buchberger dans le cadre de polynômes commutatifs est que le monôme de tête d'un produit de polynômes est le produit des monômes de tête des termes du produit. Cette propriété reste vérifiée sur des polynômes non-commutatifs sujets aux relations de définition des algèbres de Ore rationnelles, dès lors qu'on considère des ordres monomiaux sur les ∂_i. En refaisant la théorie en s'efforçant de faire toutes les combinaisons linéaires avec des facteurs à gauche, on obtient le résultat suivant (*cf.* le cours sur les bases de Gröbner classiques) :

Théorème. *Soit A une algèbre de Ore rationnelle.*

(i) *Tout idéal à gauche I de A admet pour chaque ordre monomial (admissible) sur les ∂_i une unique base de Gröbner minimale réduite G, au sens où l'une quelconque des propriétés équivalentes suivantes est vérifiée :*

1. *la partie stable du monoïde des monômes en les ∂_i engendrée par les monômes de tête des éléments de G est égale celle engendrée par ceux de I ;*
2. *tout f non nul de I est réductible par G ;*
3. *pour tout f dans A, il existe un unique r dans A dont aucun monôme ne soit divisible par un monôme de tête d'un élément de G et tel que $f-r$ soit dans l'idéal I ;*
4. *pour tout f dans I, le reste de la division (à droite) de f par G est nul.*

(ii) *Soit $P = \{p_k\}_{1\leq k\leq r}$ un système de générateurs non nuls d'un idéal à gauche I de A. Tous les S-polynômes* $\mathrm{Spoly}(p_i, p_j)$ *(définis par des combinaisons linéaires à gauche) se réduisent à 0 par P si et seulement si P est une base de Gröbner de l'idéal.*

(iii) *Une variante de l'algorithme de Buchberger termine et renvoie une base de Gröbner de tout idéal à gauche I de A.*

Plutôt que de faire la théorie, montrons le calcul sur un exemple.

En repartant des polynômes p_1 et p_2 qui annulent la suite des polynômes orthogonaux de Laguerre, montrons que le polynôme p_3 s'obtient par élimination de S par un calcul pour l'ordre $\mathsf{lex}(\partial_n, \partial_x)$. Pour cet ordre, le terme de tête de p_1 est $(n+2)\times\partial_n^2$, celui de p_2 est $-(n+1)\times\partial_n$. On calcule donc d'abord le S-polynôme de p_1 et de p_2, sous la forme

$$\mathrm{Spoly}(p_1, p_2) = p_1 + \partial_n p_2 = x\partial_x\partial_n - (n+1)\partial_n + (n+\alpha+1).$$

Ce polynôme a $\partial_x\partial_n$ pour monôme de tête et est réductible par p_2 ; après multiplication par $(n+1)$ et ajout de $x\partial_x p_2$, on obtient

$$x^2\partial_x^2 + (n+\alpha+2-x)x\partial_x - (n+1)^2\partial_n + (n+1)(n+\alpha+1) - x.$$

Ce polynôme a ∂_n pour monôme de tête et est réductible par p_2 ; après retranchement de $(n+1)p_2$, on aboutit à

$$x^2\partial_x^2 + (\alpha+1-x)x\partial_x + nx,$$

qui n'est autre que xp_3. En poursuivant, on montre que les S-polynômes de p_1 et p_2 avec p_3 se réduisent à 0 ; puisque le monôme de tête de p_2, ∂_n, divise celui de p_1, ∂_n^2, une base de Gröbner minimale pour l'ordre $\mathsf{lex}(\partial_n, \partial_x)$ est $\{p_2, p_3\}$.

De façon analogue, une base de Gröbner pour l'ordre $\mathsf{lex}(\partial_x, \partial_n)$ de l'idéal engendré par p_2 et p_3 est $\{p_1, p_2\}$. Les bases de Gröbner permettent de déterminer la redondance du système $\{p_1, p_2, p_3\}$.

Exercice 1. Calculer une base de Gröbner pour l'ordre $\mathsf{lex}(\partial_x, \partial_n)$ de l'idéal engendré par p_2 et p_3 et vérifier le point ci-dessus.

Les polynômes p_1, p_2, p_3 qui annulent la suite des polynômes orthogonaux de Laguerre sont encore plus contraints qu'il n'y paraît jusqu'à présent : p_2 se déduit en fait de p_1. En effet, ne connaissant que p_1, on peut rechercher le polynôme p_2 sous la forme indéterminée

$$p_2 = \partial_x - u(n,x)\partial_n - v(n,x),$$

pour des fractions rationnelles à déterminer u et v, et faire l'hypothèse heuristique que $\{p_1, p_2\}$ est une base de Gröbner pour l'ordre $\mathsf{lex}(\partial_x, \partial_n)$. (Cette hypothèse heuristique est en fait naturelle dès qu'on sait qu'on a affaire à une famille de polynômes orthogonaux.)

EXERCICE 2 (Presque un problème). Utiliser la théorie des bases de Gröbner pour donner un système de récurrence linéaires sur u et v qui, après résolution, redonne le polynôme p_2. (Pour la résolution, on se souviendra des conditions initiales $L_0^{(\alpha)} = 1$ et $L_1^{(\alpha)} = \alpha + 1 - x$.)

4. Module quotient et dimension de l'espace des solutions

Dans le cas commutatif, le quotient l'une algèbre A de polynômes par l'un de ses idéaux (bilatères) I reste munie d'un produit canonique et est donc une algèbre. Cette propriété n'est plus réalisée dans le cas d'une algèbre de Ore $A = k(x)\langle\partial; \sigma, \delta\rangle$, mais en voyant A comme un idéal à gauche, trivial, de lui-même, le quotient A/I conserve une addition canonique, ainsi que la stabilité par multiplication à gauche par tout élément de A, ce qui fait de ce quotient un module à gauche sur A. Ce module est de plus un espace vectoriel sur $k(x)$.

Dans le cas commutatif, un cadre particulier important est celui d'un quotient de dimension finie comme espace vectoriel, car il représente une famille finie de points solutions. Le cas d'un quotient d'une algèbre de Ore qui est un espace vectoriel sur $k(x)$ de dimension finie est lui-aussi important ; dans l'interprétation en opérateurs linéaires, il correspond en règle générale à un espace vectoriel de solutions de dimension finie sur k.

Dans la fin de cette section, nous allons quelque peu détailler ce lien dans le cas d'opérateurs différentiels ; d'autres cadres fournissent le même genre de résultats. Nous irons plus loin sur le sujet dans la section sur les fonctions ∂-finies.

4.1. Séries formelles solutions en un point régulier dans le cas différentiel. Considérons une algèbre de Ore

$$A = \mathbb{C}(x_1, \ldots, x_r)\langle\partial_1, \ldots, \partial_r; I, \ldots, I, D_{x_1}, \ldots, D_{x_r}\rangle,$$

un idéal à gauche I de cette algèbre, donné par un système différentiel linéaire. Nous voulons décrire les solutions séries annulées par tous les éléments de I, où une série est ici un élément de $\mathbb{C}[[x_1, \ldots, x_r]]$, c'est-à-dire une combinaison linéaire formelle éventuellement infinie de monômes à exposants entiers positifs. Dans cette objectif, cette section ébauche un analogue en plusieurs variables de la conversion entre équation différentielle décrivant une fonction D-finie d'une variable et équation de récurrence vérifiée par la suite P-récursive des coefficients.

Fixons un ordre monomial sur les monômes en les ∂_i, puis, pour cet ordre, une base de Gröbner B de I, donnée par des éléments de A sans fractions, c'est-à-dire avec des coefficients polynomiaux. Cette base de Gröbner B fournit un escalier ; notons S l'ensemble des multi-exposants $s = (s_1, \ldots, s_r)$ des monômes $\partial^s = \partial_1^{s_1} \cdots \partial_r^{s_r}$ sous l'escalier, c'est-à-dire des monômes qui ne sont pas réductibles par B. Le module quotient A/I a alors une base d'espace vectoriel sur $\mathbb{C}(x)$ constituée des $\partial^s + I$, les classes des ∂_s modulo I, pour s décrivant S. Soit u le polynôme produit des coefficients de tête des éléments de B et faisons l'hypothèse que u ne s'annule pas pour $x_1 = \cdots = x_r = 0$. Nous affirmons qu'alors, l'idéal I admet un espace vectoriel de solutions séries dans $\mathbb{C}[[x_1, \ldots, x_r]]$ de dimension le cardinal de S, c'est-à-dire la dimension sur $\mathbb{C}(x)$ de A/I vu comme espace vectoriel. On dit dans ce cas que le point $(0, \ldots, 0)$ est régulier pour le système différentiel linéaire définissant l'idéal I.

En effet, pour tout multi-exposant $n = (n_1, \ldots, n_r)$, la réduction du monôme ∂^n par B fournit une combinaison linéaire $\sum_{s \in S} v_{n,s}\partial^s$ congrue à ∂^n modulo I. Notons que par construction, les coefficients $v_{n,s}$ sont éléments de $\mathbb{C}[x_1, \ldots, x_r, u^{-1}]$ et ont

ainsi une évaluation bien définie en $x_1 = \cdots = x_r = 0$. Maintenant, puisque chaque élément de I s'annule sur toute solution série

$$\phi = \sum_{n_1 \in \mathbb{N}, \ldots, n_r \in \mathbb{N}} c_{n_1,\ldots,n_r} x_1^{n_1} \cdots x_r^{n_r}$$

de I, le monôme ∂^n et la somme $\sum_{s\in S} v_{n,s}\partial^s$ ont la même action sur ϕ. Après évaluation en $x_1 = \cdots = x_r = 0$, on a

$$n_1! \ldots n_r!\, c_{n_1,\ldots,n_r} = \sum_{s\in S} v_{n,s}(0,\ldots,0) s_1! \ldots s_r!\, c_{s_1,\ldots,s_r}.$$

Autrement dit, la série ϕ est totalement déterminée par ses quelques premiers coefficients c_s pour $s \in S$, en nombre donné par la dimension de A/I.

Illustrons cette idée en reprenant l'exemple des polynômes orthogonaux de Laguerre, qui étendent déjà légèrement le cadre purement différentiel qui précède. Posons

$$L_n^{(\alpha)}(x) = \sum_{k=0}^{n} \ell_{n,k} x^k.$$

En multipliant chaque p_i pour $i = 1, 2, 3$ par ∂_x^k, il vient

$$\begin{aligned}
\partial_x^k p_1 &= (n+2)\partial_n^2 \partial_x^k - (2n+\alpha+3-x)\partial_n \partial_x^k + k\partial_n \partial_x^{k-1} + (n+\alpha+1)\partial_x^k,\\
\partial_x^k p_2 &= x\partial_x^{k+1} - (n+1)\partial_n \partial_x^k + (n+\alpha+k+1-x)\partial_x^k - k\partial_x^{k-1},\\
\partial_x^k p_3 &= x\partial_x^{k+2} + (\alpha+k+1-x)\partial_x^{k+1} + (n-k)\partial_x^k.
\end{aligned}$$

Après application sur $L^{(\alpha)}$, évaluation en $x = 0$ et division par $k!$, on trouve les relations de récurrence sur la famille doublement indexée des $\ell_{n,k}$

$$\begin{aligned}
(n+2)\ell_{n+2,k} - (2n+\alpha+3)\ell_{n+1,k} + \ell_{n+1,k-1} + (n+\alpha+1)\ell_{n,k} &= 0,\\
-(n+1)\ell_{n+1,k} + (n+\alpha+k+1)\ell_{n,k} - \ell_{n,k-1} &= 0,\\
(k+1)(\alpha+k+1)\ell_{n,k+1} + (n-k)\ell_{n,k} &= 0.
\end{aligned}$$

En décalant la dernière vers l'arrière en k puis éliminant $\ell_{n,k-1}$ entre la relation obtenue et la deuxième récurrence ci-dessus, on obtient la récurrence

$$(n+1-k)\ell_{n+1,k} - (n+\alpha+1)\ell_{n,k} = 0.$$

Ce jeu de récurrences fournit tous les $\ell_{n,k}$.

4.2. Exemple : Les solutions en séries des systèmes hypergéométriques de Gel'fand, Kapranov et Zelevinsky. Prenons un exemple concret, celui des systèmes hypergéométriques dans la formulation de Gel'fand, Kapranov et Zelevinsky. L'algèbre de Ore qui intervient dans cet exemple est l'algèbre A engendrée par quatre indéterminées ∂_1, …, ∂_4 sur le corps $\mathbb{C}(x_1, \ldots, x_4)$, chaque ∂_i représentant l'opérateur de dérivation par rapport à x_i. Le système GKZ est le système

$$\begin{aligned}
p_1 &= \partial_2\partial_3 - \partial_1\partial_4,\\
p_2 &= x_1\partial_1 - x_4\partial_4 + (1-c),\\
p_3 &= x_2\partial_2 + x_4\partial_4 + a,\\
p_4 &= x_3\partial_3 + x_4\partial_4 + b,
\end{aligned}$$

pour des paramètres complexes a, b et c. L'objectif de l'exemple est de montrer que ce système admet un espace vectoriel de solutions formelles de dimension exactement 2, où par solution formelle nous entendons plus maintenant généralement une série de la forme

$$x_1^{a_1}\dots x_4^{a_4}\sum_{n_1\in\mathbb{Z},\dots,n_4\in\mathbb{Z}} c_{n_1,\dots,n_4}x_1^{n_1}\cdots x_4^{n_4},$$

pour des a_i et des coefficients complexes, ou une combinaison linéaire de telles séries. (Il y a bien un espace vectoriel sur $\mathbb{C}$ où vivent ces séries, mais pas de produit sur ces séries. En revanche, toute séries peut être multipliée par un polynôme en les x_i et leurs inverses x_i^{-1}, ainsi que dérivée formellement par rapport à chacune des indéterminées, tout en restant dans l'espace vectoriel.)

Soit I l'idéal engendré par le système $\{p_1,\dots,p_4\}$ et calculons à partir de ce système une base de Gröbner de I pour l'ordre $\mathsf{lex}(\partial_1,\dots,\partial_4)$. Les monômes de tête respectifs de p_1 et p_2 sont $\partial_1\partial_4$ et ∂_1. Le S-polynôme de p_1 et de p_2 est donc

$$\mathrm{Spoly}(p_1,p_2)=x_1p_1+\partial_4p_2=x_1\partial_2\partial_3-x_4\partial_4^2-c\partial_4,$$

dont le monôme de tête est $\partial_2\partial_3$; il est donc réductible par p_3. Après multiplication par $-x_2$ et ajout de $x_1\partial_3p_3$, on obtient

$$x_1x_4\partial_3\partial_4+ax_1\partial_3+x_2x_4\partial_4^2+cx_2\partial_4.$$

Ce polynôme a $\partial_3\partial_4$ pour monôme de tête et est donc réductible par p_4. Après multiplication par x_3 et retranchement de $x_1x_4\partial_4p_4$, on aboutit à

$$ax_1x_3\partial_3+(x_2x_3-x_1x_4)x_4\partial_4^2+\big(cx_2x_3-(b+1)x_1x_4\big)\partial_4,$$

qui est encore réductible par p_4. Après retranchement de ax_1p_4, on a finalement un polynôme qui n'est pas réductible par $\{p_1,\dots,p_4\}$, à savoir

$$p_5=(x_2x_3-x_1x_4)x_4\partial_4^2+\big(cx_2x_3-(a+b+1)x_1x_4\big)\partial_4-abx_1.$$

Par ailleurs, les S-polynômes entre les polynômes p_2, p_3 et p_4 pris deux à deux sont tous nuls, comme on le vérifie en observant que les $x_i\partial_i$ commutent deux à deux. En poursuivant les calculs sur les S-polynômes $\mathrm{Spoly}(p_i,p_5)$, on montre que tous ces derniers se réduisent à 0 par $\{p_1,\dots,p_5\}$. On obtient ainsi qu'un base de Gröbner minimale est $\{p_2,p_3,p_4,p_5\}$, avec les monômes dominants respectifs ∂_1, ∂_2, ∂_3 et ∂_4^2.

Le module quotient A/I a donc une base d'espace vectoriel sur $\mathbb{C}(x_1,\dots,x_4)$ constituée de $1+I$ et ∂_4+I, les classes respectives de 1 et ∂_4 modulo I. La structure de module est donnée explicitement par l'action des ∂_i sur ces deux éléments de base.

Exercice 3. Donner l'expression explicite de cette action en récrivant chaque $\partial_i\cdot(1+I)$ et chaque $\partial_i\cdot(\partial_4+I)$ sur la base $(1+I,\partial_4+I)$.

Revenons sur les solutions séries du système GKZ. Le polynôme p_2 agit sur un monôme par

$$p_2\cdot(x_1^{\lambda_1}\cdots x_4^{\lambda_4})=(\lambda_1-\lambda_4+1-c)x_1^{\lambda_1}\cdots x_4^{\lambda_4}.$$

(Notons la distinction entre le produit dans A noté $p_2x_1^{\lambda_1}\cdots x_4^{\lambda_4}$ et l'opération de p_2, ici sur une serie h en x, notée $p_2\cdot h$; on comparera par exemple $\partial_1x_1^5=x_1^5\partial_1+5x_1^4$ et $\partial_1\cdot x_1^5=5x_1^4$.) Ainsi, un monôme $x_1^{\lambda_1}\cdots x_4^{\lambda_4}$ ne peut apparaître avec un coefficient non nul dans une série ϕ solution du système GKZ que si $\lambda_1-\lambda_4+1-c$ est nul. En poursuivant ce type de raisonnement avec p_3 et p_4, on obtient de même les

contraintes $\lambda_2 + \lambda_4 + a = 0$ et $\lambda_3 + \lambda_4 + a = 0$ et on aboutit à ce que les seuls monômes pouvant apparaître avec un coefficient non nul sont de la forme

$$x_1^{\lambda_4+c-1} x_2^{-\lambda_4-a} x_3^{-\lambda_4-b} x_4^{\lambda_4} = \frac{x_1^{c-1}}{x_2^a x_3^b} \left(\frac{x_1 x_4}{x_2 x_3}\right)^{\lambda_4},$$

et une solution ϕ est nécessairement de la forme

$$\phi = \frac{x_1^{c-1}}{x_2^a x_3^b} f\left(\frac{x_1 x_4}{x_2 x_3}\right),$$

pour une série formelle f en y à exposants entiers relatifs. Reste à exploiter que ϕ est solution de p_1, ou de manière équivalente puisque sous la forme ci-dessus ϕ est déjà solution de p_2, p_3 et p_4, de p_5. Après avoir évalué en $y = x_1x_4/x_2x_3$, on a

$$0 = \frac{x_2^a x_3^b}{x_1^{c-2}} p_5 \cdot \phi = (1-y)yf''(y) + \big(c - (a+b+1)y\big) f'(y) - abf(y).$$

On reconnaît là l'équation hypergéométrique de Gauss, annulée par la série de Gauss

$$f_1 = {}_2\mathrm{F}_1(a, b; c; y) = \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{y^n}{n!}$$

où $(x)_n$ représente le symbole de Pochhammer, $x(x+1)\cdots(x+n-1)$. On vérifie qu'une solution formelle linéairement indépendante avec f_1 est $f_2 = y^{1-c}\,{}_2\mathrm{F}_1(a - c + 1, b - c + 1; 2 - c; y)$. On a ainsi obtenu deux solutions formelles linéairement indépendantes du système GKZ, $\phi_i = x_1^{c-1}/x_2^a x_3^b \times f_i(x_1x_4/x_2x_3)$ pour $i = 1$ et $i = 2$.

Pour tout système différentiel représenté par un idéal I de A, un résultat d'analyse, le théorème de Cauchy–Kovalevskaya, affirme l'existence, au voisinage de tout point en dehors d'une certaine variété singulière, d'un $\mathbb{C}$-espace vectoriel de solutions analytiques de dimension celle sur $\mathbb{C}(x)$ de l'espace vectoriel A/I. Or, on montre que cette variété singulière est incluse dans le lieu des zéros du produit des coefficients polynomiaux de tête d'une base de Gröbner de I écrite sans fractions.

Dans le cas de notre exemple, la dimension de A/I est 2 et la variété singulière est incluse dans le lieu des zéros de $x_1 \cdots x_4(x_2x_3 - x_1x_4)$. Hors de ce lieu, y n'est ni nul, ni infini, ni égal à 1, et les fonctions ϕ_1 et ϕ_2 sont donc analytiques puisque, moyennant quelques hypothèses sur les paramètres a, b et c, les deux séries solutions de l'équation de Gauss, f_1 et f_2, représentent des fonctions analytiques sur $\mathbb{C} \setminus \{0, 1\}$. On a donc trouvé un espace de solutions analytiques de dimension 2, et par le théorème de Cauchy–Kovalevskaya, toutes les solutions analytiques du système GKZ en dehors de sa variété singulière.

5. Les fonctions ∂-finies et leurs clôtures

Nous poursuivons maintenant avec le cas particulier important des système fonctionnels linéaires correspondant à des modules A/I de dimension finie sur $\mathbb{C}(x)$. L'objectif est ici de montrer que pour une algèbre A donnée, leurs solutions, que nous appelerons fonctions ∂-finies, forment une algèbre sur $\mathbb{C}$. Nous allons donner un algorithme pour calculer les clôtures correspondantes. Voici tout de suite la définition, déjà motivée par les sections et cours précédents.

DÉFINITION (Fonction ∂-finie). *Étant donnée une algèbre de Ore (rationnelle)*

$$A = \mathbb{C}(x_1, \ldots, x_r)\langle \partial_1, \ldots, \partial_r; \sigma_1, \ldots, \sigma_r, \delta_1, \ldots, \delta_r \rangle$$

agissant sur un $\mathbb{C}(x_1, \ldots, x_r)$-espace vectoriel V, un élément f de V est dit ∂-fini lorsque l'une des conditions équivalentes suivantes est vérifiée :

1. *pour chaque i entre 1 et r, il existe un polynôme $P_i = P_i(x_1, \ldots, x_r, \partial_i)$ dont l'action annule f ;*
2. *la famille des $\partial^a \cdot f$, où ∂^a décrit les monômes de A, engendre un espace vectoriel de dimension finie sur $\mathbb{C}(x)$;*
3. *le module quotient A/I où I note l'idéal annulateur de f pour l'action de A est un espace vectoriel de dimension finie sur $\mathbb{C}(x)$.*

Par commodité, nous appellerons fonctions les éléments de l'espace vectoriel V, ce quand bien même il ne s'agirait pas de fonctions de l'analyse, mais afin d'éviter la terminologie plus lourde et moins imagée d'éléments ∂-finis d'un module sur A.

EXERCICE 4. Vérifier l'équivalence entre les trois points de la définition ci-dessus.

5.1. Méthode du vecteur cyclique. L'algorithme envisagé pour les clôtures des fonctions ∂-finies s'appuie le calcul de vecteur cyclique. Rappelons-en l'idée. Classiquement, étant donné un espace vectoriel V sur un corps k, sur lequel on suppose donnée une action de $k[X]$, un vecteur $v \in V$ est dit *cyclique* si la famille $\{X^i \cdot v\}$ engendre V comme k-espace vectoriel. Alors, v engendre V comme $k[X]$-module. Pour calculer, on suppose que V est de dimension finie d et que l'action de X est donnée sur une base $B = (b_1, \ldots, b_d)$ de V par une matrice M telle que $X \cdot v = (a_1, \ldots, a_d)M\,{}^tB$ pour tout vecteur $v = a_1b_1 + \cdots + a_db_d$. Pour tester si v est cyclique et le cas échéant rendre explicite la structure de module de V, on range dans une matrice les lignes $(a_1, \ldots, a_d)M^i$ pour $0 \leq i \leq m$ avec m à déterminer et on cherche par l'algorithme de Gauss une dépendance linéaire entre les lignes de la matrice obtenue. En procédant avec des $m \geq 0$ successifs, la première dépendance linéaire fournit le polynôme minimal de v sous l'action de X sur V ; son degré m vérifie $m \leq d$.

Ce calcul s'étend d'abord au cadre commutatif de l'action d'une algèbre $k[X] = k[X_1, \ldots, X_r]$ de polynômes en plusieurs indéterminées. Chaque X_i correspond alors à une matrice M_i et la commutativité des X_i dans $k[X]$ induit la commutativité entre les matrices M_i. Au lieu d'itérer sur les monômes X^i par ordre croissant de i, on itère maintenant sur les monômes $X^a = X_1^{a_1} \cdots X_r^{a_r}$ selon tout ordre qui assure qu'un monôme n'est considéré qu'après tous ses diviseurs. Soit $a(0)$, $a(1)$, etc, l'ordre dans lequel les multi-exposants des monômes sont énumérés. À chaque étape, on recherche une dépendance linéaire entre des vecteurs

$$X^{a(0)} \cdot v, \ldots, X^{a(m)} \cdot v.$$

En cas d'échec, on conserve ces $m + 1$ vecteurs et on reprend la recherche après avoir ajouté le nouveau vecteur $X^{a(m+1)} \cdot v$; en cas de succès, on retire le dernier vecteur introduit, $X^{a(m)} \cdot v$, on évite par la suite tous les multiples de ce monôme, et on introduit le nouveau vecteur $X^{a(m+1)} \cdot v$ pour reprendre la recherche sur la famille

$$X^{a(0)} \cdot v, \ldots, X^{a(m-1)} \cdot v, X^{a(m+1)} \cdot v.$$

Ce calcul termine si et seulement si le quotient $k[X]/I$, vu comme k-espace vectoriel, est de dimension finie. Chaque dépendance linéaire calculée fournit un polynôme P tel que $P(X_1, \ldots, X_r) \cdot v = 0$. Lorque l'itération sur les monômes X^a suit l'ordre croissant selon un ordre monomial, l'ensemble des polynômes annulateurs obtenus constitue une base de Gröbner de I pour l'ordre choisi.

5.2. Algorithmes de clôture des fonctions ∂-finies. Le procédé du vecteur cyclique s'étend au cas de l'action d'une algèbre de Ore (rationnelle) en présence de fonctions ∂-finies. Pour une algèbre de Ore

$$A = \mathbb{C}(x)\langle \partial_1, \ldots, \partial_r; \sigma_1, \ldots, \sigma_r, \delta_1, \ldots, \delta_r \rangle,$$

l'espace vectoriel utilisé est un module du type A/I, vu comme espace vectoriel sur $\mathbb{C}(x)$, ou plutôt un module obtenu à partir de construction sur des modules de la forme A/I, comme on va le voir sur l'exemple plus bas. L'espace V étant d'une certaine dimension finie d et une base $B = (b_1, \ldots, b_d)$ de V étant fixée, l'action de chaque ∂_i sur un vecteur $v = a_1 b_1 + \cdots + a_d b_d$ est donnée par une matrice M_i sous la forme

$$\partial_i \cdot v = \big(\sigma_i(a_1, \ldots, a_d) M_i + \delta_i(a_1, \ldots, a_d)\big)\, {}^tB,$$

en adoptant une notation selon laquelle les σ_i et δ_i agissent distributivement sur les entrées de vecteurs ou de matrices. Pour le choix particulier $v = b_\ell$, on observe que la ℓ-ième ligne de M_i n'est autre que le vecteur ligne des composantes de $\partial_i \cdot b_\ell$ sur la base B.

En faisant maintenant agir ∂_j et en posant $a = (a_1, \ldots, a_d)$, on a

$$\partial_j \partial_i \cdot v = \big(\sigma_j\sigma_i(a)\sigma_j(M_i)M_j + \sigma_j\delta_i(a)M_j + \sigma_j\sigma_i(a)\delta_j(M_i) + \delta_j\sigma_i(a)M_i + \delta_j\delta_i(a)\big)\, {}^tB.$$

En tenant compte, pour $i \neq j$ de la commutation $\partial_i\partial_j = \partial_j\partial_i$ et des commutations entre morphismes de corps et σ-dérivations données par la définition des algèbres de Ore, on déduit la relation suivante, qui remplace la commutation entre les matrices du cas commutatif,

$$\sigma_j(M_i)M_j + \delta_j(M_i) = \sigma_i(M_j)M_i + \delta_i(M_j).$$

Lorsque de telles relations sont assurées, la même méthode de recherche de dépendances linéaires par la méthode de Gauss que dans le cas commutatif s'applique et fournit un calcul de l'addition et du produit de fonctions ∂-finies, ou même d'une expression polynomiale en des fonctions ∂-finies. Plutôt que de faire une présentation formelle de ces algorithmes, nous en donnons l'idée sur un exemple.

Prenons celui du calcul du produit des deux fonctions f et g en deux variables x et y, données par $f(x, y) = \exp(xy)$ et $g(x, y) = J_\mu(x+y)$, où, pour un paramètre μ complexe, J_μ est la fonction de Bessel de première espèce, solution de l'équation différentielle

$$z^2 J''_\mu(z) + z J'_\mu(z) + (z^2 - \mu^2) J_\mu(z) = 0$$

qui admet à l'origine le développement asymptotique

$$J_\mu(z) \sim \frac{1}{2^\mu} \sum_{n=0}^{\infty} \frac{(-1)^n (z/2)^{2n}}{n!\,\Gamma(n + \mu + 1)}.$$

Considérons l'algèbre de Ore $A = \mathbb{C}(\mu, x, y)\langle \partial_x, \partial_y; I, I, D_x, D_y \rangle$, où μ est maintenant un paramètre formel. Des bases de Gröbner des annulateurs I et J dans A de f et g pour l'ordre $\mathsf{lex}(\partial_y, \partial_x)$ sont respectivement

$$\{\partial_x - y, \partial_y - x\} \qquad \text{et} \qquad \{(x+y)^2\partial_x^2 + (x+y)\partial_x + (x+y)^2 - \mu^2, \partial_y - \partial_x\}.$$

En désignant maintenant par f et g les vecteurs cycliques générateurs des modules A/I et A/J, avec un petit abus de notation, on introduit donc l'espace vectoriel sur $\mathbb{C}(\mu)$ de base $B = \big(f \otimes g, f \otimes (\partial_x \cdot g)\big)$, où l'on voit le produit $h = f \otimes g$ donné par ses coordonnées $(1, 0)$. Puisque

$$\partial_x \cdot (f \otimes g) = (\partial_x \cdot f) \otimes g + f \otimes (\partial_x \cdot g) = yf \otimes g + f \otimes (\partial_x \cdot g)$$

et

$$\begin{aligned}\partial_x \cdot \big(f \otimes (\partial_x \cdot g)\big) &= yf \otimes (\partial_x \cdot g) + f \otimes (\partial_x^2 \cdot g)\\ &= yf \otimes (\partial_x \cdot g) + \big((x+y)^{-2}\mu^2 - 1\big) f \otimes g - (x+y)^{-1} f \otimes (\partial_x \cdot g),\end{aligned}$$

et des relations similaires pour l'action de ∂_y, on trouve les matrices

$$M_x = \begin{pmatrix} y & 1 \\ (x+y)^{-2}\mu^2 - 1 & y - (x+y)^{-1} \end{pmatrix}$$

et

$$M_y = \begin{pmatrix} x & 1 \\ (x+y)^{-2}\mu^2 - 1 & x - (x+y)^{-1} \end{pmatrix}.$$

Choisissons d'itérer selon un ordre raffinant le degré total en ∂_x et ∂_y. On fait d'abord agir ∂_y pour trouver $\partial_y \cdot h = \big((1,0)M_y + d(1,0)/dy\big)\,{}^tB = (x, 1)\,{}^tB$, qui n'est pas lié avec $(1, 0)$. De même, on trouve $\partial_x \cdot h = (y, 1)\,{}^tB$, qui fournit la liaison $p_1 \cdot h = 0$ pour $p_1 = \partial_x - \partial_y + (x - y)$. Pour la suite du calcul, on exclut alors tous les monômes divisibles par ∂_x ; le monôme considéré suivant est ∂_y^2. Son action sur h donne

$$\partial_y^2 \cdot h = \big((x,1)M_y + d(x,1)/dy\big)\,{}^tB = \big((x+y)^{-2}\mu^2 + x^2 - 1, 2x - (x+y)^{-1}\big)\,{}^tB,$$

et l'on obtient un second annulateur de h,

$$p_2 = (x+y)^2\partial_y^2 - (x+y)(2x^2 + 2xy - 1)\partial_y + (x+y)(x^3 + x^2y + y) - \mu^2.$$

Pour la suite du calcul, on exclut donc tous les monômes divisibles par ∂_y^2, si bien qu'il ne reste plus aucun monôme à considérer. L'idéal annulateur de h est l'idéal $Ap_1 + Ap_2$, dont $\{p_1, p_2\}$ est une base de Gröbner pour l'ordre $\mathsf{lex}(\partial_y, \partial_x)$, de monômes de tête respectifs ∂_x et ∂_y^2 ; le module A/I est donné comme $\mathbb{C}(x, y)$-espace vectoriel par sa base $(1 + I, \partial_x + I)$.

Le calcul qui précède se revisite en abandonnant l'écriture matricielle et en faisant apparaître plus explicitement les calculs de restes modulo une base de Gröbner. On récrit d'abord $\partial_y \cdot h$ sous la forme

$$\partial_y \cdot h = (\partial_y \cdot f) \otimes g + f \otimes (\partial_y \cdot g) = xf \otimes g + f \otimes (\partial_x \cdot g),$$

après réductions par les bases de Gröbner pour I et J ; ce vecteur est donc linéairement indépendant de h. On procède ensuite de même pour $\partial_x \cdot h$, de façon à avoir

$$\partial_x \cdot h = (\partial_x \cdot f) \otimes g + f \otimes (\partial_x \cdot g) = yf \otimes g + f \otimes (\partial_x \cdot g);$$

on retrouve ainsi l'annulateur p_1. Le monôme considéré suivant est ∂_y^2, d'action sur h

$$\partial_y^2 \cdot h = x^2 f \otimes g + 2xf \otimes (\partial_x \cdot g) - (x+y)^{-2} f \otimes \big((x+y)\partial_x + (x+y)^2 - \mu^2\big) g;$$

ce vecteur est donc linéairement lié à h et $\partial_y \cdot h$ et l'on retrouve le second annulateur p_2. Le calcul se termine de la même manière.

Pour l'algorithme d'addition, les mêmes idées algorithmiques fonctionnent en calculant dans la somme directe $A/I \oplus A/J$.

Bibliographie

[1] Chyzak (Frédéric) and Salvy (Bruno). – Non-commutative elimination in Ore algebras proves multivariate holonomic identities. *Journal of Symbolic Computation*, vol. 26, n° 2, August 1998, pp. 187–227.

[2] Saito (Mutsumi), Sturmfels (Bernd), and Takayama (Nobuki). – *Gröbner deformations of hypergeometric differential equations*. – Springer-Verlag, Berlin, 2000, viii+254p.

COURS 25

Sommation et intégration symboliques des fonctions spéciales

Notes de Frédéric Chyzak

Résumé

Dans ce chapitre, nous décrivons un algorithme qui peut se voir comme une extension de l'algorithme de Zeilberger pour des sommants ∂-finis, et qui traite dans le même formalisme sommation et intégration. Les quelques sommes et intégrales suivantes, que nous envisageons de traiter avec cet algorithme, montrent une variété d'applications qui vont de la combinatoire à la physique mathématique en passant par la théorie des fonctions spéciales :

$$\sum_{k=0}^{n}\left(\sum_{j=0}^{k}\binom{n}{j}\right)^3 = n2^{3n-1} + 2^{3n} - 3n2^{n-2}\binom{2n}{n},$$

$$\sum_{n=0}^{\infty} H_n(x)H_n(y)\frac{u^n}{n!} = \frac{\exp\left(\frac{4u(xy-u(x^2+y^2))}{1-4u^2}\right)}{\sqrt{1-u^2}},$$

$$\frac{1}{2}J_0(x)^2 + J_1(x)^2 + J_2(x)^2 + \cdots = \frac{1}{2},$$

$$\int_{-1}^{+1} \frac{e^{-px}T_n(x)}{\sqrt{1-x^2}}\,dx = (-1)^n\pi I_n(p),$$

$$\int_0^{+\infty} xe^{-px^2} J_n(bx)I_n(cx)\,dx = \frac{1}{2p}\exp\left(\frac{c^2-b^2}{4p}\right) J_n\left(\frac{bc}{2p}\right),$$

$$\int_0^{+\infty} xJ_1(ax)I_1(ax)Y_0(x)K_0(x)\,dx = -\frac{\ln(1-a^4)}{2\pi a^2},$$

$$\sum_{k=0}^{n}\frac{q^{k^2}}{(q;q)_k(q;q)_{n-k}} = \sum_{k=-n}^{n}\frac{(-1)^k q^{(5k^2-k)/2}}{(q;q)_{n-k}(q;q)_{n+k}}.$$

Ici, J, Y, I et K sont des variantes de fonctions de Bessel, qui apparaissent fréquemment pour décrire des modèles physiques à symétrie cylindrique ou sphérique ; H et T sont des familles de polynômes orthogonaux de Hermite et Tchébichev ; $(q;q)_n$ représente le produit $(1-q)\cdots(1-q^m)$. La première identité intervient dans une discrétisation d'une question de probabilités sur la position du maximum de trois variables aléatoires gaussiennes ; la dernière est une variante finie d'une des identités de Rogers–Ramanujan, en théorie des partitions.

1. Expression de la création télescopique en termes d'algèbres de Ore rationnelles

On a déjà exposé dans ce cours la méthode de la création télescopique, en l'appliquant à la sommation hypergéométrique définie par une combinaison de l'algorithme de Gosper et d'une idée due à Zeilberger. Cette approche se généralise en des algorithmes de sommation et intégration pour les suites et fonctions ∂-finies.

Rappelons le principe de la méthode. Soit à évaluer une somme paramétrée

$$F_n = \sum_{k=a}^{b} f_{n,k}.$$

En toute généralité, le principe de la création télescopique est de déterminer une suite auxiliaire $g = (g_{n,k})$ ainsi que des coefficients η_0, …, η_r, fonctions de la variable n, tels que se trouve vérifiée la relation

$$\eta_r(n) f_{n+r,k} + \cdots + \eta_0(n) f_{n,k} = g_{n,k+1} - g_{n,k}.$$

Ici, nous ne faisons pas plus d'hypothèses sur les η_i et g que celle de pouvoir évaluer la relation ci-dessus pour tout n quand k décrit les entiers de a à b. Dans ce cas, une sommation sur k fournit l'égalité

$$\eta_r(n) F_{n+r} + \cdots + \eta_0(n) F_n = g_{n,b+1} - g_{n,a}.$$

Si le membre de droite n'est pas déjà nul, on recherche un opérateur annulateur de ce second membre ; par composition, on obtient une récurrence homogène sur F. Dans bien des cas, on sait prédire à partir de conditions analytiques sur f la nullité du terme $g_{n,b+1} - g_{n,a}$.

Des algorithmes d'efficacités différentes ont été donnés selon le domaine de recherche des η_i et de g, et selon le compromis choisi entre efficacité et richesse de la classe de suites f en entrée. En particulier, l'algorithme de Zeilberger, optimisé pour une suite f hypergéométrique, revient à rechercher des η_i polynomiaux et une suite g similaire à f, c'est-à-dire un multiple ϕf pour une fraction rationnelle ϕ en n et k. La suite $g = \phi f$ devant être une somme indéfinie, la recherche de ϕ et des η_i se fait par une variante paramétrée de l'algorithme de Gosper. Notons que le domaine de recherche de g est l'espace vectoriel $\mathbb{C}(n,k)f$, qui n'est autre, dans le cas hypergéométrique, que le module engendré par f sur l'algèbre de Ore $A = \mathbb{C}(n,k)\langle \partial_n, \partial_k; S_n, S_k\rangle$. Nous considérons ici la généralisation au cas où f est une fonction ∂-finie et où le module $A \cdot f$ est un espace vectoriel de dimension finie sur $\mathbb{C}(n,k)$, mais pas forcément de dimension 1. Soit v_1, …, v_d les éléments d'une base vectorielle de $A \cdot f$; l'algorithme de Zeilberger étendu recherche g sous la forme indéterminée $\phi_1 v_1 + \cdots + \phi_d v_d$, pour des fractions rationnelles ϕ_i en n et k. Cette recherche se fait par une extension ∂-finie de la variante paramétrée de l'algorithme de Gosper.

Tout ce qui a été dit s'étend au monde différentiel pour l'évaluation d'une intégrale paramétrée

$$F(x) = \int_a^b f(x,y)\, dy.$$

On cherche alors une relation

$$\eta_r(x) \frac{\partial^r f}{dx^r}(x,y) + \cdots + \eta_0(n) f(x,y) = \frac{\partial g}{dy}(x,y),$$

qui après intégration fournit l'égalité

$$\eta_r(n)F^{(r)}(x) + \cdots + \eta_0(n)F(x) = \int_a^b g(x,y)\,dy.$$

La même méthode permet aussi de traiter des sommations paramétrées continûment,

$$F(x) = \sum_{k=a}^{b} f_k(x),$$

et des suites d'intégrales de la forme

$$F_n = \int_a^b f_n(y)\,dy.$$

EXERCICE 1. Formuler la relation entre f et g à rechercher dans ces deux derniers cas.

2. L'algorithme sur l'exemple $\frac{1}{2}J_0(x)^2 + J_1(x)^2 + J_2(x)^2 + \cdots = \frac{1}{2}$

Nous allons montrer que la famille paramétrée des fonctions de Bessel de première espèce, J_ν, où chaque J_ν est une solution que nous allons préciser de l'équation de Bessel

$$x^2y''(x) + xy'(x) + (x^2 - \nu^2)y(x) = 0,$$

a une somme $\frac{1}{2}J_0(x)^2 + J_1(x)^2 + J_2(x)^2 + \cdots$ qui s'évalue à $\frac{1}{2}$.

L'équation de Bessel et les fonctions de Bessel peuvent être considérées pour des valeurs complexes du paramètre ν, mais vu la nature de la somme à étudier, nous nous limiterons dorénavant à des valeurs entières $\nu \in \mathbb{N}$. En étudiant l'équation indicielle de l'équation de Bessel, on s'aperçoit qu'il existe pour chaque ν des solutions dans les séries formelles $\mathbb{C}[[x]]$ et que ces solutions constituent un espace vectoriel de dimension 1 sur $\mathbb{C}$ de séries. Une base de ces solutions formelles est donnée par la série de Bessel

$$J_\nu(x) = (z/2)^\nu \sum_{n=0}^{\infty} \frac{(-1)^n (z/2)^{2n}}{n!\,(n+\nu)!},$$

de valuation ν, qui vu la décroissance de ses coefficients est pour chaque entier ν une série entière.

EXERCICE 2. Vérifier ces résultats.

On vérifie par simple substitution et évaluation que ces fonctions J_ν satisfont aussi aux relations

$$xJ'_\nu(x) + xJ_{\nu+1}(x) - \nu J_\nu(x) = 0 \qquad \text{et} \qquad xJ_{\nu+2}(x) - 2(\nu+1)J_{\nu+1}(x) + xJ_\nu(x) = 0.$$

En introduisant l'algèbre de Ore $A = \mathbb{C}(\nu, x)\langle \partial_\nu, \partial_x; S_\nu, I, 0, D_x\rangle$ où S_ν est le décalage avant sur ν et D_x est la dérivation par rapport à x, on a donc un système d'annulateurs pour J,

$$\begin{aligned} p_1 &= x^2\partial_x^2 + x\partial_x + x^2 - \nu^2, \\ p_2 &= x\partial_x + x\partial_\nu - \nu, \\ p_3 &= x\partial_\nu^2 - 2(\nu+1)\partial_\nu + x. \end{aligned}$$

Les deux premiers forment une base de Gröbner de l'idéal engendré pour l'ordre $\mathsf{lex}(\partial_\nu, \partial_x)$; les deux derniers pour l'ordre $\mathsf{lex}(\partial_x, \partial_\nu)$.

EXERCICE 3. Pour chacun des idéaux $Ap_1 + Ap_2$ et $Ap_2 + Ap_3$, calculer la base de Gröbner minimale réduite pour chacun des deux ordres $\mathsf{lex}(\partial_\nu, \partial_x)$ et $\mathsf{lex}(\partial_x, \partial_\nu)$.

Bien évidemment, J est une fonction ∂-finie. Le module $A \cdot J$ est donné, par exemple, comme l'espace vectoriel sur $\mathbb{C}(\nu, x)$ de base $(J, \partial_\nu \cdot J)$. Pour représenter le carré de J en vue d'une sommation, on peut observer que, en tant qu'espace vectoriel, le module $A \cdot J^2$ admet la base $\big(J^2, J \times (\partial_\nu \cdot J), (\partial_\nu \cdot J)^2\big)$ et utiliser l'algorithme de clôture par produit pour obtenir une base de Gröbner. En fait, le calcul qui suit n'a même pas besoin d'une représentation aussi explicite de $f = J^2$: pour calculer la somme $\frac{1}{2}J_0(x)^2 + J_1(x)^2 + J_2(x)^2 + \cdots$ comme fonction de x, on recherche une fonction η de x, indépendante de ν, telle que $f' + \eta f$ soit la différence finie en ν d'un élément g de $A \cdot J^2$. Pour la suite du calcul, nous fixons cet élément sous la forme indéterminée donnée par

$$g(\nu) = \phi_0(\nu)J_\nu^2 + \phi_1(\nu)J_{\nu+1}^2 + \phi_2(\nu)J_\nu J_{\nu+1},$$

où nous avons omis de faire référence à la variable x dans les évaluations de g, des ϕ_i et de J, car cette variable ne va intervenir que comme paramètre dans le calcul des fractions rationnelles ϕ_i. (On peut penser qu'on travaille temporairement dans l'algèbre de Ore $A' = \mathbb{C}(\nu, x)\langle \partial_\nu; S_\nu \rangle$.)

En supposant le problème résolu, on a alors par construction la relation $f' + \eta f = (\partial_\nu - 1) \cdot g$, puis, après réduction de chaque occurence des dérivées et décalées de J par la base de Gröbner $\{p_2, p_3\}$,

$$\begin{aligned}
&2J_\nu J'_\nu + \eta J_\nu^2 = (\partial_\nu - 1) \cdot \big(\phi_0(\nu)J_\nu^2 + \phi_1(\nu)J_{\nu+1}^2 + \phi_2(\nu)J_\nu J_{\nu+1}\big) \\
&= \phi_0(\nu+1)J_{\nu+1}^2 - \phi_0(\nu)J_\nu^2 + \phi_1(\nu+1)x^{-2}\big(2(\nu+1)J_{\nu+1} - xJ_\nu\big)^2 - \phi_1(\nu)J_{\nu+1}^2 \\
&\qquad + \phi_2(\nu+1)x^{-1}J_{\nu+1}\big(2(\nu+1)J_{\nu+1} - xJ_\nu\big) - \phi_2(\nu)J_\nu J_{\nu+1},
\end{aligned}$$

laquelle se récrit

$$\begin{aligned}
&\big(2\nu x^{-1} + \eta\big)J_\nu^2 - 2J_\nu J_{\nu+1} \\
&= \big(\phi_1(\nu+1) - \phi_0(\nu)\big)J_\nu^2 - \big(4(\nu+1)x^{-1}\phi_1(\nu+1) + \phi_2(\nu+1) + \phi_2(\nu)\big)J_\nu J_{\nu+1} \\
&\quad + \big(\phi_0(\nu+1) + 4(\nu+1)^2x^{-2}\phi_1(\nu+1) - \phi_1(\nu) + 2(\nu+1)x^{-1}\phi_2(\nu+1)\big)J_{\nu+1}^2.
\end{aligned}$$

De l'indépendance linéaire des fonctions J_ν^2, $J_{\nu+1}^2$ et $J_\nu J_{\nu+1}$ sur $\mathbb{C}(\nu, x)$, on déduit les relations nécessaires

$$\begin{aligned}
-\phi_0(\nu) + \phi_1(\nu+1) &= 2\nu x^{-1} + \eta, \\
4(\nu+1)x^{-1}\phi_1(\nu+1) + \phi_2(\nu) + \phi_2(\nu+1) &= 2, \\
\phi_0(\nu+1) - \phi_1(\nu) + 4(\nu+1)^2x^{-2}\phi_1(\nu+1) + 2(\nu+1)x^{-1}\phi_2(\nu+1) &= 0.
\end{aligned}$$

En résolvant les deux premières respectivement en ϕ_0 et en ϕ_1, puis en substituant dans la dernière, on trouve la récurrence

$$\begin{aligned}
&x^2(\nu+1)\phi_2(\nu+3) - (\nu+1)(4\nu^2 + 20\nu + 24 - x^2)\phi_2(\nu+2) \\
&+(\nu+3)(4\nu^2+12\nu+8-x^2)\phi_2(\nu+1) - x^2(\nu+3)\phi_2(\nu) = -4x(\eta\nu^2 + 4\eta\nu + 3\eta + x).
\end{aligned}$$

Nous résolvons maintenant celle-ci en ses solutions rationnelles par l'algorithme d'Abramov, dans sa variante paramétrée qui résoud en $\phi_2 \in \mathbb{C}(n, \nu)$ et simultanément en $\eta \in \mathbb{C}$. Les coefficients extrêmes de la partie homogène indiquent que toute solution rationnelle doit être polynomiale, puisque le p. g. c. d. entre $(\nu-3)+1$

et $x+3$ vaut 1. La mise sous forme de différences finies de la partie homogène indique que l'opérateur associé accroît de 2 le degré d'un polynôme. Le degré de la partie inhomogène étant 2, toute solution rationnelle ne peut être qu'une constante. On trouve $\phi_2 = 1$ et $\eta = 0$, d'où après report $\phi_1 = 0$ et $\phi_0 = -2\nu/x$. Autrement dit, on a

$$\partial_x \cdot J_\nu^2 = (\partial_\nu - 1) \cdot \left(J_\nu J_{\nu+1} - 2\nu x^{-1} J_\nu^2 \right),$$

qui par sommation fournit

$$\partial_x \cdot \sum_{\nu=0}^{N} J_\nu^2 = J_{N+1}\big(J_{N+2} - 2(N+1)x^{-1} J_{N+1}\big) - J_0 J_1.$$

Comme la série $J_N \in \mathbb{C}[[x]]$ a valuation N, le membre droit tend vers $-J_0 J_1 = \frac{1}{2}\partial_x \cdot J_0^2$ quand N tend vers ∞ pour la topologie usuelle donnée par la métrique $|s| = 2^{-v}$ pour toute série non-nulle s de valuation v. On a donc

$$\partial_x \cdot \left(\tfrac{1}{2} J_0(x)^2 + J_1(x)^2 + J_2(x)^2 + \cdots \right) = 0$$

qui caractérise la somme par une condition initiale. Une simple évaluation en 0 montre que la somme vaut $\frac{1}{2}$, ce qui achève la preuve de l'identité annoncée.

3. Bases de Gröbner de modules et découplage de systèmes

Dans l'exemple qui précède, on a pour le moment effectué le découplage « à la main », mais un procédé systématique et automatique est disponible, par le biais des bases de Gröbner de modules, qui généralisent la notion de base de Gröbner pour les idéaux. Cette notion existe tant dans le domaine des polynômes commutatifs que dans le cadre non-commutatif des algèbres de Ore ; nous la présentons directement dans ce second cas.

Dans le cas d'un idéal I d'une algèbre de Ore A, les éléments de I s'interprètent comme autant d'équations vérifiées par une fonction inconnue ϕ. Dans une perspective algorithmique, chaque idéal est engendré par un nombre fini de générateurs. Une question naturelle est celle de systèmes linéaires sur un vecteur de fonctions inconnues $(\phi_1, \ldots, \phi_d)$, à coefficients dans A. On considère des systèmes d'un nombre fini d'équations de la forme $g_i = g_{i,1} \cdot \phi_1 + \cdots + g_{i,d} \cdot \phi_d$ pour des polynômes tordus $g_{i,j}$ de A. Un tel système se représente de façon compacte par une matrice $(g_{i,j})$ à entrées dans A. Les questions qui sont alors naturelles sont celle de l'algèbre linéaire pour ces matrices, dont en particulier celle de donner un algorithme du pivot de Gauss pour des coefficients dans A, c'est-à-dire non plus dans un corps, mais dans un anneau, et non-commutatif de surcroît.

Pour ce faire, au lieu de considérer simplement un ordre monomial sur les monômes ∂^a d'une algèbre de Ore $A = k(x)\langle \partial; \sigma, \delta \rangle$, pour lequel tout idéal à gauche admet une base de Gröbner, on s'intéresse plus généralement à un module libre de rang fini sur A, donné par une base sous la forme $A^d = Ae_1 + \cdots + Ae_d$ et muni d'un ordre sur les $\partial^a e_i$, dans lequel on va étendre la notion de base de Gröbner pour des sous-modules à gauche de A^d. La notion d'ordre monomial conserve formellement la même définition, si ce n'est que les e_i ne peuvent apparaître que linéairement dans les monômes $\partial^a e_i$ et qu'ils ne peuvent servir pour des multiplications à gauche. La notion de S-polynôme s'étend aussi mot pour mot, à ceci près que deux polynômes de monômes de tête $\partial^a e_i$ et $\partial^b e_j$ ont un S-polynôme nul dès lors que i et j sont différents. Les définitions et caractérisations équivalentes des bases de Gröbner d'idéaux restent alors valables pour les sous-modules du module libre A^d.

L'algorithme de Buchberger, modifié pour suivre ces nouvelles définitions, termine sur tout sous-module en fournissant une base de Gröbner. Pour certains ordres, ce calcul correspond à l'algorithme de Gauss.

Un point de vue presque équivalent, mais qui donne une variante des calculs avec un peu plus de réductions, est que le calcul est celui d'une base de Gröbner dans l'anneau $A[e_1, \ldots, e_d]$ des polynômes en les indéterminées commutatives e_i à coefficients dans l'anneau A pour l'idéal à gauche engendré par les g_i initiaux et tous les produits $e_i e_j = 0$.

Reprenons l'exemple du découplage des relations entre les coordonnées ϕ_i donnant g dans la section précédente sur la somme des carrés fonctions de Bessel. Ces relations se recodent par les éléments

$$\begin{aligned}
g_1 &:= -e_0 + \partial_\nu e_1 - (2\nu x^{-1} + \eta)e_3,\\
g_2 &:= 4(\nu+1)x^{-1}\partial_\nu e_1 + (\partial_\nu + 1)e_2 - 2e_3,\\
g_3 &:= \partial_\nu e_0 + \big(4(\nu+1)^2 x^{-2}\partial_\nu - 1\big)e_1 + 2(\nu+1)x^{-1}\partial_\nu e_2,\\
g_4 &:= (\partial_\nu - 1)e_3,
\end{aligned}$$

du module libre A^4 pour l'algèbre de Ore $A = \mathbb{C}(n,k)\langle \partial_n, \partial_k; S_n, S_k\rangle$. Ici, chaque e_i représente la fonction rationnelle inconnue ϕ_i, et l'on a astucieusement représenté les second membres de équations inhomogènes d'origine comme multiple d'une nouvelle inconnue représentée par e_3 et contrainte par g_4 à être constante.

Le découplage effectué dans la section précédente revient au calcul d'une base de Gröbner pour l'ordre $\mathsf{lex}(e_0, e_1, e_2, e_3, \partial_\nu)$. Les monômes de tête respectifs des g_i sont e_0, $\partial_\nu e_1$, $\partial_\nu e_0$ et $\partial_\nu e_3$, si bien que le seul S-polynôme non nul est $\mathrm{Spoly}(g_1, g_3)$. Il est donné par

$$\begin{aligned}
&\mathrm{Spoly}(g_1, g_3) = \partial_\nu g_1 + g_3\\
&\quad = \big(\partial_\nu^2 + 4(\nu+1)^2 x^{-2}\partial_\nu - 1\big)e_1 + 2(\nu+1)x^{-1}\partial_\nu e_2 - \big(2(\nu+1)x^{-1} + \eta\big)\partial_\nu e_3.
\end{aligned}$$

Après réductions par g_2 et g_3, ce polynôme devient

$$g_5 = -e_1 - \left(\frac{x}{4(\nu+2)}\partial_\nu^2 + \frac{x^2 - 4\nu^2 - 12\nu - 8}{4x(\nu+2)}\partial_\nu + \frac{\nu+1}{x}\right)e_2 + \left(\frac{x}{2(\nu+2)} - \eta\right)e_3,$$

qui est adjoint à la base de Gröbner en cours de calcul. L'unique nouvel S-polynôme à considérer est celui entre ce g_5 et g_2, qui est $\mathrm{Spoly}(g_2, g_4) = g_2 + 4(\nu+1)x^{-1}\partial_\nu g_5$ et a pour monôme de tête $\partial_\nu^3 e_2$. Après réduction par e_3 et renormalisation, le dernier polynôme introduit dans la base de Gröbner est

$$\begin{aligned}
\big((\nu+1)x^2\partial_\nu^3 + (\nu+1)(x^2 - 4\nu^2 - 20\nu - 24)\partial_\nu^2 \\
- (\nu+3)(x^2 - 4\nu^2 - 12\nu - 8)\partial_\nu - (\nu+3)x^2\big)e_2 \\
+ 4\big((\nu^2 + 4\nu + 3)\eta + x\big)x e_3.
\end{aligned}$$

Ce polynôme n'est autre qu'un recodage de l'équation inhomogène du troisième ordre qui a permis de déterminer ϕ_2 dans la section précédente.

Bibliographie

[1] Chyzak (Frédéric). – An extension of Zeilberger's fast algorithm to general holonomic functions. *Discrete Mathematics*, vol. 217, n° 1-3, 2000, pp. 115–134.

Quatrième partie

Algèbre Linéaire Avancée et Factorisation

COURS 26

Algèbre linéaire creuse : algorithme de Wiedemann

1. Introduction

Dans cette section, on aborde des questions radicalement différentes : on ne cherche plus à effectuer les produit, inversion, ... de matrices quelconques, mais à manipuler des matrices *creuses*.

On ne commencera pas par définir précisément de ce qu'est une matrice creuse. L'approche consiste à donner des algorithmes dont la complexité s'exprime en fonction du nombre d'entrées non nulles des matrices en entrées. De la complexité des algorithmes « creux » va découler une borne, typiquement de l'ordre $O(n)$ pour des matrices de taille $n \times n$, sur le nombre de coefficients non nuls pour que le changement de modèle soit pertinent.

Dans tout ce qui suit, on considère donc une matrice A de taille $n \times n$ et vérifiant les hypothèses suivantes :

- A est inversible,
- A contient s entrées non nulles.

On va montrer un algorithme dû à Wiedemann qui permet de calculer l'unique solution du système $Ax = y$ avec une complexité en $O(ns)$; l'algorithme original est plus général que celui présenté ici, en ce qu'il permet de traiter les matrices rectangulaires ou carrées et non inversibles, mais il se ramène au cas carré inversible.

Remarquons que si s est de l'ordre de n^2, caractérisant donc des matrices plutôt pleines, on retombe dans une complexité de l'ordre de $O(n^3)$. Le cas intéressant est celui où s est de l'ordre de n, auquel cas l'algorithme est quadratique en n : on gardera en tête que l'exposant de l'algèbre linéaire creuse est 2 dans les bons cas.

1.1. Polynôme minimal et résolution de systèmes.

L'algorithme de Wiedemann passe par le calcul du *polynôme minimal* de A. Rappelons sa définition.

Si k est le corps de base, il est possible d'associer à tout polynôme en une variable $P \in k[T]$ la matrice $P(A)$. On sait que d'après le théorème de Cayley-Hamilton, $\chi(A) = 0$, où χ est le polynôme caractéristique de A. Le polynôme minimal de A est le polynôme unitaire de plus petit degré ayant cette propriété (on montre facilement que cette propriété le rend unique).

Soit P le polynôme minimal de A, supposé connu. Puisqu'on a supposé A inversible, le terme constant de P n'est pas nul (car il divise le terme constant du polynôme caractéristique, qui n'est lui-même pas nul).

L'égalité $P(A) = 0$ se réécrit sous la forme

$$A^{-1} = \frac{-1}{p_0} \left(A^{m-1} + p_{m-1} A^{m-2} + \cdots + p_1 I_n \right)$$

Pour résoudre le système $Ax = y$, on va calculer $x = A^{-1}y$, c'est-à-dire

$$\frac{-1}{p_0}\left(A^{m-1}y + p_{m-1}A^{m-2}y + \cdots + p_1 y\right).$$

Ainsi, il suffit de calculer les itérés $y, Ay, \ldots, A^{m-1}y$, puis d'additionner les termes $p_1 y, p_2 Ay, \ldots, A^{m-1}y$, pour retrouver x.

- Chacun des $A^i y$ s'obtient à partir de $A^{i-1}y$ en $O(s)$ opérations.
- Chacun des produits par p_i, et chaque addition, se fait en $O(n)$ opérations. Remarquer que $n \leq s$ (hypothèse d'inversibilité de A).

Au total, on a donc $O(ns)$ opérations à faire pour résoudre le système si on connaît le polynôme minimal de A.

1.2. Calcul du polynôme minimal. écrivons le polynôme minimal de A sous la forme

$$P = T^m + p_{m-1}T^{m-1} + \cdots + p_0.$$

Alors la suite des puissances A satisfait la récurrence linéaire

$$A^{k+m} + p_{m-1}A^{k+m-1} + \cdots + p_0 A^k = 0,$$

et ne satisfait pas de récurrence d'ordre plus petit.

Pour tous vecteurs u et v, la suite (de nombres) $N_i := u^t A^i v$ vérifie donc

$$N_{k+m} + p_{m-1}N_{k+m-1} + \cdots + p_0 N_k = 0.$$

Si u et v sont mal choisis (nuls, par exemple), la suite N_i satisfait une récurrence d'ordre plus petit que m. La proposition suivante montre qu'en choisissant u et v au hasard, on tombe vraisemblablement sur une suite N_i qui ne satisfait pas de récurrence d'ordre plus petit.

PROPOSITION 1. *Il existe un polynôme non nul D dans $k[U_1, \ldots, U_n, V_1, \ldots, V_n]q$, de degré au plus $2n$, tel que si $D(u, v)$ est non nul, la suite N_i associée à u et v ne satisfait pas de récurrence d'ordre plus petit que m.*

L'idée est de choisir u et v au hasard. Si on n'a pas de chance, $D(u, v)$ est nul ; sinon, la suite des N_i associée à u et v permet de retrouver P par un calcul d'approximants de Padé (attention, on ne connaît pas explicitement le polynôme D, mais son existence assure que la plupart des choix de u et v sont "chanceux").

Algorithme de Wiedemann.:

Entrée :: la matrice A.

Sortie :: son polynôme minimal.

1. Choisir des vecteurs u et v aléatoires.
2. Pour $0 \leq i \leq 2n$, calculer les vecteurs $A^i v$, puis les scalaires $N_i = u^t A^i v$.
3. Retrouver la récurrence satisfaite par les N_i.

La complexité de l'étape 2 est $O(n)$ produits matrice-vecteur ou vecteur-vecteur, chacun prenant au pire $O(s)$ opérations ; au total on obtient donc $O(ns)$ opérations pour cette phase. L'algorithme d'approximants de Padé est négligeable, puisqu'il ne demande que $O(\mathsf{M}(n)\log(n)) \leq sn$ opérations.

COURS 27

Solutions rationnelles de systèmes linéaires à coefficients polynomiaux

Résumé

L'algorithmique des systèmes linéaires à coefficients des polynômes en une variable est très similaire à celle des fractions rationnelles. En outre, il est important de tirer parti du produit rapide de matrices.

On considère le système linéaire

$$A(x)Y(x) = B(x), \tag{1}$$

où A et B sont donnés et Y est inconnu. A est une matrice $n \times n$ de polynômes, régulière (de déterminant non nul) et B est un vecteur de polynômes. De manière équivalente, on peut voir A (ou B) comme un polynôme à coefficients des matrices (ou des vecteurs). Pour fixer les notations, on suppose $\deg A \leq d$ et $\deg B < d$.

On notera $\mathcal{M}_{m,k}(R)$ l'ensemble des matrices $m \times k$ (m lignes et k colonnes) dont les coefficients sont dans R. On notera $\mathbb{K}[x]_d$ l'ensemble des polynômes de degré au plus d à coefficients dans le corps $\mathbb{K}$. La fonction M est telle que la multiplication dans $\mathbb{K}[x]_d$ coûte au plus $M(d)$ opérations dans $\mathbb{K}$. L'exposant ω est tel que la multiplication de deux matrices $n \times n$ à coefficients dans k coûte au plus n^ω opérations dans k.

Nous aurons aussi besoin de produits de matrices de $\mathcal{M}_{n,n}(\mathbb{K}[x]_d)$. Dans le cas le plus général, ce produit est connu pour pouvoir être executé en $O(n^\omega M(d))$ opérations dans $\mathbb{K}$, borne que nous utiliserons dans les estimations de complexité. Lorsque le corps $\mathbb{K}$ contient suffisamment de points, par évaluation-interpolation, ce coût descend à $O(n^\omega d + n^2 M(d) \log d)$; dans les mêmes conditions, en choisissant des points en progression géométrique, cette complexité a été abaissée récemment à $O(n^\omega d + n^2 M(d))$, voir [1].

1. Des séries aux solutions rationnelles

Une première observation est que la structure de la solution cherchée est donnée par les formules de Cramer.

Lemme 1. *Le système possède une solution dont les coordonnées sont des fractions rationnelles. Ses numérateurs et dénominateurs ont degrés bornés par $nd-1$ et nd.*

Démonstration. Le système (1) se récrit

$$A_1 y_1 + \cdots + A_n y_n = B,$$

où y_i est la $i$ème coordonnée de Y et A_i la $i$ème colonne de A. La formule de Cramer

$$\det(A_1, \ldots, A_{i-1}, B, A_{i+1}, \ldots, A_n) = y_i \det(A),$$

est obtenue en remplaçant B par sa valeur dans le membre gauche et en développant le déterminant.

Il en découle que y_i est le quotient de déterminants de matrices dans $\mathcal{M}_{n,n}(\mathbb{K}[x]_d)$. Ces déterminants ont donc degré au plus $nd-1$ pour le numérateur et nd pour le dénominateur. □

L'algorithme de résolution suivant est justifié par la complexité quasi-optimale des approximants de Padé.

Résolution de $A(x)Y(x) = B(x)$ [Moenk-Carter 1979]
Entrée : $A \in \mathcal{M}_{n,n}(\mathbb{K}[x]_d)$, $B \in \mathcal{M}_{n,1}(\mathbb{K}[x]_{d-1})$
Sortie : le vecteur de fraction rationnelles Y tel que $AY = B$.

1. Calculer le développement en série de $A^{-1}B$ à précision $2nd$;
2. Reconstruire les coefficients de Y par approximant de Padé.

Dans tous les algorithmes qui vont suivre, c'est la recherche de série solution qui va dominer la complexité.

2. L'algorithme de Newton

La méthode de Newton que nous avons employée pour de nombreux calculs rapides sur les séries s'applique encore lorsque les séries ont des matrices pour coefficients.

LEMME 2. *Soit $A(x)$ une matrice $n \times n$ de séries formelles telle que $A(0)$ est inversible. Alors l'itération*

$$Y_{k+1} = Y_k + Y_k(I - AY_k) \bmod x^{2^{k+1}}$$

avec $Y_0 = A(0)^{-1}$ converge vers A^{-1} avec $A^{-1} - Y_k = O(x^{2^k})$.

L'itération de l'étape k utilise deux produits de matrices $n \times n$ de polynômes de degré au plus 2^{k+1} et a donc complexité $O(n^\omega M(2^{k+1}))$. Le lemme de complexité du diviser-pour-régner et les hypothèses habituelles sur la fonction M donnent alors la proposition suivante.

PROPOSITION 1. *Avec les hypothèses du lemme prédédent, on peut calculer N termes de la série A^{-1} en $O(n^\omega M(N))$ opérations dans $\mathbb{K}$.*

DÉMONSTRATION. [du lemme] Pour $k = 0$, le lemme découle du choix de Y_0. Si la propriété est acquise pour k, alors $I - AY_k = O(x^{2^k})$. Ensuite,

$$\begin{aligned} I - AY_{k+1} &= I - AY_k - AY_k(I - AY_k) \bmod x^{2^{k+1}} \\ &= (I - AY_k)^2 \bmod x^{2^{k+1}} \\ &= O(x^{2^{k+1}}), \end{aligned}$$

d'où la propriété voulue pour $k+1$ en multipliant à gauche par A^{-1}. □

Avec cet algorithme, le calcul de la fraction rationnelle solution de (1) demande $O(n^\omega M(nd))$ opérations dans $\mathbb{K}$, résultat que nous allons améliorer dans les sections qui suivent.

3. Développement comme une fraction rationnelle

La méthode de Newton fonctionne pour toute matrice inversible de séries. Il est possible d'améliorer l'efficacité lorsque la matrice est une matrice de polynômes. La base de cette amélioration est contenue dans le lemme suivant.

LEMME 3. *Soit $A(x) \in \mathcal{M}_{n,n}(\mathbb{K}[x]_d)$ et $B(x) \in \mathcal{M}_{n,m}(\mathbb{K}[x]_{d-1})$, avec A inversible. Pour tout k, il existe une matrice $B_k \in \mathcal{M}_{n,m}(\mathbb{K}[x]_{d-1})$ telle que*

$$A^{-1}B = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} + x^k A^{-1} B_k, \tag{2}$$

où $a_i \in \mathcal{M}_{n,m}(\mathbb{K})$.

DÉMONSTRATION. Si les a_i sont les coefficients du développement en série de $A^{-1}B$, alors

$$B - A(a_0 + \cdots + a_{k-1}x^{k-1})$$

est une matrice de $\mathcal{M}_{n,m}(\mathbb{K}[x]_{d+k-1})$ qui, par construction, est divisible par x^k, d'où le lemme. □

Ce résultat se traduit algorithmiquement comme suit.

Développement de $A^{-1}B$
Entrée : A, B, k et $S = A^{-1} \bmod x^k$;
Sortie : $a_0, \ldots, a_{k-1}$ et B_k définis par l'équation (2).

1. Calculer $SB =: a_0 + \cdots + a_{k-1}x^{k-1} \bmod x^k$;
2. Calculer $B_k = (B - A(a_0 + \cdots + a_{k-1}x^{k-1}))x^{-k}$.
3. Renvoyer $a_0, \ldots, a_{k-1}, B_k$.

PROPOSITION 2. *Soit $A \in \mathcal{M}_{n,n}(\mathbb{K}[x]_d)$ avec $A(0)$ inversible, alors on peut calculer les $N \geq d$ premiers coefficients de A^{-1} en $O(n^\omega NM(d)/d)$ opérations dans $\mathbb{K}$. Pour $B \in \mathcal{M}_{n,1}(\mathbb{K}[x]_{d-1})$, on peut calculer les $N \geq d$ premiers coefficients de $A^{-1}B$ en $O(n^2 NM(d)/d)$ opérations dans $\mathbb{K}$.*

DÉMONSTRATION. L'algorithme calcule d'abord $S = A^{-1} \bmod x^d$ en $O(n^\omega M(d))$ opérations dans $\mathbb{K}$ par l'algorithme de Newton de la première section.

Ensuite, on applique N/d fois l'algorithme ci-dessus avec $k = d$ pour calculer à chaque itération d coefficients et un nouveau $B_{(i+1)d}$. Si on part de $B_0 = I$, le résultat fournit les N premiers coefficients de A^{-1} ; si $B_0 = B$, alors on obtient les coefficients de $A^{-1}B$.

Les deux étapes de l'algorithme ci-dessus (avec $k = d$) coûtent $O(n^\omega M(d))$ opérations dans $\mathbb{K}$ si B a n colonnes, $O(n^2 M(d))$ s'il n'en a qu'une. □

Avec cet algorithme, le calcul de la fraction rationnelle solution de (1) demande $O(n^3 M(d))$ opérations dans $\mathbb{K}$.

4. L'algorithme de Storjohann

L'algorithme de Storjohann permet de calculer les N premiers coefficients du développement en série de $A^{-1}B$ en $O(n^{\omega-1} N \log N M(d))$ opérations dans $\mathbb{K}$. Lorsque $\omega < 3$, cette quantité croît avec n moins vite que la taille de l'inverse A^{-1}, qui n'est donc pas calculée en entier. Ainsi, la résolution du système linéaire (1) a un coût en $O(n^\omega M(d) \log(nd))$ opérations dans $\mathbb{K}$.

L'algorithme repose sur le développement de $A^{-1}B$ de la section précédente, joint d'une part à une technique de diviser-pour-régner, d'autre part au regroupement des calculs intermédiaires pour remplacer des groupes de n produits matrice-vecteur par des produits matrice-matrice.

Pour commencer, on peut modifier l'algorithme de développement de $A^{-1}B$ de la section précédente pour calculer B_k sans calculer tous les coefficients $a_0, \ldots, a_{k-1}$. L'entrée nécessaire est moins grosse, et la complexité plus faible lorsque k est grand devant d. Pour une série $V = v_0 + v_1 x + \cdots$, on note

$$[V]_a^b := v_a x^a + \cdots + v_{a+b} x^{a+b},$$

avec la convention que les coefficients d'indice négatif de V sont nuls.

Développement de $A^{-1}B$ tronqué
Entrée : A, B, k et $S = [A^{-1}]_{k-2d+1}^{2d-2}$;
Sortie : B_k défini par l'équation (2).

1. Calculer $U := [SB]_{k-d}^{d-1}$;
2. Calculer $B_k := [B - AU]_k^{d-1} x^{-k}$.
3. Renvoyer B_k.

Démonstration. Pour prouver la correction de cet algorithme, il faut s'assurer que les troncatures de séries sont suffisantes pour calculer le même polynôme B_k que précédemment.

Pour la première étape de l'algorithme, il suffit d'observer que B ayant degré au plus $d-1$, le coefficient de x^i dans $A^{-1}B$ ne dépend que de $[A^{-1}]_{i-d-1}^{d+1}$, pour tout i. Donc les coefficients calculés par cette première étape sont les mêmes que les a_i que précédemment, pour $i = k-d, \ldots, k-1$.

Ensuite, il faut calculer $[x^k B_k]_k^{d-1}$. L'extraction des coefficients de l'équation (2) donne

$$\begin{aligned}[x^k B_k]_k^{d-1} &= [B - A(a_0 + \cdots + a_{k-1}x^{k-1})]_k^{d-1} \\ &= [B - A(a_{k-d}x^{k-d} + \cdots + a_{k-1}x^{k-1})]_k^{d-1},\end{aligned}$$

ce qui conclut la preuve. □

Une observation importante concernant cet algorithme est que c'est le même polynôme S qui peut servir pour calculer B_{i+k} pour tout i. L'idée est alors de grouper plusieurs vecteurs B_i et de calculer les B_{i+k} correspondants par des produits matrice-matrice. Ainsi, si l'on connaît $B_0, B_{2m}, \ldots, B_{2sm}$ et $[A^{-1}]_{m-2d}^{2d}$, alors le calcul de $B_m, B_{3m}, \ldots, B_{(2s+1)m}$ ne requiert que $O(n^{\omega-1} s M(d))$ opérations dans $\mathbb{K}$.

Itérant cette idée, partant de B_0 et supposant connus $[A^{-1}]_{2^k-2d+1}^{2d-2}$ pour $k = 0, \ldots, \lceil \log(N/d) \rceil =: k_{\max}$, on obtient d'abord $B_0, B_{2^{k_{\max}}}$, puis à l'étape suivante $B_0, B_{2^{k_{\max}-1}}, B_{2^{k_{\max}}}, B_{3.2^{k_{\max}-1}}$, et ainsi de suite jusqu'à $B_0, B_d, B_{2d}, \ldots, B_{d\lceil N/d \rceil}$ en $O(k_{\max} n^{\omega-1} N M(d)/d)$ opérations dans k. En multipliant alors ces vecteurs par $[A^{-1}]_0^d$ on obtient finalement les N premiers coefficients de $A^{-1}B$ pour le même coût.

Il reste à voir comment calculer les $[A^{-1}]_{2^k-2d-1}^{2d-2}$. Là encore, le point de départ est l'identité (2), avec $B = I$, $k = m$ et $k = p$:

$$\begin{aligned} A^{-1} &= a_0 + \cdots + a_{m-1}x^{m-1} + x^m A^{-1} B_m \\ &= a_0 + \cdots + a_{m-1}x^{m-1} + x^m(a_0 + \cdots + a_{p-1}x^{p-1} + x^p A^{-1} B_p)B_m. \end{aligned}$$

La seconde ligne est obtenue par substitution de la valeur de A^{-1} donnée par la première ligne avec $m = p$. Ces équations entraînent pour tout $\ell \geq 0$ tel que $m + p - \ell \geq d$

$$\begin{cases} B_{m+p} & = [-A[A^{-1}]_{p-2d+1}^{2d-2} B_m]_p^{d-1} x^{-p}, \\ [A^{-1}]_{m+p-\ell}^{\ell-1} & = [[A^{-1}]_{p-\ell-d+1}^{\ell+d-2} B_m]_{m+p-\ell}^{\ell-1}. \end{cases}$$

L'algorithme suivant s'en déduit en utilisant cette identité avec $m = 2^k - d$, $p = 2^k$ et $\ell = d - 1$.

Développement de A^{-1} — indices puissances de 2
Entrée : $S = [A^{-1}]_0^{d-1}$,$T = [A^{-1}]_{2^k-2d+1}^{2d-2}$ et B_{2^k-d} défini par (2) avec $B = I$;
Sortie : $B_{2^{k+1}-d}$ et $[A^{-1}]_{2^{k+1}-2d+1}^{2d-2}$.

1. Calculer $[A^{-1}]_{2^{k+1}-2d+1}^{d-2} = [TB_{2^k-d}]_{2^{k+1}-2d+1}^{d-2}$;
2. Calculer $B_{2^{k+1}-d} := [-ATB_{2^k-d}]_{2^k}^{d-1}/x^{2^k}$;
3. Calculer $[A^{-1}]_{2^{k+1}-d}^{d-1} = [x^{2^{k+1}-d} B_{2^{k+1}-d} S]_{2^{k+1}-d}^{d-1}$;
4. Renvoyer $B_{2^{k+1}-d}$ et $[A^{-1}]_{2^{k+1}-2d+1}^{2d-2}$.

Pour résumer, ces algorithmes mènent au résultat suivant.

Théorème 1 (Storjohann 2002). *Soit A une matrice $n \times n$ polynomiale de degré d avec $A(0)$ inversible et B un vecteur polynomial de degré au plus $d - 1$, alors on peut calculer le numérateur et le dénominateur de $A^{-1}B$ en $O(n^\omega M(d))$ opérations dans $\mathbb{K}$.*

Notes

Dans tout ce texte, nous avons supposé que $A(0)$ est inversible. En fait, ces résultats s'étendent au cas où A est inversible sans que $A(0)$ le soit. Il suffit de tirer aléatoirement un point et d'effectuer les développements en série au voisinage de ce point. L'algorithme devient probabiliste. Si la caractéristique est positive, il se peut que A soit inversible sans que sa valeur en aucun point du corps ne le soit. On peut alors construire une extension du corps assez grande pour trouver un point ou effectuer ces calculs. Ces considérations sont prises en compte dans [3].

Bibliographie

[1] Bostan (Alin) and Schost (Éric). – Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, vol. 21, n° 4, 2005, pp. 420–446. – Festschrift for Arnold Schönhage.

[2] Moenck (Robert T.) and Carter (John H.). – Approximate algorithms to derive exact solutions to systems of linear equations. In *EUROSAM '79 : Proceedings of the International Symposiumon on Symbolic and Algebraic Computation. Lecture Notes in Computer Science*, vol. 72, pp. 65–73. – Springer-Verlag, London, UK, 1979.

[3] Storjohann (Arne). – High-order lifting. In Mora (Teo) (editor), *ISSAC'2002*. pp. 246–254. – ACM Press, July 2002. Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, July 07–10, 2002, Université de Lille, France.

COURS 28

Réduction de réseaux et algorithme LLL

Notes de Frédéric Chyzak

Résumé

La recherche de vecteurs courts dans des réseaux de vecteurs à coefficients entiers permet, en calcul formel, la factorisation en temps polynomial et la recherche de dépendances linéaires entre constantes réelles données par des approximations, et, dans des domaines plus proches de la cryptanalyse, la mise en évidence de faiblesses de cryptosystèmes et de générateurs pseudo-aléatoires. Nous présentons l'algorithme LLL maintenant célèbre pour la réduction de réseaux et analysons sa complexité.

Ce texte suit d'assez près mais dans un autre ordre les chapitres 16 et 17 du livre *Modern computer algebra* de von zur Gathen et Gerhard.

1. Réseaux, vecteurs courts et résultats principaux

On appelle *réseau* de $\mathbb{Z}^n$ un $\mathbb{Z}$-module $\sum_{i=1}^{n} \mathbb{Z}v_i$ engendré par des vecteurs v_i linéairement indépendants sur $\mathbb{Z}$. Le thème de ces notes de cours est la recherche de vecteurs « courts » dans un réseau donné : par exemple, le réseau engendré par les vecteurs $(12, 2)$ et $(13, 4)$ contient le vecteur plus court $(2, 1)$. Ici, court s'entend pour la norme euclidienne : un vecteur $v = (v_1, \ldots, v_n)$ a pour norme $\|v\| = (v_1^2 + \cdots + v_n^2)^{1/2}$.

Cette question est motivée par un certain nombre d'applications :

- factorisation en temps polynomial ;
- cassage d'un cryptosystème basé sur le problème du sac-à-dos ;
- mise en évidence de la faiblesse de générateurs pseudo-aléatoires ;
- recherche de dépendances linéaires entre nombres donnés par des approximations numériques, dont la recherche du polynôme minimal d'un nombre algébrique.

La première de ces applications fera l'objet d'un autre cours ; la deuxième et la dernière sont détaillées dans la section qui suit.

La recherche d'un vecteur de norme minimale dans un réseau donné est un problème difficile, en fait, même NP-dur. Il est donc naturel de relâcher la contrainte de minimalité et de considérer le problème de la recherche approchée à un facteur constant près, mais le problème de la recherche à un facteur $\sqrt{2}$ près reste NP-dur. En revanche, le problème redevient polynomial si on autorise le facteur d'approximation à croître avec la dimension n du réseau.

De façon précise, on décrit dans la suite l'algorithme LLL, qui pour une base $(v_1, \ldots, v_n)$ d'un réseau L renvoie en particulier un vecteur u approximant les plus courts vecteur du réseau à pas plus d'un facteur $2^{(n-1)/2}$ près :

$$\|u\| \leq 2^{(n-1)/2} \min\{\, \|f\| : f \in L,\ f \neq 0 \,\}.$$

Cet algorithme termine après $O(n^4 \log A)$ opérations arithmétiques correspondant à $O_{\log}(n^5 \log^2 A)$ opérations binaires, où la constante A borne chacun des v_i. (Seule la complexité arithmétique est démontrée dans ce qui suit.)

2. Applications

2.1. Cassage du cryptosystème de Merkle et Hellman. Merkle et Hellman ont proposé en 1978 un cryptosystème basé sur le problème du sac-à-dos. Ce cryptosystème devait nécessiter des calculs moins lourds que le célèbre système RSA. Néanmoins, une faiblesse du système reposait sur l'existence d'un vecteur court dans un réseau.

L'idée de Merkle et Hellman est la suivante. Bob se dote d'une clé secrète, une famille d'entiers positifs b_1, …, b_n, telle que chaque b_i soit supérieur à la somme des précédents. Bob se donne aussi un multiplicateur c et un module m, eux deux aussi secrets. Il publie la clé privée constituée des restes positifs a_i de cb_i modulo m. Quand Alice veut lui envoyer le message constitué de la suite de bits $(x_1, \ldots, x_n)$, elle construit la somme $s = x_1a_1 + \cdots + x_na_n$ qui constitue le message crypté. Bob n'a plus qu'à multiplier par l'inverse de c modulo m et à tester si les b_i sont dans la somme restante (en commençant par les plus grands) pour déterminer les x_i.

Le problème sur lequel s'appuie ce cryptosystème, celui du sac-à-dos, consiste précisément à décider l'existence des $x_i \in \{0, 1\}$ tels que $s = x_1a_1 + \cdots + x_na_n$. Ce problème est NP-complet en l'absence d'hypothèse sur la famille des a_i. Mais dans le cas présent, l'origine des a_i crée une faiblesse cryptographique. Considérons les vecteurs $(0, \ldots, 0, 1, 0, \ldots, 0, -a_i) \in \mathbb{Z}^{n+1}$, où 1 est en i-ième position, ainsi que le vecteur $(0, \ldots, 0, s)$. Tous ces vecteurs sont « longs », puisque de l'ordre de m, mais le réseau qu'ils engendrent contient le vecteur $(x_1, \ldots, x_n, 0)$, lequel est manifestement très court. L'algorithme LLL qui va suivre permet de le retrouver en complexité polynomiale.

2.2. Relations de dépendance entre constantes numériques. Étant donnés n nombres réels non nuls, r_1, …, r_n, une relation de dépendance linéaire à coefficients entiers entre ces réels est une relation de la forme

$$c_1r_1 + \cdots + c_nr_n = 0$$

pour des coefficients c_i entiers non tous nuls. Lorsqu'on se donne une approximation rationnelle de chaque réel r_i, ou mieux, la possibilité d'obtenir autant de chiffres décimaux que souhaité, la recherche des vecteurs courts dans un réseau permet la détermination de relations de dépendance linéaire entre les r_i. Pour cela, on considère les combinaisons linéaires à coefficients entiers des vecteurs lignes de la matrice

$$F = \begin{bmatrix} 1 & 0 & \ldots & 0 & a_1 \\ 0 & 1 & \ldots & 0 & a_2 \\ & & \ddots & & \vdots \\ 0 & \ldots & 0 & 1 & a_{n-1} \\ 0 & \ldots & 0 & 0 & a_n \end{bmatrix},$$

où chaque a_i est une troncature entière de Nr_i pour un grand entier N. (Par exemple, N est une grande puissance de 10.)

Un vecteur court est alors de la forme

$$v = (c_1, \ldots, c_{n-1}, c_1a_1 + \cdots + c_na_n) \simeq \big(c_1, \ldots, c_{n-1}, N(c_1r_1 + \cdots + c_nr_n)\big).$$

En particulier,

$$|c_1 r_1 + \cdots + c_n r_n| \simeq \left|N^{-1}(c_1 a_1 + \cdots + c_n a_n)\right| \leq N^{-1}|v|$$

se doit d'être petit, ce qui fait de

$$c_1 r_1 + \cdots + c_n r_n = 0$$

un bon candidat pour une relation de dépendance entre les r_i. Bien qu'un tel argument ne constitue pas une preuve, la preuve formelle de relation entre constantes a dans un bon nombre de cas été grandement facilitée une fois que la bonne relation a pu être découverte expérimentalement par la méthode proposée ci-dessus. Dans un certain nombre de cas, même, des identités n'ont pu être prouvées que parce qu'elles avaient été découvertes heuristiquement en utilisant l'algorithme LLL.

Donnons un exemple. Des arguments théoriques donnent la certitude que le nombre

$$V = \int_0^\infty \frac{\sqrt{x}\,\ln^5 x}{(1-x)^5}\,\mathrm{d}x$$

peut se représenter comme l'évaluation en π d'une fonction polynomiale à coefficients rationnels. Il s'agit donc de trouver une dépendance linéaire entre

$$V, 1, \pi, \ldots, \pi^d$$

pour un degré de polynôme d à déterminer. Tout système de calcul formel généraliste connait des approximations pour π et permet de calculer aisément une approximation numérique de V. On prend donc par exemple $a_1 = N = 10^{25}$, $a_2 = 31415926535897932384626434 \simeq N\pi$, …, $a_9 = 94885310160705740071285755038 \simeq N\pi^8$, $a_{10} = -16699473719229070496187243 3 \simeq NV$ pour construire la matrice F. Les normes des vecteurs lignes de cette matrice sont toutes supérieures à $N = 10^{25}$. Par l'algorithme LLL, on trouve une base du même réseau de vecteurs, dont le premier vecteur ligne,

$$v = (c_1, \ldots, c_{n-1}, c_1 a_1 + \cdots + c_n a_n) = (0, 0, 120, 0, 140, 0, -15, 0, 0, 33),$$

est de norme inférieure à 200, tous les autres vecteurs de base étant de norme supérieure à 400. Les a_i étant connus, on trouve

$$(c_1, \ldots, c_n) = (0, 0, 120, 0, 140, 0, -15, 0, 0, 24),$$

d'où la relation linéaire

$$V = \int_0^\infty \frac{\sqrt{x}\,\ln^5 x}{(1-x)^5}\,\mathrm{d}x = \frac{5\pi^2}{24}(3\pi^4 - 28\pi^2 - 24)$$

qui résout le problème posé.

2.3. Polynôme minimal de nombres algébriques. Un nombre complexe α est dit algébrique (sur les nombres rationnels) lorsqu'il est solution d'un polynôme P à coefficients rationnels

$$P(\alpha) = p_0 + \cdots + p_d \alpha^d = 0.$$

Pour pouvoir utiliser l'approche de la section précédente, on se limite au cas de nombres réels.

Étant donnée une approximation numérique (réelle) d'un nombre α qu'on a de bonnes raisons de penser être algébrique, la détermination heuristique de P

peut se voir comme la recherche d'une relation de dépendance linéaire sur des approximations numériques rationnelles de

$$1, \alpha, \dots, \alpha^d.$$

Dès lors qu'on a une borne supérieure sur le degré d de P, on peut donc employer la méthode de la section précédente.

Par exemple, soit à déterminer si un nombre

$$r \simeq 0{,}266252646290196114530247765575844548176501286103 95\dots$$

est algébrique. Par la méthode esquissée, en testant jusqu'à $d = 6$ et pour $N = 10^{28}$, on trouve (à partir de vecteurs lignes de norme supérieure à 10^{20}), le vecteur court

$$v = (-1, 0, 0, 54, 0, 0, -10),$$

de norme inférieure à 55, tous les autres vecteurs calculés étant de norme supérieure à 5000. En poursuivant avec la méthode, on trouve

$$(c_1, \dots, c_7) = (-1, 0, 0, 54, 0, 0, -54),$$

soit

$$P = 54X^6 - 54X^3 + 1.$$

En effet, le nombre r s'avère être une approximation du nombre algébrique

$$\alpha = \sqrt[3]{\frac{1}{2} - \frac{5\sqrt{3}}{18}}.$$

3. Le procédé d'orthogonalisation de Gram–Schmidt

L'algorithme LLL aura fort à voir avec le procédé d'orthogonalisation de Gram–Schmidt qui, partant d'une base $(f_1, \dots, f_n)$ d'un espace vectoriel sur $\mathbb{Q}$, calcule une base orthogonale du même espace vectoriel.

Le contexte de cette méthode est celui de l'espace vectoriel $\mathbb{Q}^n$ muni du produit scalaire euclidien $(a, b) \mapsto (a|b) = a_1b_1 + \cdots + a_nb_n$ qui a servi à définir la norme euclidienne par $\|a\|^2 = (a|a)$. Rappelons que les vecteurs a et b sont dits orthogonaux lorsque leur produit scalaire est nul et que l'orthogonal d'un ensemble $S \subset \mathbb{Q}^n$ est l'ensemble noté $S^\perp$ des vecteurs orthogonaux à tous les éléments de S. On introduit les espaces vectoriels emboîtés $U_i = \mathbb{Q}f_1 \oplus \cdots \oplus \mathbb{Q}f_i$. La projection orthogonale sur U_i est le projecteur linéaire sur U_i parallèlement à $U_i^\perp$. On définit alors f_i^* comme la projection orthogonale de f_i sur $U_i^\perp$ (donc parallèlement à $U_i^{\perp\perp}$, qui n'est autre que U_i).

Les considérations qui précèdent définissent uniquement les f_i^* à partir des f_i et des U_i ; pour le calcul, on détermine les f_i^* pour des i successifs par les formules

$$f_i^* = f_i - \sum_{j=1}^{i-1} \mu_{i,j} f_j^* \qquad \text{où} \qquad \mu_{i,j} = \frac{(f_i|f_j^*)}{(f_j^*|f_j^*)},$$

où $\mu_{i,j}$ est ajusté à l'unique valeur convenable pour avoir orthogonalité entre f_i^* et f_j^* quand $i > j$.

Après avoir posé $\mu_{i,i} = 1$ et $\mu_{i,j} = 0$ quand $i < j$, on obtient une matrice

$$M = (\mu_{i,j}) = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ \mu_{i,j} & & 1 \end{bmatrix}.$$

vérifiant la relation

$$M \begin{bmatrix} f_1^* \\ \vdots \\ f_n^* \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}.$$

En particulier, les espaces vectoriels sur $\mathbb{Q}$ engendrés par les f_i d'une part et par les f_i^* d'autre part sont les mêmes.

Soit maintenant un vecteur non nul f du réseau engendré par les f_i, qui s'écrit donc $f = \lambda_1 f_1 + \cdots + \lambda_n f_n$ pour des entiers λ_i non tous nuls. En passant au carré de la norme et en utilisant le théorème de Pythagore, on a

$$\|f\|^2 = \sum_{i=1}^{n} \lambda_i^2 \|f_i^*\|^2 \geq \|f_k^*\|^2 \geq \left(\min_{j=1}^{n} \|f_j^*\| \right)^2$$

dès lors que $\lambda_k \neq 0$, car alors $\lambda_k^2 \geq 1$.

À ce stade, nous aurions trouvé des vecteurs parmi les plus courts, si ce n'est que les f_i^* sont généralement éléments de l'espace vectoriel engendré par les f_i (avec des coefficients rationnels), mais hors du réseau engendré par les même f_i (avec des coefficients entiers). Il n'en reste pas moins que l'algorithme LLL qui va suivre s'appuie sur l'orthogonalisation de Gram–Schmidt de façon à contrôler des transformations de la base du réseau, car, à un niveau intuitif, plus une base de réseau est « orthogonale », plus elle est à même de contenir un vecteur court. Cette heuristique se justifie par la notion de déterminant d'un réseau : étant donné une base $(f_1, \ldots, f_n)$ d'un réseau, le déterminant de cette famille, c'est-à-dire le déterminant de la matrice $F = (f_{i,j})$ obtenue en plaçant les vecteurs lignes $f_i = (f_{i,1}, \ldots, f_{i,n})$ les uns au dessus des autres, est, au signe près, invariant par changement de base. En effet, soit $(g_1, \ldots, g_n)$ une autre base du même réseau et $G = (g_{i,j})$ la matrice associée, avec des notations évidentes ; il existe alors une matrice U à coefficients entiers, admettant un inverse à coefficients entiers, telle que $F = UG$. En passant aux déterminants, on a que $\det U$ vaut 1 au signe près, donc que F et G ont même déterminant au signe près. Toujours au signe près, ce déterminant n'est autre que le volume du parallèloïde construit en s'appuyant sur les f_i. Les côtés de ce parallèloïde sont d'autant plus courts que ce parallèloïde est orthogonal.

Une base $(f_1, \ldots, f_n)$ d'un réseau est dite *réduite* lorsque les images f_i^* des f_i par le procédé d'orthogonalisation de Gram–Schmidt ont la propriété que $\|f_i^*\|^2 \leq 2\|f_{i+1}^*\|^2$ pour tout i entre 1 et $n-1$. En particulier, f_i^* n'est dans ce cas pas plus de $2^{(i-1)/2}$ fois plus petit que f_1^*, c'est-à-dire que f_1. Il s'ensuit que pour tout vecteur non-nul f du réseau, on a la relation

$$\|f\| \geq \min_{j=1}^{n} \|f_j^*\| \geq \min_{j=1}^{n} 2^{-(j-1)/2} \|f_1\| \geq 2^{-(n-1)/2} \|f_1\|.$$

Autrement dit, le premier élément d'une base réduite est un vecteur court, au sens où pour tout f non nul du réseau

$$\|f_1\| \leq 2^{(n-1)/2} \|f\|.$$

4. L'algorithme LLL

L'algorithme qui suit a été introduit par Lenstra, Lenstra et Lovász en 1982 dans l'objectif de réaliser la factorisation de polynômes d'une variable à coefficients entiers en complexité arithmétique polynomiale. Pour un réel x, on note $\lceil x \rfloor$ l'entier

de plus proche de x (par défaut, l'entier immédiatement inférieur si x est un demi-entier).

ALGORITHME : LLL
ENTRÉE : $f_1, \ldots, f_n$, des vecteurs de $\mathbb{Z}^n$ engendrant un réseau
SORTIE : $g_1, \ldots, g_n$, des vecteurs de $\mathbb{Z}^n$ constituant une base réduite du même réseau
COMPLEXITÉ : $O(n^4 \log A)$ opérations arithmétiques et $O(n^5 \log^2 A)$ opérations binaires, où $A = \max_{i=1}^n \|f_i\|$

1. initialiser g_i à f_i pour chaque i de 1 à n
2. calculer l'orthogonalisation de Gram–Schmidt (g_i^*) de (g_i), ainsi que la matrice M des $\mu_{i,j}$ telle que $M\,{}^t(g_1^*, \ldots, g_n^*) = {}^t(g_1, \ldots, g_n)$
3. pour i à partir de 2, tant que $i \leq n$, faire
   (a) pour j de $i-1$ à 1, remplacer g_i par $g_i - \lceil \mu_{i,j} \rfloor g_j$ et réaliser la même transformation sur les lignes de M
   (b) si $i \geq 2$ et si $\|g_{i-1}^*\|^2 > 2\|g_i^*\|^2$,
      (i) échanger g_{i-1} et g_i
      (ii) recalculer g_{i-1}^* et g_i^* et les lignes correspondantes de M par le procédé de Gram–Schmidt
      (iii) décrémenter i

   sinon incrémenter i
4. renvoyer $(g_1, \ldots, g_n)$

Traitons l'exemple donné par les deux vecteurs $f_1 = (12, 2)$ et $f_2 = (13, 4)$ de $\mathbb{Z}^2$. Après les étapes (1) et (2), on a la relation

$$\begin{bmatrix} 1 & 0 \\ \frac{41}{37} & 1 \end{bmatrix} \begin{bmatrix} 12 & 2 \\ -\frac{11}{37} & \frac{66}{37} \end{bmatrix} = \begin{bmatrix} 12 & 2 \\ 13 & 4 \end{bmatrix}.$$

Comme $\lceil \frac{41}{37} \rfloor = 1$, la transformation (de réduction) de l'étape (3)(a) revient à des multiplications à gauche par la matrice $\left[\begin{smallmatrix} 1 & 0 \\ -1 & 1 \end{smallmatrix}\right]$ et transforme la relation matricielle ci-dessus en

$$\begin{bmatrix} 1 & 0 \\ \frac{4}{37} & 1 \end{bmatrix} \begin{bmatrix} 12 & 2 \\ -\frac{11}{37} & \frac{66}{37} \end{bmatrix} = \begin{bmatrix} 12 & 2 \\ 1 & 2 \end{bmatrix}.$$

Les normes à considérer vérifient $\|g_1^*\|^2 = 4 \times 37 > 2\|g_2^*\|^2 = 2 \times 11^2/37$, ce qui provoque un échange en (3)(b)(i) avec recalcul en (3)(b)(ii) par le procédé de Gram–Schmidt, et fournit la nouvelle relation matricielle

$$\begin{bmatrix} 1 & 0 \\ \frac{16}{5} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ \frac{44}{5} & -\frac{22}{5} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 12 & 2 \end{bmatrix}.$$

Comme $\lceil \frac{16}{5} \rfloor = 3$, une nouvelle transformation (de réduction) à l'étape (3)(a) revient à des multiplications à gauche par la matrice $\left[\begin{smallmatrix} 1 & 0 \\ -3 & 1 \end{smallmatrix}\right]$ et transforme la relation matricielle ci-dessus en

$$\begin{bmatrix} 1 & 0 \\ \frac{1}{5} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ \frac{44}{5} & -\frac{22}{5} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 9 & -4 \end{bmatrix}.$$

Comme $\|g_1^*\|^2 = 5 \leq 2\|g_2^*\| = 2 \times 11^2 \times 4$, l'algorithme termine en renvoyant $g_1 = (1, 2)$ et $g_2 = (9, -4)$.

5. Preuve de l'algorithme LLL

Nous développons maintenant des invariants et des variants de l'algorithme qui permettent de montrer la correction et la terminaison de l'algorithme, puis une borne polynomiale sur sa complexité arithmétique.

5.1. Correction. À l'entrée de la boucle (3), la base de réseau (g_i) et son orthogonalisée (g_i^*) par la méthode de Gram–Schmidt vérifient la relation $M\,{}^t(g_1^*,\dots,g_n^*) = {}^t(g_1,\dots,g_n)$. Observons d'abord que cet invariant reste respecté à chaque sortie du corps de boucle. Il est immédiat que la relation matricielle reste vérifiée après la transformation de l'étape (3)(a) ; montrons que, pour λ quelconque et tout $j<i$, le remplacement de g_i par $g_i-\lambda g_j$ et de chaque $\mu_{i,k}$ par $\mu_{i,k}-\lambda\mu_{j,k}$ quand $1\le k\le j$ fait de la nouvelle famille des g_i^* l'orthogonalisée de la nouvelle famille des g_i. (Dans l'algorithme, on choisit $\lambda=\lceil\mu_{i,j}\rfloor$.) Puisque la famille d'espaces emboîtés $\mathbb{Q}g_1\oplus\cdots\oplus\mathbb{Q}g_k$, d'une part, et celle des $\mathbb{Q}g_1^*\oplus\cdots\oplus\mathbb{Q}g_k^*$, d'autre part, n'ont pas changé, il suffit pour cela de vérifier que le nouveau g_i^* vérifie bien sa définition. En termes des anciennes valeurs, ceci se résume, pour $j<i$, à vérifier la relation

$$g_i^* = g_i - \sum_{k=1}^{i-1}\mu_{i,k}g_k^* = \big(g_i-\lambda g_j\big) - \sum_{k=1}^{i-1}\big(\mu_{i,k}-\lambda\mu_{j,k}\big)g_k^* + \lambda\bigg(g_j-\sum_{k=1}^{j}\mu_{j,k}g_k^*\bigg).$$

La dernière parenthèse étant nulle, l'invariant recherché reste préservé par la transformation (3)(a). Quand à l'étape (3)(b), on observe de nouveau que la famille d'espaces emboîtés $\mathbb{Q}g_1\oplus\cdots\oplus\mathbb{Q}g_k$, d'une part, et celle des $\mathbb{Q}g_1^*\oplus\cdots\oplus\mathbb{Q}g_k^*$, d'autre part, ne sont pas perturbées par l'échange (3)(b)(i), sauf peut-être pour $k=i$ ou $k=i-1$. C'est pourquoi l'algorithme recalcule explicitement g_{i-1}^* puis g_i^* par les formule de Gram–Schmidt pour restaurer l'invariant.

Un second invariant respecté par la même boucle (3) est la suite d'inégalités $\|g_1^*\|^2\le 2\|g_2^*\|^2$, …, $\|g_{i-2}^*\|^2\le 2\|g_{i-1}^*\|^2$. Cet invariant se réduit en effet à une absence de contraintes à l'entrée de la boucle (3), pour $i=2$. Puis, la stratégie de l'étape (3)(b) est de simplement incrémenter i si la suite d'inégalité peut se prolonger par $\|g_{i-1}^*\|^2\le 2\|g_i^*\|^2$, ou au contraire de décrémenter i lorsque g_{i-1}^* est modifié.

Ainsi, si une exécution de l'algorithme sort de la boucle (3), la base (g_i) est réduite, ce qui prouve la correction de l'algorithme.

5.2. Terminaison. Pour montrer que l'algorithme termine, on va montrer que l'échange et la mise à jour (3)(b)(i–ii), seule étape qui modifie les g_i^*, les réduit en norme, et réduit dans le même temps une grandeur entière positive associée aux carrés des normes $\|g_i\|^2$. Le nombre d'échange ne pourra donc être que fini. Un invariant entre nombre d'incrémentations et de décrémentations dans l'étape (3)(b) montre alors la terminaison de la boucle (3), et donc de l'algorithme.

Pour un passage dans l'étape (3)(b) avec $i\ge 2$ et $\|g_{i-1}^*\|^2>2\|g_i^*\|^2$, notons (h_k) la base du réseau obtenue après l'échange (3)(b)(i) et (h_k^*) l'orthogonalisée de Gram–Schmidt obtenue après le recalcul (3)(b)(ii). On a $h_i=g_{i-1}$, $h_{i-1}=g_i$, et $h_k=g_k$ sinon. De même, on a les égalités entre espaces vectoriels $\mathbb{Q}g_1\oplus\cdots\oplus\mathbb{Q}g_k=\mathbb{Q}h_1\oplus\cdots\oplus\mathbb{Q}h_k$, sauf pour $k=i-1$. Ainsi, les vecteurs orthogonalisés h_k^* et g_k^* sont égaux, sauf peut-être pour $k=i$ et $k=i-1$. Posons $U=\mathbb{Q}g_1\oplus\cdots\oplus\mathbb{Q}g_{i-2}$. Par la définition du procédé d'orthogonalisation de Gram–Schmidt, on a d'abord $g_i=h_{i-1}=h_{i-1}^*+u_1$ pour $u_1\in U$, et aussi $g_i=g_i^*+\mu_{i,i-1}g_{i-1}^*+u_2$ pour $u_2\in U$.

Comme u_1 et u_2 sont les projetés orthogonaux du même vecteur orthogonalement à U, ils sont égaux, de même que h^*_{i-1} et $g^*_i + \mu_{i,i-1} g^*_{i-1}$. En passant aux carrés des normes, on obtient $\|h^*_{i-1}\|^2 = \|g^*_i\|^2 + \mu^2_{i,i-1}\|g^*_{i-1}\|^2 < \|g^*_{i-1}\|^2/2 + (1/2)^2\|g^*_{i-1}\|^2$, où on a utilisé l'hypothèse que le test en (3)(b) a été positif et le fait que l'étape (3)(a) a forcé la relation $\mu_{i,i-1} \leq 1/2$. En résumé, $\|h^*_{i-1}\|^2 < 3\|g^*_{i-1}\|^2/4$. Par la même méthode, en considérant des projections orthogonales de g_{i-1} sur $U' = U \oplus \mathbb{Q}g_{i-1}$, on montre la relation $\|h^*_i\| \leq \|g^*_i\|$.

Notons G_i la matrice obtenue en superposant les i premiers vecteurs g_1, …, g_i, et G^*_i celle obtenue à partir des g^*_k correspondants. La matrice de Gram des vecteurs g_1, …, g_i est la matrice de taille $i \times i$ dont l'entrée (u, v) est le produit scalaire $(g_u|g_v)$. C'est donc une matrice de $\mathbb{Z}^{i\times i}$, qui s'exprime aussi $G_i\, {}^tG_i$. Puisque le bloc M_i de taille $i \times i$ en haut à gauche dans la matrice M est encore trigonal avec des 1 sur la diagonale, donc de déterminant 1, on a

$$d_i = \det(M_i G^*_i\, {}^tG^*_i\, {}^tM_i) = \det(M_i)\det(G^*_i\, {}^tG^*_i)\det({}^tM_i) = \|g^*_1\|^2 \dots \|g^*_i\|^2.$$

Ce nombre est donc un entier strictement positif.

Le variant qui va s'avérer adéquat est le produit $D = d_1 \dots d_{n-1}$, de nouveau un entier strictement positif. Ce nombre est divisé par au moins 4/3 à chaque échange aux étapes (3)(b)(i–ii), tout en restant entier strictement positif. Il ne peut donc y avoir qu'un nombre fini d'échanges lors d'une exécution de l'algorithme LLL. Considérons les nombres e d'échanges en (3)(b)(i) et v de passages en (3)(b) avec un test négatif effectués depuis le début d'une exécution de l'algorithme. En observant ces valeurs à chaque entrée de la boucle (3), on a l'invariant $i = 2 - e + v$. Chaque passage dans la boucle incrémente l'un ou l'autre de ces deux nombres. Mais comme le nombre e ne peut croître indéfiniment, le nombre v doit ultimement ne plus cesser de croître, et i avec, ce jusqu'à la valeur $i = n + 1$ qui provoque la fin de la boucle (3). L'algorithme termine donc.

5.3. Complexité arithmétique. Au début de l'algorithme, le nombre D qui vient d'être introduit pour montrer la terminaison vaut

$$D_0 = \|g^*_1\|^{2(n-1)}\|g^*_2\|^{2(n-2)} \dots \|g^*_{n-1}\|^2.$$

Mais chaque g^*_i étant alors une certaine projection de l'entrée f_i, on a

$$D_0 \leq \|f_1\|^{2(n-1)}\|f_2\|^{2(n-2)} \dots \|f_{n-1}\|^2 \leq A^{n(n-1)}.$$

Après e échanges en (3)(b)(i), on a l'encadrement $1 \leq D \leq (3/4)^e D_0$, d'où la borne supérieure $n(n-1)\log_{4/3} A$ sur e.

Notons e_{final} et v_{final} les valeurs finales de e et v. On vient de prouver la relation $e_{\text{final}} = O(n^2 \log A)$; par ailleurs l'invariant sur i donne $n+1 = 2 - e_{\text{final}} + v_{\text{final}}$, d'où $v_{\text{final}} = O(n^2 \log A)$. Comme l'orthogonalisation initiale se fait en $O(n^3)$ opérations arithmétiques et que chacune des étapes (3)(a) et (3)(b)(i–ii) se fait en $O(n^2)$ opérations arithmétiques, la complexité arithmétique totale de l'algorithme LLL est

$$O(n^3) + (e_{\text{final}} + v_{\text{final}})O(n^2) = O(n^4 \log A).$$

La borne sur la complexité en bits est réellement plus technique et n'est pas présentée ici. L'idée de la preuve est que les entiers utilisés dans l'algorithme ne dépassent pas la taille $O(n \log A)$. La borne annoncée n'est ensuite que le produit de la borne sur la complexité arithmétique par cette borne sur les entiers utilisés.

COURS 29

Factorisation des polynômes

1. Introduction

Le but de la dernière partie du cours est de montrer comment factoriser des polynômes dans $\mathbb{Z}[X]$. Pour factoriser un polynôme F à coefficients entiers, la majorité des algorithmes actuels procèdent ainsi :

1.: Factorisation de $F \mod p$ dans $\mathbb{Z}/p\mathbb{Z}[X]$, p étant un nombre premier.

2.: Remontée de Hensel, pour passer à une factorisation modulo une puissance suffisante de p.

3.: Recombinaison (par recherche exhaustive ou réduction de réseau) pour retrouver les "vrais" facteurs, c'est-à-dire ceux sur $\mathbb{Z}$.

Le mieux est de donner un exemple : soit $F = x^4 - 2x^3 + 9x^2 - 8x + 20$. Modulo $p = 5$, on a

$$F \equiv x(x+1)(x+3)(x+4) \mod 5.$$

Ensuite, on obtient successivement

- $F \equiv (x+10)(x+11)(x+13)(x+14) \mod 5^2$
- $F \equiv (x+260)(x+261)(x+363)(x+364) \mod 5^4$
- $F \equiv (x+220885)(x+220886)(x+169738)(x+169739) \mod 5^8$.

On renormalise tous les coefficients entre -195312 et 195312 (ici, $195312 = \frac{5^8}{2}$) :

$$\begin{aligned} F &\equiv (x-169740)(x-169739)(x+169738)(x+169739) \mod 5^8 \\ &\equiv f_1 f_2 f_3 f_4 \mod 5^8. \end{aligned}$$

En effectuant toutes les combinaisons possibles, on constate que

$$f_2 f_4 \equiv x^2 + 4 \mod 5^8 \quad \text{et} \quad f_1 f_3 \equiv x^2 - 2x + 5 \mod 5^8.$$

Voilà la factorisation.

Dans le cours présent, on s'intéresse à la première partie, la factorisation dans un corps fini. Il existe quantité d'algorithmes pour cette question, probabilistes ou déterministes. Après quelques rappels / compléments sur les corps finis, on présente l'algorithme de Berlekamp, algorithme "classique" (et déterministe) pour cette question.

2. Corps finis, quelques rappels

Préliminaires : groupes commutatifs finis, arithmétique. Soit G un groupe commutatif fini, noté multiplicativement (dans la suite, G sera le groupe $K-\{0\}$, K étant un corps fini). On appelle *ordre* d'un élément a de G le plus petit entier positif (non nul) ω tel que $a^\omega = 1$. On utilisera dans la suite les résultats suivants :

LEMME 1. *L'ordre de tout élément divise le cardinal $|G|$ de G, et donc $a^{|G|} = 1$ pour tout a de G.*

DÉMONSTRATION. L'ensemble des puissances de a est un sous-groupe de G, de cardinal ω. On utilise alors le théorème de Lagrange, qui dit que le cardinal d'un sous-groupe de G divise $|G|$. □

Le résultat suivant n'est pas bien difficile non plus ; n'importe quel cours sur la théorie des groupes se doit d'en donner une preuve.

LEMME 2. *Soit Ω le ppcm des ordres des éléments de G. Alors, il existe un élément d'ordre Ω ; en particulier, Ω divise $|G|$.*

Enfin, on utilisera le résultat arithmétique suivant.

LEMME 3. *Soit p, m, n trois entiers positifs, avec $p > 1$. Si $p^m - 1$ divise $p^n - 1$ alors m divise n.*

DÉMONSTRATION. L'hypothèse dit que $p^n \equiv 1$ modulo $p^m - 1$. On pose $n = qm + r$, avec $0 \leq r < m$. Modulo $p^m - 1$, on a donc $p^r \equiv p^n$, donc $p^r \equiv 1$. Puisque $r < m$, on a donc $p^r = 1$ et $r = 0$. □

Corps finis. L'exemple le plus simple de corps fini est un quotient de la forme $\mathbb{Z}/p\mathbb{Z}$, où p est un nombre premier, et on le note à l'anglaise $\mathbb{F}_p$.

Soit ensuite P un polynôme irréductible de $\mathbb{F}_p[X]$, de degré d. Alors, $K = \mathbb{F}_p[X]/P$ est également un corps ; puisque $1, X, \ldots, X^{d-1}$ forment une base de K comme espace vectoriel sur $\mathbb{F}_p$, le cardinal de K est $q = p^d$.

PROPOSITION 1. *Soit K un corps fini à q éléments. Pout tout a dans $K-\{0\}$, on a $a^{q-1} = 1$; par suite, on a*

$$X^q - X = \prod_{a \in K} (X - a).$$

DÉMONSTRATION. Soit alors a non nul dans K. Le corps K ayant cardinal q, $K-\{0\}$ est un groupe fini de cardinal $q-1$, donc $a^{q-1} = 1$ (Lemme 1). On en déduit que $a^q = a$ pout tout a de K, nul ou pas. Ceci montre que le polynôme $X^q - X$ s'annule sur les q éléments de K, donc $\prod_{a \in K}(X - a)$ divise $X^q - X$. Comme les deux polynômes sont unitaires et de même degré, ils sont égaux. □

COROLLAIRE 1. *Pour tout $i = 1, \ldots, p-1$, le coefficient binomial $\binom{p}{i}$ vaut zéro modulo p.*

DÉMONSTRATION. Rappelons que ce coefficient est le coefficient de X^i dans $(X+1)^p$. La proposition précédente entraîne les égalités

$$\begin{aligned} X^p - X &= \textstyle\prod_{a \in \mathbb{F}_p}(X - a) \\ &= \textstyle\prod_{a \in \mathbb{F}_p}(X - a + 1) \\ &= \textstyle\prod_{a \in \mathbb{F}_p}((X+1) - a) \\ &= (X+1)^p - (X+1). \end{aligned}$$

On en déduit que $(X+1)^p = X^p + 1$, qui est ce que l'on voulait montrer. □

Le corollaire suivant est, d'une manière ou d'une autre, à la base de la plupart des résultats "arithmétiques" sur les corps finis.

COROLLAIRE 2. *Soit P un polynôme quelconque dans $\mathbb{F}_p[X]$ et soit K le quotient $\mathbb{F}_p[X]/P$ (qui n'est pas nécessairement un corps). L'application*

$$\phi : \begin{array}{ccc} K & \to & K \\ a & \mapsto & a^p \end{array}$$

est $\mathbb{F}_p$-linéaire.

DÉMONSTRATION. Soient λ dans $\mathbb{F}_p$ et a, b dans K. L'égalité $\phi(\lambda a) = \lambda\phi(a)$ est une conséquence de la Proposition 1, appliquée à $\mathbb{F}_p$. L'égalité $\phi(a+b) = \phi(a)+\phi(b)$ est une conséquence du corollaire précédent (développer le produit $(a+b)^p$). □

Voyons quelques conséquences de ce corollaire.

PROPOSITION 2. *Pour tous $q_0, \ldots, q_n$ dans $\mathbb{F}_p$, on a*

$$\left(\sum_i q_i X^i\right)^p = \sum_i q_i X^{ip}.$$

DÉMONSTRATION. Conséquence facile de la Proposition 1 et du Corollaire 1. □

Le résultat suivant n'est pas utilisé dans l'algorithme de Berlekamp, mais on s'en servira ultérieurement (phase "Distinct Degree Factorization" d'algorithmes probabilistes).

PROPOSITION 3. *Pour $n > 0$, le polynôme $X^{p^n} - X$ est le produit des polynômes unitaires et irréductibles de $\mathbb{F}_p[X]$ dont le degré divise n.*

DÉMONSTRATION. Soit P un polynôme irréductible de degré m, avec $m \mid n$, et soit K le quotient $\mathbb{F}_p[X]/P$; on se rappelle que $|K| = p^m$. Soit ensuite x la classe de X dans K ; la Proposition 1 appliquée à x montre que $x^{p^m-1} = 1$ donc $x^{p^m} = x$, d'où on déduit $x^{p^n} = x$. Donc P divise $X^{p^n} - X$.

Réciproquement, soit P un polynôme irréductible de degré m tel que P divise $X^{p^n} - X$; on veut montrer que m divise n. Soit K le quotient $K = \mathbb{F}_p[X]/P$ et soit x l'image de X dans K. Puisque $P(x) = 0$, $x^{p^n} - x$ est nul. On en tire que pour tous λ_i dans $\mathbb{F}_p$, on a les égalités :

$$\begin{array}{rcl} (\sum_i \lambda_i x^i)^{p^n} & = & \sum_i \lambda_i (x^i)^{p^n} \\ & = & \sum_i \lambda_i (x^{p^n})^i \\ & = & \sum_i \lambda_i x^i, \end{array}$$

la première égalité s'obtenant en appliquant n fois le Corollaire 2. Donc tout élément de K est racine de $X^{p^n} - X$, de sorte que tout élément non nul de K est racine de $X^{p^n-1} - 1$.

Soit maintenant Ω l'exposant du groupe $K - \{0\}$; d'après le Lemme 2, Ω divise $|K - \{0\}| = p^m - 1$. Par ailleurs, par définition, tout élément non nul de K annule $X^\Omega - 1$, donc Ω est plus grand que $p^m - 1$: Il s'en suit que $\Omega = p^m - 1$. Le Lemme 2 montre alors qu'il existe un élément α d'ordre $p^m - 1$ dans $K - \{0\}$.

On a vu que $\alpha^{p^n-1} = \alpha$; on en déduit que $p^m - 1$ divise $p^n - 1$. Le Lemme 3 permet de conclure que m divise n.

À ce stade, on sait donc que les facteurs irréductible de $X^{p^n}-X$ sont exactement les polynômes irréductibles de degré divisant n. Il reste à montrer qu'il n'y a pas de facteur multiple. Pour cela, il suffit de constater que la dérivée de $X^{p^n}-X$ vaut -1, et en particulier que son pgcd avec $X^{p^n}-X$ vaut 1. On utilise ensuite la Proposition 4. □

3. Partie sans carré et décomposition sans carré

Une première étape des algorithmes de factorisation est souvent de se ramener à un polynôme sans facteurs multiples, que l'on appellera ici polynômes sans carré.

Pour cette question, on peut distinguer deux niveaux de raffinement, illustrés sur un exemple : soit par exemple

$$P=(X+1)(X+2)(X+3)^2(X+4)^2(X+5)^3(X+6)^3,$$

sur un corps dans lequel $1,2,3,4,5$ et 6 sont tous distincts. La *partie sans carré* de P sera le produit où on a "supprimé" toutes les multiplicités, c'est-à-dire

$$(X+1)(X+2)(X+3)(X+4)(X+5)(X+6).$$

La *décomposition sans carré* va plus loin, et sépare les facteurs selon leur multiplicité. Ici, elle donne donc :

$$[(X+1)(X+2),1],\quad [(X+3)(X+4),2],\quad \text{et}\quad [(X+5)(X+6),3].$$

On voit bien que le calcul de décomposition sans carré est l'amorce de la factorisation ; en outre, c'est un calcul sensiblement plus facile que celui de la factorisation.

En effet, on va voir plus bas un algorithme de calcul de la partie sans carré, de complexité essentiellement linéaire en le degré, qui repose uniquement sur des calculs de pgcd bien choisis.

Il n'est pas difficile d'en déduire un algorithme pour la décomposition sans carré : on divise P par sa partie sans carré, on calcule la partie sans carré du résultat, etc … Ce faisant, on n 'est plus en complexité linéaire, ce qui n'est pas très bon ; il est possible de faire mieux, et de rester essentiellement linéaire, en utilisant un algorithme dû à Yun, que je ne donnerai pas ici faute de temps.

Polynômes sans carrés. Soit P un polynôme de $\mathbb{F}_p[X]$. On dit que P est *sans carré* s'il n'existe pas de polynôme Q de $\mathbb{F}_p[X]$ tel que Q^2 divise P.

Lemme 4. *Soit P dans $\mathbb{F}_p[X]$. La dérivée P' vaut 0 si et seulement si P s'écrit sous la forme Q^p.*

Démonstration. Si $P=Q^p$, on a bien $P'=pQ'Q^{p-1}=0$. Réciproquement, on voit que si $P'=0$, P est de la forme $P=\sum_i q_i X^{pi}$, qui est une puissance p-ième d'après la Proposition 2. □

Proposition 4. *P est sans carré si et seulement si* $\operatorname{pgcd}(P,P')=1$.

Démonstration. Écrivons la factorisation en irréductibles de P :

$$P=P_1^{a_1}\cdots P_s^{a_s},$$

où tous les a_i sont ≥ 1. En dérivant, on obtient

$$P'=\sum_i a_i P_i' \frac{P}{P_i}.$$

Si P admet un facteur carré, il existe au moins un $a_i \geq 2$, pour lequel P_i divise $\frac{P}{P_i}$. Alors, P_i divisant tous les autres $\frac{P}{P_j}$, il divise P', de sorte que $\mathrm{pgcd}(P, P')$ est différent de 1.

Pour la réciproque, on note pour commencer que les P_i étant irréductibles, leurs dérivées ne sont pas nulles (proposition précédente). Supposons que alors $\mathrm{pgcd}(P, P')$ est différent de 1. Alors, il existe un P_i qui divise P'. Puisqu'il divise tous les autres $\frac{P}{P_j}$, il divise $a_i P_i' \frac{P}{P_i}$.

Puisque P_i' est non nul, P_i ne peut pas le diviser (raison de degré), donc P_i divise $a_i \frac{P}{P_i}$. Si a_i est non nul, P_i divise $\frac{P}{P_i}$, donc P_i^2 divise P, et on a fini. Si a_i est nul (dans $\mathbb{F}_p$), cela signifie que p divise a_i dans $\mathbb{N}$, donc a_i ne vaut pas 1 ; à nouveau, on a fini. □

Décomposition et partie sans carré. Soit toujours P dans $\mathbb{F}_p[X]$, et écrivons sa factorisation en irréductibles :

$$P = P_1^{a_1} \cdots P_s^{a_s}.$$

On regroupe alors les P_i apparaissant avec des exposants similaires. On note ainsi Q_1 le produit des P_i apparaissant à la puissance 1, Q_2 le produit des P_i apparaissant à la puissance 2, … Q_s le produit des P_i apparaissant à la puissance s. Il est alors possible de réécrire P sous la forme :

$$P = Q_1 Q_2^2 \cdots Q_s^s. \tag{1}$$

Attention il est possible que certains Q_i soient égaux à 1. Par ailleurs, tous les Q_i sont sans carré, et ils sont tous premiers entre eux deux à deux.

On appelle *décomposition sans carré* de P la donnée des polynômes $Q_1, \ldots, Q_s$, et *partie sans carré* de P le produit $Q_1 \cdots Q_s$; il s'agit manifestement d'un polynôme sans carré.

Proposition 5. *Si p est plus grand que s, alors la partie sans carré de P est $P/\mathrm{pgcd}(P, P')$.*

Démonstration. En dérivant $P = Q_1 Q_2^2 \cdots Q_s^s$, on obtient

$$P' = \left(Q_2 \cdots Q_s^{s-1}\right)\left(Q_1' Q_2 \cdots Q_s + \cdots + s Q_1 \cdots Q_{s-1} Q_s'\right).$$

Les Q_i' ne sont pas nuls (car les Q_i sont sans carré). Puisque $1, \ldots, s$ ne sont pas nuls modulo p, le membre de droite est donc premier avec tous les Q_i, donc avec P. Le pgcd de P et P' est donc $Q_2 \cdots Q_s^{s-1}$, ce qui donne le résultat. □

Si p est plus petit que (ou égal à) s, il est possible que ce résultat soit pris en défaut : dans $\mathbb{F}_2[X]$, on a $X^2 + 1 = (X+1)^2$ et $(X^2+1)' = 0$. Dans ce cas, $P/\mathrm{pgcd}(P, P')$ vaut 1, alors que la partie sans carré est $X + 1$.

La proposition suivante précise ce type de comportement.

Proposition 6. *Le quotient $P/\mathrm{pgcd}(P, P')$ vaut*

$$\prod_{i \text{ non multiple de } p} Q_i.$$

Démonstration. En reprenant la preuve précédente, on voit que le pgcd de P et P' vaut $(Q_2 \cdots Q_s^{s-1})$ multiplié par

$$\prod_{i \text{ multiple de } p} Q_i.$$

□

Il reste cependant possible d'obtenir les parties manquantes par calcul de pgcd.

Proposition 7. *Soient n le degré de P, $U = \mathrm{pgcd}(P, P')$ et $V = P/U$. Soit ensuite $W = \mathrm{pgcd}(U, V^n)$. Alors U/W vaut*

$$\prod_{i \text{ multiple de } p} Q_i^i.$$

Démonstration. D'après la proposition précédente, V^n vaut

$$\prod_{i \text{ non multiple de } p} Q_i^n,$$

et U vaut

$$(Q_2 \cdots Q_s^{s-1}) \prod_{i \text{ multiple de } p} Q_i.$$

Puisque n est une borne supérieure sur les multiplicités s, on en déduit que W vaut

$$\prod_{i \text{ non multiple de } p} Q_i^{i-1},$$

d'où le résultat. □

Remarque : le W de la proposition est une puissance p-ième, d'après la Proposition 2.

Il est enfin possible de déduire un algorithme de calcul de la partie sans carré de toutes ces considérations.

Partie sans carré d'un polynôme.:

Entrée :: P dans $\mathbb{F}_p[X]$.

Sortie :: La partie sans carré de P.

1. Calculer $U = \mathrm{pgcd}(P, P')$, $V = P/U$ et $W = \mathrm{pgcd}(U, V^n)$;
2. Calculer la racine p-ième W_0 de W ;
3. Calculer récursivement la partie sans carré S de W_0 ;
4. Renvoyer VS.

Proposition 8. *Cet algorithme calcule la partie sans carré de P en $O(\mathsf{M}(n)\log(n))$ opérations, où M est une fonction de multiplication.*

Démonstration. La correction de cet algorithme provient des propositions précédentes, seule la complexité est plus délicate à analyser. Les calculs de pgcd se font en $O(\mathsf{M}(n)\log(n))$ opérations ; remarquer qu'il suffit de calculer V^n modulo U, et que cela se fait en $O(\log(n))$ multiplications modulo U par exponentiation binaire.

Ensuite, W_0 s'obtient sans calcul (voir la Proposition 2) ; son degré est au plus égal à n/p. On en déduit que le temps de calcul $T(n)$ satisfait la récurrence

$$T(n) \leq T\left(\frac{n}{p}\right) + C\ \mathsf{M}(n)(\log(n)),$$

C étant une constante. On en déduit le résultat. □

4. Algorithme de Berlekamp

On va exposer ici un algorithme historique de factorisation, l'algorithme de Berlekamp. Son analyse de complexité mène au résultant suivant.

PROPOSITION 9. *Soit P un polynôme sans carré de $\mathbb{F}_p[X]$. On peut factoriser P par un algorithme déterministe dont la complexité est de*

$$O(n^\omega \ + \ n\,\mathsf{M}(n)\log(n)\,p)$$

opérations de $\mathbb{F}_p$, où ω est l'exposant de l'algèbre linéaire sur $\mathbb{F}_p$ et M une fonction de multiplication.

Quelques remarques :

- L'hypothèse que P est sans carré n'est pas une limitation ici : même en utilisant la version "naïve" de décomposition sans carré suggérée dans la section précédente, on reste dans une complexité inférieure au n^ω qui apparaît ici.
- La complexité de cet algorithme est linéaire en p : cela le rend rapidement peu pratique, dès lors que p atteint quelques milliers ou quelques millions. À l'heure actuelle, on ne connaît pas d'algorithme de factorisation déterministe de complexité logarithmique en p (borne qu'atteignent les algorithmes probabilistes).

Rappel : restes chinois. Soient $P_1, \ldots, P_r$ des polynômes définis sur un corps k quelconque. Supposons que $P_1, \ldots, P_r$ sont tous premiers entre eux deux-à-deux, et soit P leur produit. Soit ϕ l'application "restes modulo $P_1, \ldots, P_r$" :

$$\phi : \begin{array}{ccc} k[X] & \to & k[X]/P_1 \times \cdots \times k[X]/P_r \\ a & \mapsto & (a \mod P_1, \ldots, a \mod P_r). \end{array}$$

Les P_i étant premiers entre eux deux à deux, on sait que :

- ϕ est surjective,
- son noyau est l'ensemble des multiples de P.

Autrement dit, ϕ induit un isomorphisme

$$k[X]/P \equiv k[X]/P_1 \times \cdots \times k[X]/P_r.$$

L'application de Berlekamp. Soit P un polynôme sans carré. En vertu du Corollaire 2, l'application

$$\phi : \begin{array}{ccc} \mathbb{F}_p[X]/P & \to & \mathbb{F}_p[X]/P \\ a & \mapsto & a^p - a \end{array}$$

est une application $\mathbb{F}_p$-linéaire. L'étude de son noyau va permettre de factoriser P, à partir de la proposition suivante. On note $P_1, \ldots, P_r$ les facteurs (inconnus) de P :

$$P = P_1 \cdots P_r.$$

PROPOSITION 10. *Soit $a \in \mathbb{F}_p[X]$, non constant, de degré plus petit que P, et tel que $\phi(a \mod P) = 0$. Alors :*

- *pour $i = 1, \dots, r$, $a \mod P_i$ est une constante ;*
- *ces constantes ne sont pas toutes égales.*

DÉMONSTRATION. Puisque tous les P_i sont distincts et irréductibles, on a d'après le paragraphe précédent un isomorphisme

$$\mathbb{F}_p[X]/P \equiv \mathbb{F}_p[X]/P_1 \times \cdots \times \mathbb{F}_p[X]/P_r,$$

donné par

$$a \mod P \mapsto (a \mod P_1, \dots, a \mod P_r).$$

Les P_i étant irréductibles, chaque $\mathbb{F}_p[X]/P_i$ est un corps. Dans la représentation "produit de corps", l'application ϕ s'écrit bien sûr :

$$(a_1, \dots, a_r) \mapsto (a_1^p - a_1 \mod P_1, \dots, a_r^p - a_r \mod P_r).$$

Dans cette représentation, on a alors la caractérisation suivante du noyau de ϕ.

LEMME 5. *Le r-uplet $(a_1, \dots, a_r)$ est dans le noyau de ϕ si et seulement si $a_i \in \mathbb{F}_p \subset \mathbb{F}_p[X]/P_i$ pour tout i.*

DÉMONSTRATION. On sait (Proposition 1) que le polynôme $X^p - X$ se factorise dans $\mathbb{F}_p$, et donc dans $\mathbb{F}_p[X]/P_i$, sous la forme $\prod_{a \in \mathbb{F}_p}(X - a)$. Puisque $\mathbb{F}_p[X]/P_i$ est un corps, $a_i \in \mathbb{F}_p[X]/P_i$ est une racine de $X^p - X$ si et seulement s'il annule un des facteurs $X - a$, donc si et seulement si $a_i \in \mathbb{F}_p$. □

Ceci prouve le premier point de la proposition. Quant au second, remarquons que si tous les $a_i = a \mod P_i$ sont les mêmes, on a nécessairement $a = a_i$ (pout tout i), car on a supposé a de degré plus petit que P. En particulier, a devrait être constant, ce qui est contraire à nos hypothèses. □

La clé de l'algorithme de Berlekamp est la proposition suivante.

PROPOSITION 11. *Soit a comme dans la proposition précédente. Il existe b dans $\mathbb{F}_p$ tel que $\operatorname{pgcd}(P, a - b)$ est un diviseur non trivial de P.*

DÉMONSTRATION. Pour b quelconque dans $\mathbb{F}_p$, on a l'égalité

$$\operatorname{pgcd}(P, a - b) = \prod_i P_i,$$

où le produit est pris sur tous les P_i tels que P_i divise $a - b$, c'est-à-dire tels que $a \mod P_i = b$. On sait qu'il existe au moins deux valeurs distinctes pour les $a \mod P_i$. Chacune d'entre elles donnant un diviseur de P, ces diviseurs sont stricts. □

D'où l'algorithme.

Calcul d'un facteur.:

Entrée :: P dans $\mathbb{F}_p[X]$, sans carré.

Sortie :: Un facteur de P.

1. Calculer la matrice de l'application ϕ, et un vecteur a dans le noyau.
2. Calculer tous les $\operatorname{pgcd}(P, a - b)$, jusqu'à trouver un facteur non trivial.

PROPOSITION 12. *L'algorithme précédent calcule un facteur non trivial de P en*

$$O(n^\omega + \mathsf{M}(n)\log(n)\,p)$$

opérations de $\mathbb{F}_p$.

DÉMONSTRATION. Le premier pas est de calculer la matrice de ϕ. On commence par calculer $X^p \bmod P$: cela demande $O(\log(p))$ opérations modulo P, soit $O(\mathsf{M}(n)\log(p))$ opérations dans $\mathbb{F}_p$. Ensuite, il faut calculer $(X^2)^p = (X^p)^2, (X^3)^p = (X^p)^3, \ldots$, ce qui se fait en n multplications mod P, soit $O(n\mathsf{M}(n))$ opérations en tout.

L'algèbre linéaire coûte n^ω. Une fois obtenu un vecteur du noyau, chaque essai de pgcd coûte $O(\mathsf{M}(n)\log(n))$ opérations, et il faut potentiellement en faire p. En mettant tout bout-à-bout, on obtient le résultat ci-dessus. □

Pour trouver la factorisation complète, il suffit de réitérer le procédé ce qui entraîne *a priori* un surcoût de n. En fait, il n'y a pas besoin de refaire de l'algèbre linéaire, de sorte que le coût total de la factorisation est

$$O(n^\omega + n\mathsf{M}(n)\log(n)\,p)$$

opérations de $\mathbb{F}_p$.

Exercices

EXERCICE 1 (Factorisation probabiliste en temps polynomial). Le but de cet exercice est de donner un algorithme (probabiliste) pour la factorisation dans $\mathbb{F}_p[X]$, dont la complexité soit polynomiale en $(n, \log(p))$, où n est le degré du polynôme à factoriser. Ici, p est un nombre premier différent de 2 ; les complexités des algorithmes seront énoncées en nombre d'opérations dans $\mathbb{F}_p$.

L'algorithme proposé fonctionne en plusieurs étapes. On admettra le fait suivant (qui généralise un résultat montré en cours) : pour tout $d \geq 0$, le polynôme

$$X^{p^d} - X$$

est le produit de tous les polynômes irréductibles et unitaires de $\mathbb{F}_p[X]$, dont le degré divise d.

Question 1: Donner un algorithme, qui, étant donné P dans $\mathbb{F}_p[X]$, calcule le produit de tous les polynômes unitaires de degré 1 qui divisent P. Quelle est sa complexité ?

Question 2: Donner un algorithme, qui, étant donné P dans $\mathbb{F}_p[X]$, calcule $P_1, \ldots, P_n$, où n est le degré de P, et P_i est le produit de tous les polynômes irréductibles et unitaires de degré i qui divisent P. Quelle est sa complexité ?

Question 3: Expliquer comment retrouver simplement les facteurs de P et leurs multiplicités à partir des facteurs des P_i.

Dans la seconde étape, on peut donc faire l'hypothèse que le polynôme à factoriser est de la forme

$$Q = Q_1 \cdots Q_r,$$

avec $\deg Q_1 = \cdots = \deg Q_r = s$ et $\deg Q = m = rs$, et où les Q_i sont irréductibles.

On admet dans ces conditions que si on tire un élément A aléatoirement dans $\mathbb{F}_p[X]/Q_i - \{0\}$, avec la distribution uniforme, alors

$$A^{\frac{p^s-1}{2}}$$

vaut 1 (resp. -1) avec probabilité $\frac{1}{2}$.

Question 4: Soit A dans $\mathbb{F}_p[X]/Q$, et $B = A^{\frac{p^s-1}{2}}$ dans $\mathbb{F}_p[X]/Q$. Donner une borne supérieure sur la complexité du calcul de B. Que peut valoir $(B \bmod Q_1, \ldots, B \bmod Q_r)$?

On admettra que si A est tiré aléatoirement dans $\mathbb{F}_p[X]/Q - \{0\}$, avec la distribution uniforme, les valeurs de $(B \bmod Q_1, \ldots, B \bmod Q_r)$ sont toutes équiprobables.

Question 5: Montrer que

$$\operatorname{pgcd}(B-1, Q) = \prod_{i \text{ tel que } (B \bmod Q_i)=1} Q_i.$$

En déduire que si $(B \bmod Q_1, \ldots, B \bmod Q_r) \neq (1, \ldots, 1)$ ainsi que $(B \bmod Q_1, \ldots, B \bmod Q_r) \neq (-1, \ldots, -1)$ alors $\operatorname{pgcd}(B-1, Q)$ est non-trivial. En déduire ensuite un algorithme qui calcule un facteur propre de Q, non constant, avec probabilité $\geq \frac{1}{2}$. Quelle est sa complexité ?

Conclure sur la complexité de la factorisation de Q (on ne demande pas une estimation fine de complexité).