

Algorithmes rapides pour les polynômes, séries formelles et matrices

Alin Bostan

▶ **To cite this version:**

Alin Bostan. Algorithmes rapides pour les polynômes, séries formelles et matrices. Journées Nationales du Calcul Formel 2010, May 2010, Luminy, France. pp.75-262. hal-00780433

HAL Id: hal-00780433
https://inria.hal.science/hal-00780433

Submitted on 23 Jan 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Journées Nationales de Calcul Formel

Rencontre organisée par :
Jean-Guillaume Dumas, Grégoire Lecerf, Delphine Boucher et Thomas Cluzeau

2010

Alin Bostan

Algorithmes rapides pour les polynômes, séries formelles et matrices

Vol. 1, n° 2 (2010), p. 75-262.

<http://ccirm.cedram.org/item?id=CCIRM_2010__1_2_75_0>

Centre international de rencontres mathématiques
U.M.S. 822 C.N.R.S./S.M.F.
Luminy (Marseille) France

cedram

Texte mis en ligne dans le cadre du
Centre de diffusion des revues académiques de mathématiques
http://www.cedram.org/

Les cours du C.I.R.M.
Vol. **1** n° 2 (2010) 75-262

Algorithmes rapides pour les polynômes, séries formelles et matrices

Alin Bostan

Table des matières

Avant-propos 81

Chapitre 1. Calcul Formel et Complexité 83
1. Décider, calculer 83
1.1. Fondements logiques 83
1.2. Structures et constructions de base 83
1.3. Équations comme structures de données 85
2. Calculer rapidement 86
2.1. Mesures de complexité 87
2.2. La notation $O(\cdot)$ 88
2.3. Diviser pour régner 88
Avertissement 90
Notes 91
Bibliographie 91

Chapitre 2. Multiplication rapide 93
1. Introduction, résultats principaux 93
2. Algorithme naïf 95
3. Algorithme de Karatsuba 96
4. Transformée de Fourier rapide 97
4.1. Idée de l'algorithme 97
4.2. Racines primitives de l'unité 97
4.3. Transformée de Fourier rapide 98
4.4. Interpolation 99
4.5. Conclusion 100
4.6. Mais où sont nos racines ? 100
5. L'algorithme de Schönhage et Strassen 101
5.1. Racines virtuelles de l'unité 101
5.2. L'algorithme 101
6. Algorithmes pour les entiers 102
7. Un concept important : les fonctions de multiplication 103
Exercices 103
Notes 104
Bibliographie 105

Chapitre 3. Algèbre linéaire dense : de Gauss à Strassen 107
1. Introduction 107
1.1. L'algorithmique des matrices : tentative de classification 107

Cours professé lors de la rencontre « Journées Nationales de Calcul Formel » organisée par Jean-Guillaume Dumas, Grégoire Lecerf, Delphine Boucher et Thomas Cluzeau. 3-7 mai 2010, C.I.R.M. (Luminy).

1.2. Résultat principal 108
1.3. Applications. 109
2. Multiplication de matrices 109
2.1. Multiplication naïve 110
2.2. Algorithme de Winograd 110
2.3. Algorithme de Waksman 110
2.4. Algorithme de Strassen 111
2.5. Interprétation des formules de Strassen 113
2.6. Peut-on faire mieux que 2,81 ? 114
2.7. En pratique 114
3. Autres problèmes d'algèbre linéaire 115
3.1. Élimination de Gauss 115
3.2. Résultat principal 116
3.3. La multiplication n'est pas plus difficile que l'inversion. 116
3.4. L'inversion, le calcul de déterminant et la résolution de système ne sont pas plus difficiles que la multiplication. 116
3.5. Calcul du polynôme caractéristique 118
Exercices 119
Notes 121
Bibliographie 124

Chapitre 4. Calculs rapides sur les séries 127
1. Séries formelles 128
2. La méthode de Newton pour le calcul d'inverses 130
2.1. Convergence quadratique pour l'inverse d'une série formelle 130
2.2. Algorithme 130
2.3. Complexité 130
2.4. Division de séries 131
2.5. Application à la division Euclidienne 131
2.6. Application au logarithme 131
2.7. Extension : inverse de matrices 131
3. Itération de Newton formelle et applications 132
3.1. Un résultat général 132
3.2. Algorithme 132
3.3. Applications 133
3.4. Exponentielle 133
3.5. Sommes de Newton 133
3.6. Application à la somme et au produit composés 134
3.7. Inverse compositionnel 135
3.8. Systèmes 136
4. La composition des séries 136
4.1. Méthode naïve 136
4.2. Pas de bébés—pas de géants 136
4.3. L'algorithme de Brent & Kung 137
Exercices 138
Notes 139
Bibliographie 139

Chapitre 5. Division euclidienne, fractions rationnelles et récurrences linéaires à coefficients constants 141
1. Division de polynômes 141
1.1. Méthode naïve 141
1.2. Algorithme rapide 141
1.3. Le cas des entiers 142
1.4. Application aux calculs modulaires 142
2. Développement de fractions rationnelles 143

3. Suites récurrentes linéaires à coefficients constants 144
3.1. Prélude : les nombres de Fibonacci 144
3.2. Calcul rapide des termes d'une suite 144
3.3. Propriétés de clôture 145
3.4. Application : tests de primalité 146
Notes 146
Bibliographie 147

Chapitre 6. Calculs modulaires, évaluation et interpolation 149
1. Introduction 149
2. Présentation, résultats 150
3. Interpolation de Lagrange 151
4. Algorithmes rapides 152
4.1. Les idées 152
4.2. L'arbre des sous-produits 152
4.3. Évaluation multipoint rapide 153
4.4. Sommes de fractions 154
4.5. Interpolation 155
4.6. Évaluation et interpolation sur une suite géométrique. 156
Exercices 157
Notes 157
Bibliographie 158

Chapitre 7. Pgcd et résultant 159
1. Algorithme d'Euclide 159
1.1. Le pgcd 159
1.2. Calcul du pgcd 160
1.3. Pgcd étendu et inversion modulaire 161
2. Résultant 162
2.1. Matrice de Sylvester 162
2.2. Applications du résultant 164
2.3. Propriétés et calcul quadratique 166
2.4. Calcul avec des nombres algébriques 167
2.5. Sous-résultants 167
3. Algorithme d'Euclide rapide 170
3.1. Demi-pgcd : définition. 170
3.2. Demi-pgcd : algorithme. 171
3.3. Complément : calcul d'un reste choisi. 172
3.4. Le cas des entiers. 173
Exercices 173
Notes 173
Bibliographie 174

Chapitre 8. Algorithmique des séries D-finies 177
1. Équations différentielles et récurrences 177
1.1. Définitions 177
1.2. Méthode naïve 177
1.3. Exemples d'applications 178
1.4. Algorithme rapide 179
2. Propriétés de clôture 180
2.1. Somme et produit 180
2.2. Produit d'Hadamard 181
3. Séries algébriques 181
4. Limitations 182
Exercices 183
Notes 184

Bibliographie 184

Chapitre 9. Approximants de Padé et de Padé-Hermite 187
1. Reconstruction rationnelle 187
1.1. Calcul de la reconstruction rationnelle 188
1.2. Approximants de Padé 189
1.3. Algorithme de Berlekamp-Massey 190
1.4. Interpolation rationnelle de Cauchy 190
1.5. Algorithmes rapides 191
2. Approximants de Padé-Hermite 191
2.1. Algorithme de Derksen : idées et résultats préliminaires 192
2.2. Algorithme de Derksen : fonctionnement 193
2.3. Approximants de Padé-Hermite de type arbitraire 194
2.4. Applications 194
2.5. Calcul quasi-optimal 196
Notes 196
Bibliographie 197

Chapitre 10. Algèbre linéaire creuse : algorithme de Wiedemann 199
1. Introduction 199
2. Polynôme minimal et résolution de systèmes 199
3. Calcul du polynôme minimal 200
4. Calcul du déterminant 201
5. Calcul du rang 201
Exercices 201
Notes 202
Bibliographie 202

Chapitre 11. Algèbre linéaire structurée 203
1. Introduction 203
2. Le cas quasi-Toeplitz 206
2.1. Produit matrice-vecteur, cas quasi-Toeplitz 207
2.2. Addition et produit en représentation compacte par générateurs 207
2.3. Inversion en représentation compacte par générateurs 208
2.4. Résolution rapide de systèmes quasi-Toeplitz 209
Exercices 209
Notes 210
Bibliographie 210

Chapitre 12. Principe de transposition de Tellegen 213
1. Introduction 213
2. La version en termes de graphes du principe de Tellegen 214
3. Principe de Tellegen pour les programmes linéaires 216
3.1. Machines à registres 216
3.2. Transposition de programme 216
3.3. Optimisations 217
4. Applications 217
4.1. Produit médian des polynômes 218
4.2. Division avec reste et extension des récurrences à coefficients constants 219
4.3. Évaluation multipoint et interpolation 220
4.4. Bonus : l'évaluation et l'interpolation ont des coûts équivalents 221
Notes 222
Bibliographie 224

Chapitre 13. Récurrences linéaires à coefficients polynomiaux : N-ième terme, N premiers termes 227
1. Calcul naïf de $N!$ et de suites P-récursives 227

1.1. Cas de la factorielle 227
1.2. Cas général 228
2. Pas de bébés et pas de géants 229
2.1. Factorisation déterministe des entiers. 229
3. Scindage binaire 230
3.1. Cas de la factorielle 230
3.2. Récurrences d'ordre 1 230
3.3. Calcul de $e = \exp(1)$ 231
3.4. Suites polynomialement récursives 231
Exercices 231
Notes 232
Bibliographie 232

Chapitre 14. Solutions rationnelles de systèmes linéaires à coefficients polynomiaux 235
1. Des séries aux solutions rationnelles 235
2. Développement comme une fraction rationnelle 236
3. L'algorithme de Storjohann 237
Notes 238
Bibliographie 238

Chapitre 15. Solutions séries d'équations différentielles 241
1. Équations différentielles d'ordre 1 241
1.1. L'équation linéaire homogène du premier ordre 241
1.2. L'équation linéaire inhomogène du premier ordre 242
1.3. Le cas non-linéaire 242
2. Équations différentielles linéaires d'ordre supérieur et systèmes d'ordre 1 243
2.1. Une méthode par « diviser pour régner » 243
2.2. L'itération de Newton matricielle 244
3. Cas particuliers 246
3.1. Équations différentielles linéaires à coefficients polynomiaux 246
3.2. Équations différentielles linéaires à coefficients constants 246
4. Extensions 247
4.1. Composer se ramène à résoudre 247
4.2. Systèmes non-linéaires 248
Notes 248
Bibliographie 248

Chapitre 16. Exercices récapitulatifs 249
Notes 259
Bibliographie 259

Annexe : liste des algorithmes 261

Avant-propos

Le calcul formel calcule des objets mathématiques exacts. Ce cours explore deux directions : la calculabilité et la complexité. La calculabilité étudie les classes d'objets mathématiques sur lesquelles des réponses peuvent être obtenues algorithmiquement. La complexité donne ensuite des outils pour comparer des algorithmes du point de vue de leur efficacité.

Ce cours passe en revue l'algorithmique efficace sur les objets fondamentaux que sont les entiers, les polynômes, les matrices, les séries et les solutions d'équations différentielles ou de récurrences linéaires. On y montre que de nombreuses questions portant sur ces objets admettent une réponse en complexité (quasi-)optimale, en insistant sur les principes généraux de conception d'algorithmes efficaces.

Ces notes sont dérivées du cours « Algorithmes efficaces en calcul formel » du Master Parisien de Recherche en Informatique (2004–2010), co-écrit avec Frédéric Chyzak, Marc Giusti, Romain Lebreton, Bruno Salvy et Éric Schost. Le support de cours complet est disponible à l'url `http://mpri.master.univ-paris7.fr/C-2-22.html`.

CHAPITRE 1

Calcul Formel et Complexité

Résumé

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, 1968)**.** *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. La notion d'effectivité permet de préciser ce point de vue.

Définition 1. Une structure algébrique (groupe, anneau, corps, espace vectoriel,...) est dite *effective* si l'on dispose :
- d'une structure de données pour en représenter les éléments ;
- d'algorithmes pour en effectuer les opérations et pour y tester l'égalité.

Par exemple, dans un anneau effectif, outre l'égalité, les opérations requises sont l'addition, la soustraction et la multiplication.

1.2. **Structures et constructions 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. Ils s'assemblent ensuite à l'aide de tableaux ou de listes pour en former de plus complexes.

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 carrée 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 :

$$\begin{aligned}
P_0 &= 7X^5 - 22X^4 + 55X^3 + 94X^2 - 87X + 56,\\
P_1 &= 62X^4 - 97X^3 + 73X^2 + 4X + 83,\\
P_2 &= \mathrm{rem}(P_0, P_1) = \frac{113293}{3844}X^3 + \frac{409605}{3844}X^2 - \frac{183855}{1922}X + \frac{272119}{3844},\\
P_3 &= \mathrm{rem}(P_1, P_2) = \frac{18423282923092}{12835303849}X^2 - \frac{15239170790368}{12835303849}X + \frac{10966361258256}{12835303849},\\
P_4 &= \mathrm{rem}(P_2, P_3) = -\frac{21613227465379239544 8637}{44148979404824831944178}X - \frac{631179956389122192280133}{88297958809649663888356},\\
P_5 &= \mathrm{rem}(P_3, P_4) = \frac{20556791167692068695002336923491296504125}{3639427682941980248860941972667354081}.
\end{aligned}$$

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.

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 pour la recherche de noyau ou d'inverse de matrices.

Proposition 1. *Si $\mathbb{K}$ est un corps effectif, l'espace vectoriel $\mathbb{K}^n$ l'est aussi, ainsi que l'anneau $\mathcal{M}_n(\mathbb{K})$.*

Polynômes. Les polynômes peuvent être représentés 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.

Dans les deux cas, nous avons clairement :

Proposition 2. *Si $\mathbb{A}$ est un anneau effectif, alors $\mathbb{A}[X]$ l'est aussi.*

L'usage récursif de cette proposition fournit les polynômes multivariés.

Fractions rationnelles. 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. De même, les fractions rationnelles sont représentées par des paires de polynômes. Les opérations d'addition, produit, division, se réduisent aux additions et multiplications sur les coefficients. Plus généralement, nous obtenons :

Proposition 3. *Si $\mathbb{A}$ est un anneau intègre effectif, alors son corps des fractions est effectif.*

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.

Proposition 4. *Si $\mathbb{A}$ est un anneau effectif et $N \in \mathbb{N}$, alors $A[X]/(X^{N+1})$ est un anneau effectif.*

Cette structure de données joue un rôle très important non seulement pour des calculs d'approximations, mais aussi comme une représentation *exacte*. En voici trois exemples :

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 la manipulation des nombres algébriques (Chapitre 4), et dans le calcul efficace de la division euclidienne de polynômes, de suites récurrentes linéaires (comme le calcul rapide du 10 000-ième nombre de Fibonacci au Chapitre 5) et du polynôme minimal d'une matrice creuse (Chapitre 10).
2. 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 premières valeurs d'une de ses solutions. L'outil algorithmique pour effectuer ces calculs de *devinette* (*guessing* en anglais) est le calcul rapide d'approximants de Padé-Hermite (Chapitre 9).
3. Un polynôme en deux variables peut être reconstruit à partir du développement en série d'une solution. L'efficacité de la factorisation des polynômes à deux variables, ou encore de la résolution de systèmes polynomiaux par la méthode dite de *résolution géométrique*, repose de manière cruciale sur cette opération, qui doit être effectuée rapidement. Ces applications dépassent le cadre de ce cours et n'y seront pas abordées.

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. Le résultat est spectaculaire.

Proposition 5. *Si $\mathbb{K}$ est un corps effectif, alors sa clôture algébrique $\overline{\mathbb{K}}$ l'est aussi.*

Les opérations d'addition et de multiplication peuvent être effectuées à l'aide de résultants (Chapitre 7). Ceux-ci peuvent être calculés efficacement à l'aide de séries (Chapitre 4). La division s'obtient par l'algorithme d'Euclide sur les polynômes (Chapitre 7), et le test à zéro se déduit du pgcd. Par exemple, il est possible de prouver assez facilement une identité comme

$$\frac{\sin\frac{2\pi}{7}}{\sin^2\frac{3\pi}{7}} - \frac{\sin\frac{\pi}{7}}{\sin^2\frac{2\pi}{7}} + \frac{\sin\frac{3\pi}{7}}{\sin^2\frac{\pi}{7}} = 2\sqrt{7} \tag{1}$$

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

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 que bien sûr 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 (Chapitre 8). Le test à zéro se ramène à tester l'égalité d'un nombre fini de conditions initiales. Une partie de ces résultats se résume ainsi :

Proposition 6. *Si $\mathbb{K}$ est un corps effectif, les séries formelles de $\mathbb{K}[[X]]$ qui sont solution d'équations différentielles linéaires à coefficients dans $\mathbb{K}[X]$ forment un anneau effectif.*

En d'autres termes, des structures de données finies permettent de manipuler ces objets infinis et d'en tester l'égalité ou la nullité.

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 $\sin^2 X + \cos^2 X = 1$. Le pendant du résultat précédent est donc :

Proposition 7. *Si $\mathbb{K}$ est un corps effectif, l'ensemble des suites de $\mathbb{K}^{\mathbb{N}}$ solutions de récurrences linéaires à coefficients dans $\mathbb{K}[n]$ forme un anneau effectif.*

En conclusion, 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. Ce cours est consacré 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 500 000 000 de chiffres ;
   - factorielle de 20 000 000 (environ 140 000 000 de chiffres) ;
   - factorisation d'un entier de 45 chiffres (150 bits).
2. Polynômes dans $\mathbb{K}[X]$:
   - produit de deux polynômes de degré 14 000 000 (plus d'un an avec la méthode naïve) ;
   - pgcd de deux polynômes de degré 600 000 ;
   - résultant de deux polynômes de degré 40 000 ;
   - factorisation d'un polynôme de degré 4 000.
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 $4\,500 \times 4\,500$ 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'introduire une mesure de complexité des différents algorithmes permettant d'expliquer, voire de prédire, les différences entre les 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 anneau $\mathbb{A}$ donné et nous mesurerons alors la complexité en nombre d'opérations arithmétiques dans $\mathbb{A}$.

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 Chapitre 13 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

1. La machine a un processeur AMD 64 à 2,2 GHz et une mémoire de 2 Go ; le système d'exploitation est linux.

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 données 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 il n'y a pas de différence importante entre les deux.

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.

Définition 2. Si N est la somme de la taille de l'entrée et de la taille de la sortie, un algorithme sera dit *quasi-optimal* lorsque sa complexité est bornée par $O(N \log^k N)$ pour un $k \geq 0$ arbitraire.

L'essentiel 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-ième 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 (voir Figure 1). Le coût de la recombinaison et éventuellement du découpage préliminaire est borné par une fonction T de la taille des entrées. Lorsque les entrées ont une taille inférieure à

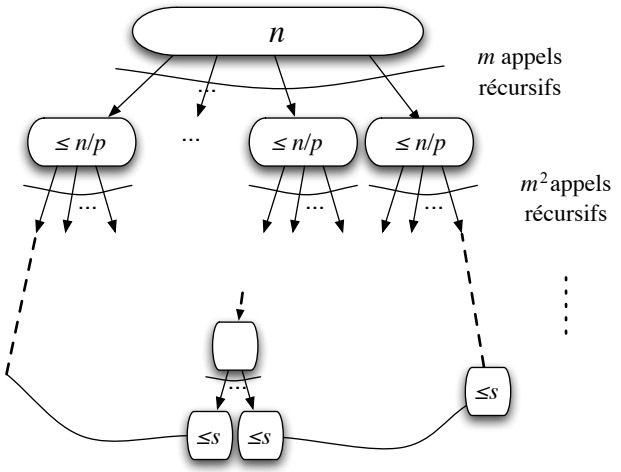


FIGURE 1: Les appels récursifs d'un algorithme « diviser pour régner ».

p ou suffisamment petite (s) un autre algorithme de coût κ indépendant de n est invoqué. Le coût total obéit alors à une récurrence de la forme

$$C(n) \leq \begin{cases} T(n) + mC(\lceil n/p \rceil), & \text{si } n \geq s (\geq p) \\ \kappa & \text{sinon.} \end{cases} \tag{2}$$

La notation $\lceil x \rceil$ désigne l'entier k tel que $k - 1 < x \leq k$ (le « plafond » de x).

Qualitativement, le coût total de cette approche dépend fortement de la fonction T. Lorsque T est relativement coûteuse, les premiers niveaux de l'arbre contribuent à l'essentiel du coût et, à une constante près, le coût est donc dominé par la première étape de récursion. Les algorithmes à base d'itération de Newton du Chapitre 4 sont de ce type. À l'inverse, pour une T assez faible, le bas de l'arbre domine le coût qui sera proportionnel au nombre de feuilles [2] $O((n/s)^{\log_p m})$. L'algorithme de multiplication de polynômes de Karatsuba du Chapitre 2 et l'algorithme de Strassen pour la multiplication de matrices (Chapitre 3) rentrent dans cette catégorie. Enfin, il est possible que tous les $O(\log_p(n/s))$ niveaux de l'arbre contribuent de manière assez équilibrée, menant à un coût en $O(T(n) \log n)$. La transformée de Fourier rapide (Chapitre 2) et le classique algorithme de tri fusion sont dans cette dernière catégorie.

Un cadre commode pour nos applications est résumé dans la proposition suivante, dont une version simplifiée que nous utiliserons dans la plupart des cas est donnée ensuite par le théorème 2. Nous commençons par le cas simple où la taille est une puissance de p, où il est possible de donner des constantes précises dans les inégalités.

Proposition 8. *Soit C une fonction obéissant à l'inégalité* (2) *avec $m > 0$, $\kappa > 0$ et T une fonction telle que*

$$qT(n) \leq T(pn), \qquad n \in \mathbb{N}, \tag{3}$$

avec $1 < q$. Alors, si $n \to \infty$ est une puissance de p,

$$C(n) \leq \begin{cases} \left(1 - \frac{m}{q}\right)^{-1} T(n) + \kappa\, n^{\log_p m} & \text{si } q > m, \\ T(n) \log_p n + \kappa\, n^{\log_p q} & \text{si } q = m, \\ n^{\log_p m} m^{1 - \log_p s} \left(\frac{T(n)}{n^{\log_p q}} \frac{q^{\log_p s}}{m - q} + \kappa \right) & \text{si } q < m. \end{cases}$$

2. La notation $\log_p x$ représente le logarithme en base p de x, c'est-à-dire $\log x / \log p$. L'identité $a^{\log_p b} = b^{\log_p a}$ sera utile.

Démonstration. Par récurrence, l'inégalité (3) entraîne d'abord $q^{\log_p n}T(1) \leq T(n)$. Ensuite, l'utilisation répétée de l'inégalité sur C donne, avec $k = \lfloor \log_p \frac{n}{s} \rfloor + 1$:

$$C(n) \leq T(n) + mC\left(\frac{n}{p}\right),$$

$$\leq T(n) + mT\left(\frac{n}{p}\right) + \cdots + m^{k-1}T\left(\frac{n}{p^{k-1}}\right) + m^k C\left(\frac{n}{p^k}\right), \tag{4}$$

$$\leq T(n)\left(1 + \frac{m}{q} + \cdots + \left(\frac{m}{q}\right)^{k-1}\right) + m^k\kappa, \tag{5}$$

où la dernière ligne résulte de (3).

Si $m < q$, la somme entre parenthèses est majorée par la série géométrique, et le dernier terme m^k est un $O(n^{\log_p m})$ ce qui donne l'inégalité.

Si $m = q$, la somme comporte k termes égaux et le dernier terme $m^k = q^k = O(q^{\log_p n}) = O(T(n))$ permet de conclure.

Si $m > q$, récrire la somme sous la forme

$$\left(\frac{m}{q}\right)^{k-1}\left(1 + \frac{q}{m} + \cdots + \left(\frac{q}{m}\right)^{k-1}\right)$$

montre que le premier terme de (5) est majoré par

$$T(n)\left(\frac{m}{q}\right)^k \frac{1}{m-q}$$

dont l'ordre de grandeur est au moins celui du dernier terme puisque $T(n) \geq q^k$. □

En pratique la régularité de T est souvent plus forte que celle donnée par (3) et on ne cherche qu'à comparer des estimations de $O()$. Le résultat prend alors une forme plus simple, et sans restriction sur n.

Théorème 2 (Diviser pour régner)**.** *Soit C une fonction obéissant à l'inégalité* (2) *avec $m > 0$ et $\kappa > 0$, et soit T une fonction* croissante *telle que*

$$qT(n) \leq T(pn) \leq rT(n), \qquad n \in \mathbb{N}, \tag{6}$$

avec $1 < q \leq r$. Alors, lorsque $n \to \infty$

$$C(n) = \begin{cases} O(T(n)), & \text{si } q > m, \\ O(T(n)\log n), & \text{si } q = m, \\ O\left(n^{\log_p m}\frac{T(n)}{n^{\log_p q}}\right) & \text{si } q < m. \end{cases}$$

Démonstration. Soit N la puissance de p telle que $n \leq N < pn$. En déroulant l'inégalité (2) on obtient une inégalité similaire à (4) où la division par p est remplacée par l'opération $x \mapsto \lceil x/p \rceil$. La fonction T étant supposée croissante, toutes ces valeurs de T sont majorées par les valeurs en N/p^i, $i = 0, \ldots, k-1$ et on obtient donc une majoration (5) où n est remplacé par N. Il ne reste plus qu'à majorer cette expression. La croissance de T et l'hypothèse (6) donnent $T(N) \leq T(pn) = O(T(n))$, ce qui permet de conclure. □

Exemple 4. La complexité de l'algorithme de Karatsuba du Chapitre 2 s'obtient avec $m = 3$, $p = 2$ et T linéaire, si bien que $q = r = 2$, d'où une complexité $C(n) = O(n^{\log_2 3})$.

Avertissement

Pour toute la suite du cours, et sauf mention contraire, nous faisons les conventions de notation et de terminologie suivantes :

- tous les anneaux et corps sont supposés effectifs, commutatifs et unitaires ;
- les estimations de complexité sont en nombre d'opérations arithmétiques dans l'anneau de base ;
- le symbole □ marque la fin d'une démonstration ;
- les logarithmes sont en base 2, lorsque la base n'est pas spécifiée ;

– le lemme « diviser pour régner » fait référence à l'énoncé de la Proposition 3 ;
– le théorème « diviser pour régner » fait référence à l'énoncé du Théorème 2.

Notes

Les références générales sur les algorithmes du calcul formel sont deux livres : celui de von zur Gathen et Gerhard [8] et celui, plus élémentaire, de Geddes, Czapor et Labahn [4]. La complexité est également utilisée comme fil conducteur dans le livre plus difficile de Bürgisser, Clausen et Shokrollahi [3].

Le théorème de Richardson [6] s'applique à des fonctions. Pour des constantes, l'approche la plus récente [7] réduit le test à zéro à une conjecture de théorie des nombres due à Schanuel 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* [5]. 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, *straight-line program*, machine de Turing, ...) sont bien présentés par Aho, Hopcrof et Ullman dans [1].

La jolie identité (1) est tirée de [2].

Bibliographie

[1] Aho (Alfred V.), Hopcroft (John E.), and Ullman (Jeffrey D.). – *The design and analysis of computer algorithms*. – Addison-Wesley Publishing Co., 1974, x+470p.

[2] Beck (Matthias), Berndt (Bruce C.), Chan (O-Yeat), and Zaharescu (Alexandru). – Determinations of analogues of Gauss sums and other trigonometric sums. *Int. J. Number Theory*, vol. 1, n3, 2005, pp. 333–356.

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

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

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

[6] Richardson (Daniel). – Some undecidable problems involving elementary functions of a real variable. *Journal of Symbolic Logic*, vol. 33, n1, 1968, pp. 514–520.

[7] Richardson (Daniel). – How to recognize zero. *Journal of Symbolic Computation*, vol. 24, n6, 1997, pp. 627–645.

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

CHAPITRE 2

Multiplication rapide

Résumé

Les algorithmes rapides de calcul de produit de polynômes et d'entiers sont au cœur de l'algorithmique efficace en calcul formel. La plupart des gains de complexité dans les chapitres ultérieurs reposent sur l'efficacité de la multiplication. Pour multiplier deux polynômes de degré n à coefficients dans un anneau $\mathbb{A}$, la méthode classique requiert $O(n^2)$ opérations dans $\mathbb{A}$. 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 chapitre 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 chapitre 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 $\mathbb{Z}[X]$.

$$\begin{aligned}&(3X^2 + 2X + 1) \times (6X^2 + 5X + 4)\\ &= (3 \cdot 6)X^4 + (3 \cdot 5 + 2 \cdot 6)X^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)X^2 + (2 \cdot 4 + 1 \cdot 5)X + (1 \cdot 4)\\ &= 18X^4 + 27X^3 + 28X^2 + 13X + 4.\end{aligned}$$

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

$$\begin{aligned}&(3 \cdot 10^2 + 2 \cdot 10 + 1) \times (6 \cdot 10^2 + 5 \cdot 10 + 4)\\ &= (3 \cdot 6)10^4 + (3 \cdot 5 + 2 \cdot 6)10^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)10^2 + (2 \cdot 4 + 1 \cdot 5)10 + (1 \cdot 4)\\ &= 18 \cdot 10^4 + 27 \cdot 10^3 + 28 \cdot 10^2 + 13 \cdot 10 + 4\\ &= 2 \cdot 10^5 + 9 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10 + 4 = 209934.\end{aligned}$$

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, $(\mathbb{A}, +, \times)$ désignera un anneau (commutatif et unitaire). Tout d'abord, nous considérons la complexité arithmétique ; il s'agit de minimiser le nombre d'opérations dans $\mathbb{A}$ $(+, -, \times)$ du produit des polynômes en degré borné. Les premiers résultats à retenir de ce chapitre sont les suivants.

La multiplication des polynômes de degré au plus n dans $\mathbb{A}[X]$ requiert :
- $O(n^2)$ *opérations dans* $\mathbb{A}$ *par* l'algorithme naïf *;*
- $O(n^{1,59})$ *opérations dans* $\mathbb{A}$ *par* l'algorithme de Karatsuba *;*
- $O(n \log n \log\log n)$*, voire dans certains cas* $O(n \log n)$ *opérations dans* $\mathbb{A}$*, 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. Par exemple, lorsque $\mathbb{A}$ est un corps fini de taille « raisonnable » (typiquement, dont les éléments sont représentés sur quelques mots machine), pour le produit de polynômes 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 $O(n \log n)$ battent l'implantation de Karatsuba 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. Plus fréquemment, des degrés de l'ordre de la centaine ou du millier font partie du quotidien du calculateur formel (au moins dans des calculs intermédiaires).

Ces algorithmes ne sont pas concurrents, mais complémentaires. Les bonnes implantations passent automatiquement sur l'algorithme le plus efficace en fonction du degré. Nous donnons dans la Figure 1 la courbe de complexité pratique [1] de la multiplication polynomiale à coefficients dans le corps fini $\mathbb{A} = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

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 des implantations actuelles de la FFT.

La situation pour les entiers est similaire, même si l'implantation des algorithmes rapides pour les entiers est bien plus 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 entiers de quelques centaines de chiffres sont courants dès que l'on sort des exemples jouets. Au-delà, des problèmes venus de la cryptologie ou de la théorie des nombres

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

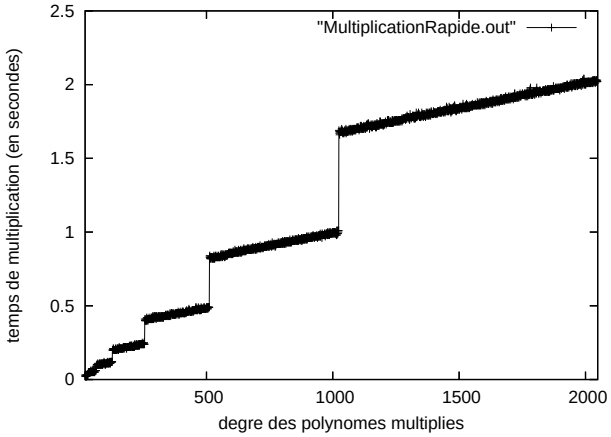


FIGURE 1: Courbe de complexité pratique de la multiplication dans $\mathbb{A}[X]$, où $\mathbb{A} = \mathbb{Z}/4179340454199820289\,\mathbb{Z}$.

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.

Notation 1. On travaille avec des polynômes F et G à coefficients dans un anneau $\mathbb{A}$, 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 $\mathbb{A}$. 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 $\mathbb{A}$ et mn additions dans $\mathbb{A}$.

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 en valeur absolue par un entier H.

Exercice 4. Shigeru Kondo a calculé 25 000 000 000 décimales de π sur un PC de bureau [2]. 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.

2. Calcul effectué en 2003 ; le record actuel est de $2,7\times10^{12}$ chiffres, obtenu par Fabrice Bellard `http://bellard.org/pi/pi2700e9/announce.html`.

Algorithme de Karatsuba

Entrée : F, G de degré au plus $n-1$.

Sortie : $H = FG$.

1. Si $n = 1$, renvoyer FG.
2. Décomposer F et G selon l'équation (1).
3. Calculer $A_1 = F^{(0)}G^{(0)}$ et $A_2 = F^{(1)}G^{(1)}$ récursivement.
4. Calculer $A_3 = F^{(0)} + F^{(1)}$ et $A_4 = G^{(0)} + G^{(1)}$.
5. Calculer $A_5 = A_3A_4$ récursivement.
6. Calculer $A_6 = A_5 - A_1$ et $A_7 = A_6 - A_2$.
7. Renvoyer $A_1 + A_7X^k + A_2X^{2k}$.

Figure 2: Multiplication de polynômes par l'algorithme de Karatsuba.

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, parmi les 4 du produit par l'algorithme quadratique. Soient en effet à multiplier les polynômes

$$F = f_0 + f_1X \quad \text{et} \quad G = g_0 + g_1X.$$

Le produit $H = FG$ s'écrit

$$H = f_0g_0 + (f_0g_1 + f_1g_0)X + f_1g_1X^2.$$

On évite d'effectuer les 4 produits $f_0g_0, f_0g_1, f_1g_0, f_1g_1$ en remarquant que le coefficient de X s'écrit

$$f_0g_1 + f_1g_0 = (f_0 + f_1)(g_0 + g_1) - f_0g_0 - f_1g_1.$$

Cette écriture mène à un algorithme qui effectue au total 3 multiplications et 4 additions.

Quelques additions sont perdues 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. En s'inspirant de l'observation précédente, nous scindons F et G en deux. Nous supposons donc que F et G sont de degré au plus $n-1$, et choisissons $k = \lceil n/2 \rceil$. Posons alors

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

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

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

L'algorithme est présenté en Figure 2.

Le théorème qui suit établit maintenant la complexité de l'algorithme.

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

Démonstration. Un appel en degré $< n$ effectue 3 appels récursifs en degré $< n/2$, plus quelques additions, comptées comme suit : l'étape 4 effectue 2 additions en tailles $< n/2$; l'étape 6 effectue 2 additions en tailles $< n$; quant à l'étape 7, A_1 et A_2X^n sont à supports monomiaux distincts, donnant la somme $A_1 + A_2X^n$ sans coût arithmétique, puis la somme totale en 1 addition en taille $< n$. Le coût $K(n)$ satisfait donc à la récurrence

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

où le terme $4n$ vient compter le nombre d'additions dans $\mathbb{A}$. Le lemme « diviser pour régner » permet alors de conclure avec $p = q = s = 2$, $m = 3$, $T(n) = 4n$, et $\kappa = 1$ (coût de l'algorithme naïf en degré 0). □

Le résultat général suivant se déduit pour un degré quelconque.

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

En effet, soit N la plus petite puissance de 2 telle que $N > n$. Alors, le produit de F et de G perd au pire un facteur constant par rapport à l'estimation du Théorème 1.

Exercice 5. Borner la constante cachée dans l'estimation asymptotique du Corollaire 1.

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 6. Soit n une puissance de 2. Établir un algorithme hybride de multiplication dans $\mathbb{A}[X]$, 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 minimise $\gamma(d)$ et comparer le résultat avec celui du Théorème 1.

Exercice 7. Est-il possible de multiplier par un algorithme universel, c'est-à-dire indépendant de l'anneau de base $\mathbb{A}$, deux polynômes de degré au plus 1 en utilisant seulement 2 multiplications dans $\mathbb{A}$?

Exercice 8. Soit $\mathbb{A}$ un anneau et soient A et B deux polynômes de degré au plus 3 dans $\mathbb{A}[X]$.

1. Estimer le nombre de multiplications de $\mathbb{A}$ requises par l'algorithme de Karatsuba pour calculer le produit AB.
2. On suppose que 2, 3 et 5 sont inversibles dans $\mathbb{A}$ et que la division d'un élément de $\mathbb{A}$ par 2, 3 et 5 est gratuite. Donner un algorithme qui multiplie A et B en utilisant au plus 7 multiplications dans $\mathbb{A}$.
3. On suppose que 2, 3 et 5 sont inversibles dans $\mathbb{A}$. Donner un algorithme de multiplication polynomiale dans $\mathbb{A}[X]$ de complexité arithmétique $O(n^{1,4})$.

Dans la suite de l'exercice, on suppose que l'anneau $\mathbb{A}$ est de caractéristique nulle.

4. Montrer que, pour tout entier $\alpha \geq 2$, il existe un algorithme de multiplication polynomiale dans $\mathbb{A}[X]$ 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 $\mathbb{A}[X]$ 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 rapide (aussi appelée FFT pour *Fast Fourier Transform*) 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 $\mathbb{A}[X]$, de degrés strictement inférieurs à $n/2$ (ou plus généralement tels que $\deg FG < n$).

4.1. Idée de l'algorithme.

En supposant que l'anneau $\mathbb{A}$ le permette, l'idée générale de l'algorithme est présentée en Figure 3. Il s'agit d'évaluer en des points bien choisis, de multiplier les évaluations, et de reconstruire les coefficients du produit à partir de ces valeurs (à supposer que cette opération d'interpolation soit possible, voir ci-dessous). Avec l'hypothèse $\deg H < n$, les coefficients de $H \bmod X^n - 1$ qui sont renvoyés sont bien ceux de H. Le coût des étapes de précalcul et de produit point à point est linéaire en n, et il reste donc à voir comment effectuer rapidement les opérations d'évaluation et d'interpolation.

4.2. Racines primitives de l'unité.

Définition 1. L'élément ω de $\mathbb{A}$ est une *racine n-ième de l'unité* si $\omega^n = 1$; c'est une racine n-ième *primitive* de l'unité si de plus $\omega^t - 1$ est non diviseur de zéro dans $\mathbb{A}$ pour $t \in \{1, \ldots, n-1\}$ (c'est-à-dire que $\alpha(\omega^t - 1) = 0$ implique $\alpha = 0$).

Multiplication par FFT

Entrée : F, G deux polynômes, n un entier, ω une racine primitive n^{e} de 1.

Sortie : $H = FG \bmod X^n - 1$.

1. *Précalcul.* Calculer les puissances $\omega^2, \dots, \omega^{n-1}$.
2. *Évaluation.* Calculer les valeurs :
$$\mathsf{Ev}(F) = \big(F(\omega^0), \dots, F(\omega^{n-1})\big); \quad \mathsf{Ev}(G) = \big(G(\omega^0), \dots, G(\omega^{n-1})\big).$$
3. *Produit point à point.*
$$\big(\mathsf{Ev}(F), \mathsf{Ev}(G)\big) \mapsto \mathsf{Ev}(FG) = \big(FG(\omega^0), \dots, FG(\omega^{n-1})\big).$$
4. *Interpolation.*
$$\mathsf{Ev}(FG) \mapsto FG.$$

Figure 3: Multiplication de polynômes par transformée de Fourier discrète.

Remarque. Dans cette définition, la condition t *diviseur strict de* n suffit, elle entraîne la propriété pour $t \in \{1, \dots, n-1\}$.

En effet, en posant g le pgcd de t et de n, il existe d'après le théorème de Bézout deux entiers $p, q \in \mathbb{N}$ tels que $g = tp + nq$. L'égalité $\alpha(\omega^t - 1) = 0$ entraîne alors $0 = \alpha(\omega^{tp} - 1) = \alpha(\omega^{g-nq} - 1) = \alpha(\omega^g - 1)$, et donc $\alpha = 0$, puisque g est un diviseur strict de n.

Exemple 1. Si $\mathbb{A}$ est un corps, cette dernière condition revient simplement à dire que les ω^t sont distincts pour $t \in \{0, \dots, n-1\}$, ou que ω engendre le groupe des racines n-ièmes de l'unité. Par exemple, dans $\mathbb{C}$, -1 n'est pas une racine 4-ième primitive de l'unité, alors que i l'est. Plus généralement, lorsque $\mathbb{A} = \mathbb{C}$, les racines primitives n-ième de l'unité sont de la forme $\exp(2qi\pi/n)$, pour q premier avec n.

Exemple 2. Dans $\mathbb{A} = \mathbb{Z}/8\mathbb{Z}$, 3, 5 ou 7 sont racines carrées primitives de 1.

Les propriétés que nous utiliserons sont résumées dans le lemme suivant.

Lemme 1. *Si ω est racine primitive n-ième de l'unité, alors*

1. *ω^{-1} aussi ;*
2. *si $n = pq$ alors ω^p est une racine primitive q-ième de l'unité ;*
3. *pour $\ell \in \{1, \dots, n-1\}$,*
$$\sum_{j=0}^{n-1} \omega^{\ell j} = 0.$$

Démonstration. Tout d'abord ω est bien inversible : l'identité $\omega^n = 1$ montre que ω^{n-1} est un inverse de ω. Ensuite, ω^{-1} est une racine de l'unité : le produit de l'identité précédente par ω^{-n} donne $1 = \omega^{-n}$. Enfin, elle est primitive : lorsque $\omega^t - 1$ n'est pas diviseur de 0, son produit par l'inversible ω^{-t} non plus.

La deuxième propriété est immédiate. Pour la troisième, le produit de la somme par $1 - \omega^\ell$ télescope les sommants, ce qui donne

$$(1 - \omega^\ell) \sum_{j=0}^{n-1} \omega^{\ell j} = 1 - \omega^{\ell n} = 1 - (\omega^n)^\ell = 0.$$

Comme $1 - \omega^\ell$ n'est pas diviseur de 0, la somme est bien nulle. □

4.3. **Transformée de Fourier rapide.** L'opération

$$\mathsf{DFT} : F \in \mathbb{A}[X] \mapsto (F(1), F(\omega), \dots, F(\omega^{n-1})$$

s'appelle la *transformée de Fourier discrète*. Son calcul rapide est effectué par un algorithme de type « diviser pour régner ».

Transformée de Fourier Rapide

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

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

1. Si $n = 1$, renvoyer f_0.
2. Sinon, soit $k = n/2$. Calculer

$$R_0(X) = \sum_{j=0}^{k} (f_j + f_{j+k})X^j, \quad \bar{R}_1(X) = R_1(\omega X) = \sum_{j=0}^{k} (f_j - f_{j+k})\omega^j X^j.$$

3. Calculer récursivement $R_0(1), R_0(\omega^2), \ldots, R_0\big((\omega^2)^{k-1}\big)$ et $\bar{R}_1(1), \bar{R}_1(\omega^2), \ldots, \bar{R}_1\big((\omega^2)^{k-1}\big)$.
4. Renvoyer $R_0(1), \bar{R}_1(1), R_0(\omega^2), \bar{R}_1(\omega^2), \ldots, R_0\big((\omega^2)^{k-1}\big), \bar{R}_1\big((\omega^2)^{k-1}\big)$.

FIGURE 4: La transformée de Fourier rapide (FFT).

Pour appliquer cette idée, supposons que n est pair, $n = 2k$. Alors, $\omega^k = -1$ puisque

$$(\omega^k - 1)(\omega^k + 1) = \omega^n - 1 = 0$$

et le premier facteur n'est pas diviseur de 0. Le polynôme F est décomposé par division euclidienne de deux façons :

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

avec $\deg R_0 < k$ et $\deg R_1 < k$. Ces décompositions vont nous permettre le calcul de F sur les puissances paires et impaires de ω. En effet, si ℓ est pair, $\omega^{k\ell} = 1$ et donc $F(\omega^\ell) = R_0(\omega^\ell)$. De même, si ℓ est impair, $F(\omega^\ell) = R_1(\omega^\ell)$.

Outre l'application récursive, le point crucial qui est la source de l'efficacité de l'algorithme de transformée de Fourier rapide (Figure 4) et qui conduit au choix de racines primitives de l'unité, est que le calcul de R_0 et R_1 est très simple (étape 2). Le fait que les ω^{2^i} utilisés dans les appels récursifs sont bien des racines primitives découle du Lemme 1.

Théorème 2. *L'algorithme de Transformée de Fourier Rapide de la Figure 4 requiert au plus* $\frac{3n}{2} \log n$ *opérations dans* $\mathbb{A}$ *; les multiplications font toutes intervenir une puissance de* ω*.*

Démonstration. Puisque les puissances de ω sont connues, 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 de $n/2$ multiplications (pour le calcul de $\bar{R}_1$), plus 2 appels récursifs en degré $n/2$. Sa complexité $F(n)$ satisfait donc à la récurrence :

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

et le lemme « diviser pour régner » (avec $p = q = m = s = 2$, $T(n) = 3n/2$ et $\kappa = 0$) permet de conclure. □

Exercice 9. Montrer que l'algorithme de la Figure 4 requiert $n \log n$ additions dans $\mathbb{A}$, $\frac{1}{2} n \log n$ multiplications d'éléments de $\mathbb{A}$ par des puissances de ω, mais aucune autre multiplication dans $\mathbb{A}$.

La transformée de Fourier discrète est un morphisme d'algèbres sur $\mathbb{A}$ de $\mathbb{A}[X]/(X^n - 1)$ dans $\mathbb{A}^n$ avec comme multiplication dans $\mathbb{A}^n$ la multiplication coordonnée par coordonnée. Cette observation permet d'économiser des transformées inverses en effectuant plusieurs calculs directement sur les transformées.

4.4. **Interpolation.** 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

$$V_\omega = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ \vdots & & & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)^2} \end{bmatrix}.$$

Lemme 2. *Si $\omega \in \mathbb{A}$ est racine n-ième primitive de l'unité, alors $V_{\omega^{-1}} V_\omega = nI_n$.*

Démonstration. D'après le Lemme 1, ω est inversible. La matrice $V_{\omega^{-1}}$ est donc bien définie. Le coefficient de la i-ième ligne et j-ième colonne du produit s'écrit

$$\sum_{k=0}^{n-1} \omega^{(i-1)k} \omega^{-(j-1)k} = \sum_{k=0}^{n-1} \omega^{(i-j)k}.$$

D'après le Lemme 1, cette dernière somme est nulle pour $0 < i - j < n$. Par symétrie des matrices, elle est aussi nulle si $0 < j - i < n$. Les coefficients en dehors de la diagonale sont donc tous nuls. Sur la diagonale, la somme comporte n termes, tous égaux à 1, d'où le résultat. □

Autrement dit, l'interpolation sur les puissances de ω se ramène à une FFT sur les puissances de ω^{-1}, qui est bien primitive d'après le Lemme 1.

4.5. **Conclusion.** Les résultats de cette section sont résumés dans le théorème suivant.

Théorème 3. *Si 2 est inversible dans $\mathbb{A}$ et n une puissance de 2, étant donnée une racine primitive n-ième dans $\mathbb{A}$, le produit de deux polynômes dont la somme des degrés est inférieure à n peut être calculé en $\frac{9}{2} n \log n + O(n)$ opérations dans $\mathbb{A}$. Seuls n des produits sont entre deux éléments de $\mathbb{A}$ qui ne sont pas des puissances de ω.*

Démonstration. La complexité de l'algorithme de multiplication par FFT (Figure 3) est de 3 FFT en degré n, soit $\frac{9}{2} n \log n$ opérations, plus $O(n)$ divisions par n (ce qui est possible puisque 2 est inversible) et $O(n)$ multiplications pour calculer les puissances de ω. Les multiplications de la FFT font intervenir des puissances de ω, les autres sont celles de l'étape de produit point à point, au nombre de n. □

Exercice 10. Montrer que sous les hypothèses du théorème précédent, on peut multiplier ces polynômes en utilisant $3n \log n + O(n)$ additions dans $\mathbb{A}$, $\frac{3}{2} n \log n + O(n)$ multiplications par des puissances de ω, n multiplications arbitraires dans $\mathbb{A}$ et n divisions par n.

Exercice 11. 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 $\mathbb{A}$ une racine n_0-ième primitive de l'unité. Cette racine étant connue, on peut multiplier les polynômes de degré au plus $\frac{1}{2} n - 1$ en $9n \log n + O(n)$ opérations dans $\mathbb{A}$.

Exercice 12. Soit $n = 2^k$, et supposons qu'on dispose d'une racine n-ième primitive de l'unité $\omega \in \mathbb{A}$. Soit P et Q deux polynômes dans $\mathbb{A}[X]$ 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 $\mathbb{A}$.

4.6. **Mais où sont nos racines ?** Même dans le cas favorable où $\mathbb{A}$ est un corps, il n'y existe pas nécessairement toutes les racines primitives de l'unité nécessaires pour appliquer les techniques des algorithmes précédents. Dans le cas particulier des corps finis, on sait donner une réponse précise à cette question d'existence.

Proposition 1. *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 13.

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 groupe multiplicatif $(\mathbb{F}_q \setminus \{0\}, \times)$) alors $\alpha^{(p-1)/n}$ est une racine n-ième primitive de l'unité.

Schönhage-Strassen sur les polynômes

Entrée : F et G de degré $< n$, où $n = 2^k$.

Sortie : $FG \bmod X^{2n} + 1$.

1. Soit $d = 2^{\lceil k/2 \rceil}$. Récrire F et G sous la forme
$$\overline{F}(X, Y) = F_0(X) + F_1(X)Y + \cdots + F_d(X)Y^{d-1},$$
$$\overline{G}(X, Y) = G_0(X) + G_1(X)Y + \cdots + G_d(X)Y^{d-1},$$
de sorte que les F_i et G_i aient degré $< d$ et $F(X) = \overline{F}(X, X^d)$, $G(X) = \overline{G}(X, X^d)$.
2. Calculer $H := \overline{FG}$ dans $\mathbb{B}[Y]$ par FFT, où $\mathbb{B} = \mathbb{A}[X]/(X^{2d} + 1)$ et les produits dans $\mathbb{B}$ sont effectués par le même algorithme.
3. Renvoyer $H(X, X^d)$.

FIGURE 5: L'algorithme de Schönhage-Strassen sur les polynômes.

Ce résultat mène à la notion de *premiers 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 $\ell 2^e + 1$, avec e « suffisamment grand » (qui est appelé *l'exposant* de p). 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}-ième de l'unité (21 en est une) ; on peut donc y multiplier des polynômes de degrés colossaux par l'algorithme en $O(n \log n)$.

Ces premiers sont donc très utiles pour la FFT sur des corps finis. 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 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. Si on choisit au hasard un élément de $\mathbb{F}_p$, la probabilité qu'il soit primitif tend vers $6/\pi^2$, soit plus de $0,6$.

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.

5.1. Racines virtuelles de l'unité.

Le point de départ de l'algorithme est résumé dans le résultat suivant.

Lemme 3. *Si 2 est inversible dans $\mathbb{A}$ et n est une puissance de 2, alors $\omega = X \bmod (X^n + 1)$ est une racine $2n$-ième primitive de l'unité dans $\mathbb{A}[X]/(X^n + 1)$.*

Démonstration. Comme $\omega^n = -1$, ω est bien une racine $2n$-ième de l'unité. Il reste à prouver qu'elle est primitive. D'après la remarque en p. 98, il s'agit de montrer que $\omega^t - 1$ n'est pas diviseur de 0, quel que soit le diviseur strict t de $2n$. Puisque n est une puissance de 2, l'entier t divise n, et donc $\omega^t - 1$ divise l'élément $\omega^n - 1 = -2$ qui est supposé inversible, ce qui conclut. □

5.2. L'algorithme.

L'algorithme est décrit en Figure 5. Il calcule le produit FG modulo $X^{2n} + 1$; ce n'est pas une limitation ici, car le produit a un degré inférieur à $2n$, et cela a l'avantage d'assurer la cohérence dans les appels récursifs. La première étape ne demande pas d'opération arithmétique, et l'évaluation de la dernière étape se fait en complexité linéaire.

Dans $\mathbb{B}$, $\omega = X^2$ est une racine primitive de l'unité d'ordre $2d$, avec $d \approx \sqrt{n}$. On peut donc multiplier des polynômes de degré inférieur à d dans $\mathbb{B}[Y]$ en $O(d \log d)$ opérations dans $\mathbb{B}$ par FFT. D'après le théorème 3, cette opération nécessite :

- $O(d \log d)$ opérations $(+, -)$ dans $\mathbb{B}$, chacune en $O(d)$ opérations dans $\mathbb{A}$;
- $O(d \log d)$ multiplications par une puissance de ω dans $\mathbb{B}$, chacune en $O(d)$ opérations dans $\mathbb{A}$, par simples décalages des indices et éventuels changements de signes ;
- $O(d)$ divisions par d dans $\mathbb{B}$, chacune en $O(d)$ opérations dans $\mathbb{A}$;
- au plus d produits dans $\mathbb{B}$.

Le coût $C(n)$ de cet algorithme obéit donc à l'inégalité

$$C(n) \leq Kn \log n + dC(d)$$

où K est une constante indépendante de n et de $\mathbb{A}$. Posant $c(k) = C(2^k)/2^k$, cette inégalité se récrit

$$c(k) \leq Kk + c(\lceil k/2 \rceil),$$

d'où le résultat final par application du théorème « diviser pour régner ».

Théorème 4. *Soit $\mathbb{A}$ un anneau dans lequel 2 est inversible (d'inverse connu). On peut multiplier des polynômes de $\mathbb{A}[X]$ de degré au plus n en $O(n \log n \log\log n)$ opérations $(+, -, \times)$ dans $\mathbb{A}$.*

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,

$$A = 11111111100101100111 \quad \text{et} \quad B = 11010010000110011101111,$$

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 $\mathbb{Z}[X]$ puis on évalue R en $X = 2$. Pour multiplier P et Q dans $\mathbb{Z}[X]$, il suffit d'effectuer leur produit dans $\mathbb{A}[X] = (\mathbb{Z}/p\mathbb{Z})[X]$, où p est un nombre premier tel que $2D > p > D$. Par le Théorème 4, cette multiplication polynomiale en degré D peut se faire en $O(D \log D \log\log D)$ opérations $(+, -, \times)$ dans $\mathbb{A} = \mathbb{Z}/p\mathbb{Z}$.

Puisque chaque opération de $\mathbb{A}$ 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 négligeable. Ici $AB = R(2) = 11010001001101011110110111011010110110101001$, ou encore, en écriture décimale $AB = 3594211782057$.

Une légère amélioration est possible si l'on choisit pour p un premier de Fourier supérieur à $2D$. Dans notre exemple, on peut prendre $p = 193 = 3 \cdot 2^6 + 1$. En effet, il existe dans $\mathbb{A} = \mathbb{Z}/193\,\mathbb{Z}$ une racine primitive $\omega = 125$, d'ordre $2^6 = 64$, donc supérieur à $2D = 40$. Ce choix mène à un algorithme de complexité binaire $O(D \log^3 D)$.

Une autre approche est de calculer PQ dans $\mathbb{Z}[X]$ par DFT en calculant avec des nombres complexes représentés par des approximations numériques flottantes. Une estimation des erreurs

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.

numériques montre 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 $O(D \log D \log\log D \log\log\log D \ldots)$.

Une meilleure solution est cependant détenue 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 5. *On peut multiplier deux entiers de D chiffres binaires en utilisant $O(D \log D \log\log D)$ opérations binaires.*

En élaborant largement au-delà de ces techniques de FFT, la complexité du produit entier peut être ramenée à $O\big(n \log n \cdot 2^{O(\log^* n)}\big)$, où $\log^* n$ est le nombre d'itérations du logarithme pour ramener n en dessous de 1. Par exemple (pour la base 2), $\log^* 256 = 4$, car $\log 256 = 8$, $\log 8 = 3$, $1 < \log 3 < 2$ et enfin $\log\log 3 < 1$.

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 polynomiale (pour un anneau $\mathbb{A}$) si :

- on peut multiplier les polynômes de $\mathbb{A}[X]$ de degré au plus n en au plus $\mathsf{M}(n)$ opérations dans $\mathbb{A}$;
- 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 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 par exemple d'établir que dans des algorithmes du type « inversion de série formelle par l'opérateur de Newton » (Chap. 4), le coût total est essentiellement celui de la dernière étape.

De la même manière, on est amené 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.

Exercices

Exercice 14. Soit $\mathbb{A}$ un anneau, soit $a \in \mathbb{A}$ et soit P un polynôme de $\mathbb{A}[X]$, de degré au plus n. On se propose de calculer le polynôme $Q(X) = P(X+a)$.

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

Exercice 15. Soit $\mathbb{A}$ un anneau, soit $\kappa, n \in \mathbb{N}$ et soit P un polynôme de $\mathbb{A}[X]$, de degré au plus n.

1. Donner un algorithme direct 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 n'utilisant que $O\big(\mathsf{M}(n\kappa)\big)$ opérations $(+, -, \times)$ dans $\mathbb{A}$.

Exercice 16 (Produit court)**.** Pour deux polynômes $A, B \in \mathbb{Q}[X]$ de degré $< n$, le polynôme $AB \bmod X^n$ est appelé *le produit court* de A et B. Le but de cet exercice est de montrer qu'il est possible de calculer plus efficacement le produit court que le produit entier AB. Le gain est d'un facteur constant.

1. Prouver que tel est le cas avec la multiplication polynomiale naïve.

On suppose que l'on dispose d'un algorithme de multiplication polynomiale qui, en degré n, utilise $\mathsf{M}_\alpha(n) = Kn^\alpha$ opérations dans $\mathbb{Q}$, avec $\alpha > 1$ et $K > 0$.

2. Donner un algorithme de type « diviser pour régner » pour le calcul du produit court de A, B.
3. Montrer que la complexité de cet algorithme est $\frac{1}{2^\alpha - 2}\,\mathsf{M}_\alpha(n) + O(n \log n)$. En déduire l'impact sur la multiplication naïve, et celle de Karatsuba.
4. Modifier la conception de l'algorithme précédent, en découpant en deux sous-problèmes de tailles βn et $(1-\beta)n$, avec $1/2 \le \beta \le 1$ laissé en paramètre. Prouver que le nouvel algorithme utilise
$$\frac{\beta^\alpha}{1 - 2(1-\beta)^\alpha} \cdot \mathsf{M}_\alpha(n) + O(n \log n)$$
opérations dans $\mathbb{Q}$.
5. Montrer que le minimum de la fonction $\beta \mapsto \frac{\beta^\alpha}{1-2(1-\beta)^\alpha}$ est atteint en $\beta_{\min} = 1 - \left(\frac{1}{2}\right)^{\frac{1}{\alpha-1}}$ et en déduire qu'on peut calculer le produit court en $C(\alpha)\,\mathsf{M}_\alpha(n) + O(n \log n)$ opérations dans $\mathbb{Q}$, où
$$C(\alpha) = \frac{\left(2^{\frac{1}{\alpha-1}} - 1\right)^\alpha}{2^{\frac{\alpha}{\alpha-1}} - 2}.$$
6. En admettant que $C(\log_2 3) = 0,8077700579$, que peut-on conclure ?

Exercice 17. Soient $P_1, \ldots, P_t$ des polynômes de $\mathbb{A}[X]$, de degrés $d_1, \ldots, d_t$. Montrer que le produit $P_1 \cdots P_t$ peut s'effectuer en $O(\mathsf{M}(n) \log t)$ opérations dans $\mathbb{A}$, où $n = \sum_i d_i$.

Notes

Le mathématicien russe A. N. Kolmogorov avait conjecturé au début des années 1960 qu'il serait impossible de multiplier deux entiers en un coût binaire sous-quadratique. En 1962 cette conjecture fut infirmée par Karatsuba et Ofman [23]. Une généralisation de l'algorithme de [23] a été proposée peu de temps après par Toom [34] et Cook [8]. L'algorithme de Toom-Cook a une complexité binaire de $O(n \cdot 32^{\sqrt{\log n}})$ pour multiplier deux entiers de taille binaire n ; ce résultat peut être importé dans le monde polynomial (voir Exercice 8 pour une version plus faible).

L'algorithme FFT a une longue histoire, qui remonte à Gauss [21, 10, 12]. Il s'agit d'un progrès algorithmique très célèbre : Dongarra et Sullivan [14] le placent parmi les dix algorithmes qui ont marqué le plus le développement des sciences de l'ingénieur du 20-ième siècle. L'article fondateur de Cooley et Tukey [11] est l'un des plus cités en informatique [4]. La variante de la DFT décrite dans ce chapitre, appelée *decimation-in-frequency* en anglais, est due à Gentleman et Sande [19] ; il s'agit d'une version duale (au sens du principe de transposition de Tellegen évoqué au Chapitre 12) de l'algorithme *decimation-in-time* de [11]. Les livres [3, 29, 36] et les articles [15, 1, 16] constituent de bonnes références pour le lecteur désireux de se familiariser avec la myriade de techniques de type FFT ; en particulier, l'article [16] décrit l'une des implantations les plus efficaces de la transformée de Fourier complexe (appelée FFTW, pour *Fastest Fourier Transform in the West.*)

Une avancée importante a été la découverte de Schönhage et Strassen [32] 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 [28]. Le cas particulier de la caractéristique 2 est traité dans l'article [31]. Plus récemment,

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

Cantor et Kaltofen [5] ont donné un algorithme qui multiplie des polynômes de degré n sur une algèbre $\mathbb{A}$ (non nécessairement commutative, ni associative) en $O(n \log n \log\log n)$ opérations dans $\mathbb{A}$.

La borne de complexité binaire $O\big(n \log n \cdot 2^{O(\log^* n)}\big)$ mentionnée en fin de section 6, pour multiplier des entiers de n chiffres, a été obtenue dans [17, 18, 13]. À ce jour, on ne connaît pas de borne analogue pour la complexité aritmétique de la multiplication polynomiale en degré n.

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 [26] a donné une première réponse partielle, en montrant que dans un modèle simplifié où $\mathbb{A} = \mathbb{C}$ 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. Plus récemment, Bürgisser et Lotz [4] ont étendu ce type de borne à la multiplication dans $\mathbb{C}[X]$. Concernant les entiers, Cook et Aanderaa [9] 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. Cette borne a été améliorée à $\Omega(n \log n)$ dans [30] pour un modèle restreint (*multiplication en ligne*).

Malgré la simplicité de leur idée de base, les algorithmes de Karatsuba et de Toom-Cook soulèvent encore des questions délicates ; les travaux récents [37, 25, 7, 2] traitent diverses généralisations (degrés déséquilibrés ou non-puissances, caractéristique positive de l'anneau des scalaires). Par exemple, connaître le nombre minimum de multiplications $\mathsf{sm}(n)$ (resp. $\mathsf{sm}_p(n)$) suffisantes pour multiplier des polynômes de degré donné n sur un anneau arbitraire (resp. sur un anneau de caractéristique p) est un problème important et difficile. Pour les premières valeurs de n, les meilleures bornes connues actuellement, dues à [25], sont $\mathsf{sm}(1) \leq 3$, $\mathsf{sm}(2) \leq 6$, $\mathsf{sm}(3) \leq 9$, $\mathsf{sm}(4) \leq 13$, $\mathsf{sm}(5) \leq 17$ et $\mathsf{sm}(6) \leq 22$. L'optimalité de ces bornes est un problème difficile. Pour $\mathsf{sm}_p(n)$, on sait par exemple que $\mathsf{sm}_2(1) = 3, \mathsf{sm}_2(2) = 6, \mathsf{sm}_2(3) = 9$ [22] et que $\mathsf{sm}_7(4) = 10$, $\mathsf{sm}_5(4) \leq 11$ [6].

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'exercice 16, inspiré de l'article [27], apporte une réponse positive dans le 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 question liée, le calcul du *produit médian* [20], sera abordée au Chapitre 12.

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 [35] 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 a une courbe de complexité « plus lisse » que la DFT.

Le produit de t polynômes de degrés arbitraires $d_1, \ldots, d_t$ peut s'effectuer [33] en $O(\mathsf{M}(n)(1 + H(d_1, \ldots, d_t)))$ opérations, où $H(d_1, \ldots, d_t) = -\sum_i \frac{d_i}{n} \cdot \log \frac{d_i}{n}$ est l'*entropie* du vecteur $(d_1/n, \ldots, d_n/n)$. La multiplication des polynômes à plusieurs variables peut se ramener à la multiplication univariée grâce à une technique connue sous le nom d'*astuce de Kronecker* [24].

Bibliographie

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

[2] Bodrato (Marco) and Zanoni (Alberto). – Integer and polynomial multiplication : towards optimal Toom-Cook matrices. In *ISSAC'07*, pp. 17–24. – ACM, New York, 2007.

[3] Brigham (E. Oran). – *The fast Fourier transform*. – Prentice-Hall, 1974, xiii+252p.

[4] Bürgisser (Peter) and Lotz (Martin). – Lower bounds on the bounded coefficient complexity of bilinear maps. *J. ACM*, vol. 51, n3, 2004, pp. 464–482.

[5] Cantor (D. G.) and Kaltofen (E.). – On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, vol. 28, n7, 1991, pp. 693–701.

[6] Cenk (M.), Koc (C. K.), and Ozbudak (F.). – Polynomial multiplication over finite fields using field extensions and interpolation. In *ARITH '09*. pp. 84–91. – IEEE Computer Society, 2009.

[7] Chung (Jaewook) and Hasan (M. Anwar). – Asymmetric squaring formulae. In *ARITH '07 : Proc. of the 18th IEEE Symp. on Computer Arithmetic*. pp. 113–122. – IEEE, 2007.

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

[9] 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.

[10] Cooley (James W.). – How the FFT gained acceptance. In *A history of scientific computing (Princeton, NJ, 1987)*, pp. 133–140. – ACM, New York, 1990.

[11] 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.

[12] Cooley (James W.) and Tukey (John W.). – On the origin and publication of the FFT paper. *Current Contents*, vol. 33, n51–52, 1993, pp. 8–9.

[13] De (A.), Kurur (P. P.), Saha (C.), and Saptharishi (R.). – Fast integer multiplication using modular arithmetic. In *STOC'08*. pp. 499–506. – ACM, New York, NY, USA, 2008.

[14] Dongarra (J.) and Sullivan (F.). – Top ten algorithms. *Computing in Science & Engineering*, vol. 2, n1, 2000, pp. 22–23.

[15] Duhamel (P.) and Vetterli (M.). – Fast Fourier transforms : a tutorial review and a state of the art. *Signal Processing. An Interdisciplinary Journal*, vol. 19, n4, 1990, pp. 259–299.

[16] Frigo (M.) and Johnson (S. G). – The Design and Implementation of FFTW3. *Proceedings of the IEEE*, vol. 93, n2, 2005, pp. 216–231.

[17] Fürer (Martin). – Faster integer multiplication. In *STOC'07—Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pp. 57–66. – ACM, New York, 2007.

[18] Fürer (Martin). – Faster integer multiplication. *SIAM J. Comput.*, vol. 39, n3, 2009, pp. 979–1005.

[19] Gentleman (W. M.) and Sande (G.). – Fast Fourier Transforms: for fun and profit. In *AFIPS'66*. pp. 563–578. – ACM, New York, NY, USA, 1966.

[20] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, vol. 14, n6, March 2004, pp. 415–438.

[21] Heideman (Michael T.), Johnson (Don H.), and Burrus (C. Sidney). – Gauss and the history of the fast Fourier transform. *Arch. Hist. Exact Sci.*, vol. 34, n3, 1985, pp. 265–277.

[22] Kaminski (Michael). – A lower bound for polynomial multiplication. *Theoret. Comput. Sci.*, vol. 40, n2-3, 1985, pp. 319–322.

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

[24] Moenck (Robert T.). – Another polynomial homomorphism. *Acta Informat.*, vol. 6, n2, 1976, pp. 153–169.

[25] Montgomery (Peter L.). – Five, six, and seven-term Karatsuba-like formulae. *IEEE Trans. Comput.*, vol. 54, n3, 2005, pp. 362–369.

[26] Morgenstern (Jacques). – Note on a lower bound of the linear complexity of the fast Fourier transform. *J. Assoc. Comput. Mach.*, vol. 20, 1973, pp. 305–306.

[27] Mulders (Thom). – On short multiplications and divisions. *Applicable Algebra in Engineering, Communication and Computing*, vol. 11, n1, 2000, pp. 69–88.

[28] 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, n2, 1980, pp. 205–215.

[29] Nussbaumer (Henri J.). – *Fast Fourier transform and convolution algorithms*. – Springer-Verlag, Berlin, 1981, *Springer Series in Information Sciences*, vol. 2, x+248p.

[30] Paterson (M. S.), Fischer (M. J.), and Meyer (A. R.). – An improved overlap argument for on-line multiplication. In *Complexity of computation*, pp. 97–111. – AMS, 1974.

[31] Schönhage (A.). – Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Informat.*, vol. 7, n4, 1976/77, pp. 395–398.

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

[33] Strassen (V.). – The computational complexity of continued fractions. *SIAM J. Comput.*, vol. 12, n1, 1983, pp. 1–27.

[34] 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.

[35] van der Hoeven (Joris). – The truncated Fourier transform and applications. In *ISSAC'04*, pp. 290–296. – ACM, New York, 2004.

[36] Van Loan (Charles). – *Computational frameworks for the fast Fourier transform*. – SIAM, Philadelphia, PA, 1992, *Frontiers in Applied Mathematics*, vol. 10, xiv+273p.

[37] Weimerskirch (André) and Paar (Christof). – Generalizations of the Karatsuba algorithm for efficient implementations, 2006. Cryptology ePrint Archive : Report 2006/224, available at `http://eprint.iacr.org/2006/224`.

CHAPITRE 3

Algèbre linéaire dense : de Gauss à Strassen

Résumé

L'algèbre linéaire est au cœur de nombreux problèmes algorithmiques. La multiplication matricielle usuelle et la méthode du pivot de Gauss ont une complexité arithmétique cubique en la taille. Dans ce chapitre, nous montrons que pour la plupart des questions d'algèbre linéaire dense – multiplication, inversion, déterminant, résolution de système – il existe des algorithmes plus efficaces, de complexité strictement sous-cubique.

1. Introduction

En mathématiques, il est de coutume de considérer qu'un problème est rendu trivial lorsqu'il est ramené à une question d'algèbre linéaire. Du point de vue calculatoire se pose cependant la question de l'efficacité des opérations matricielles.

1.1. L'algorithmique des matrices : tentative de classification.

Les problèmes algorithmiques en algèbre linéaire cachent des difficultés très subtiles. En général, les premières questions que l'on est amené à se poser en rapport avec la manipulation des matrices concernent le calcul efficace du produit matrice-matrice, du produit matrice-vecteur, de l'inverse, ainsi que la résolution de systèmes.

Les réponses à ces questions varient fortement selon le type de matrices considérées. Une classification possible est la suivante.

Les matrices denses. Ce sont les matrices quelconques, sans aucune structure particulière. Nous verrons que l'algorithmique des matrices denses peut se ramener essentiellement au produit matriciel, dont la complexité est une question extrêmement délicate.

Les matrices creuses. Beaucoup de problèmes linéaires se formulent en termes de matrices possédant un nombre important d'éléments nuls, dites « creuses ». Dans ce cas, l'algorithmique dense est inappropriée ; il est possible de tirer parti de la forme creuse en utilisant des outils mieux adaptés, à base de récurrences linéaires.

Les matrices structurées. Enfin, il existe des familles de matrices particulières, souvent associées à des applications linéaires entre espaces de polynômes, telles que les matrices de type Sylvester pour le résultant, de type Vandermonde pour l'évaluation et l'interpolation, de type Toeplitz et Hankel pour l'approximation de Padé et de Padé-Hermite, etc ... Pour ces types de matrices clairement identifiés, on développe soit une algorithmique ad-hoc, soit une théorie unifiée reposant sur le concept de « rang de déplacement ».

Schématiquement, l'algorithmique des matrices denses est la plus lente, de complexité située entre $O(n^2)$ et $O(n^3)$, en taille n. La complexité du calcul avec les matrices creuses est de l'ordre de $O(n^2)$, alors que celle des matrices structurées est en $O(n)$, à des facteurs logarithmiques près.

L'algorithmique des matrices denses constitue l'objet d'étude de ce chapitre. Les matrices creuses seront abordées brièvement au Chapitre 10 et les matrices structurées feront l'objet d'une étude plus approfondie au Chapitre 11.

Nous allons travailler avec un corps effectif noté $\mathbb{K}$ et avec l'algèbre des matrices carrées $\mathcal{M}_n(\mathbb{K})$ à coefficients dans $\mathbb{K}$. Signalons toutefois que la plupart des résultats de ce chapitre s'étendent au cas où le corps $\mathbb{K}$ est remplacé par un anneau effectif $\mathbb{A}$, et les matrices carrées par des matrices rectangulaires à coefficients dans $\mathbb{A}$.

1.2. **Résultat principal.** Les questions qui seront étudiées, ou simplement évoquées, dans ce chapitre sont celles de la complexité du calcul du produit matriciel dans $\mathcal{M}_n(\mathbb{K})$, de l'inverse et du déterminant, du polynôme caractéristique ou minimal, de la résolution de systèmes linéaires, ou encore de la mise sous diverses « formes canoniques » (échelon, LUP, Frobenius, ...)

Pour ces questions, on dispose d'une première famille d'algorithmes « naïfs », reposant sur l'application directe des définitions ou des propriétés mathématiques. Si l'algorithme naïf de multiplication a une complexité raisonnable –cubique– en la taille, dans d'autres cas, l'emploi des algorithmes naïfs est vivement déconseillé. Par exemple, si $A = (a_{i,j})_{i,j=1}^n$ est une matrice de $\mathcal{M}_n(\mathbb{K})$, on n'utilisera pas la définition

$$\det(A) = \sum_{\sigma \in \mathcal{S}_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)}$$

pour le calcul de son déterminant ; en effet, cela mènerait à un algorithme de complexité $O(n \cdot n!)$, donc hyper-exponentielle en la taille de A. De même, la définition du rang d'une matrice comme la taille maximale d'un mineur non-nul ne se prête pas directement, via une recherche exhaustive, à une algorithmisation efficace. Dans un registre légèrement différent, les formules de Cramer pour résoudre un système linéaire ne sont pas d'une grande utilité pratique (en revanche, elles servent à borner les tailles des solutions, et s'avèrent utiles dans les estimations de complexité).

Une seconde classe d'algorithmes, basés sur la *méthode d'élimination de Gauss*, permet de résoudre de manière raisonnablement efficace la plupart des problèmes évoqués plus haut. Appliquée à une matrice A de $\mathcal{M}_n(\mathbb{K})$, cette méthode fournit des algorithmes de complexité $O(n^3)$ pour le calcul de $\operatorname{rg}(A)$, de $\det(A)$, de A^{-1} (si A est inversible), celui d'une forme échelonnée et d'une décomposition LUP de A, ainsi que pour la résolution d'un système linéaire de matrice A.

Le résultat principal de ce chapitre est que l'on peut faire mieux, et qu'il existe une constante $2 \leq \omega < 3$ (dépendante *a priori* du corps de base $\mathbb{K}$), qui contrôle la complexité du produit matriciel et de toutes les opérations d'algèbre linéaire.

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

Définition 1 (Exposant de la multiplication matricielle)**.** On dit que le réel θ est un *exposant faisable* pour la multiplication de matrices sur le corps $\mathbb{K}$ s'il existe un algorithme de complexité arithmétique $O(n^\theta)$ pour multiplier deux matrices arbitraires de $\mathcal{M}_n(\mathbb{K})$. On définit $\omega_{\text{mul}} = \inf\{\theta \mid \theta \text{ est un exposant faisable}\}$.

A priori, on devrait mettre en indice le corps $\mathbb{K}$ pour les exposants θ et ω_{mul}. Les résultats présentés dans la suite sont cependant indépendants de $\mathbb{K}$ [1]. L'algorithme naïf pour le produit est de complexité $O(n^3)$, donc $\theta = 3$ est un exposant faisable et $\omega_{\text{mul}} \leq 3$. D'un autre côté, ω_{mul} est forcément au moins égal à 2 : le nombre d'éléments d'une matrice est n^2 et il faut bien autant d'opérations pour l'engendrer.

Ensuite, on peut définir de la même manière les exposants de tous les autres problèmes, de la forme ω_{inv}, ω_{polcar}, $\omega_{\det}$, ω_{LUP}, etc ... On a alors le résultat suivant.

Théorème 1. *Les exposants ω_{mul}, ω_{inv}, ω_{polcar}, $\omega_{\det}$, ω_{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. Nous nous contenterons 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 $\log_2(7) < 2{,}81$.

1. On peut montrer que la constante ω_{mul} ne dépend que de la caractéristique du corps de base $\mathbb{K}$ et il est conjecturé qu'elle est même entièrement indépendante du corps $\mathbb{K}$.

Nous montrerons également que $\omega_{\text{det}} \leq \omega_{\text{polcar}} \leq \omega_{\text{mul}} = \omega_{\text{inv}}$ et laisserons en exercice (Exercice 11) la preuve de l'inégalité $\omega_{\text{inv}} \leq \omega_{\text{det}}$ qui permet de conclure que cette suite d'inégalités est en fait une suite d'égalités.

1.3. **Applications.** Les problèmes d'algèbre linéaire mentionnés ci-dessus sont très fréquemment rencontrés en pratique ; les résultats du Théorème 1 seront invoqués à plusieurs reprises dans ce cours. Par exemple, l'algorithme de Beckermann-Labahn (Chapitre 9) pour le calcul d'approximants de Padé-Hermite repose ultimement sur des produits de matrices de polynômes. C'est aussi le cas de l'algorithme de Storjohann pour la résolution de systèmes linéaires à coefficients polynomiaux (Chapitre 14), et de l'algorithme de recherche de solutions séries d'équations algébriques (Chapitre 4) ou différentielles linéaires (Chapitre 15).

L'algèbre linéaire est au cœur de nombreux autres problèmes algorithmiques qui ne seront pas abordés dans ce cours ; c'est le cas de la résolution de systèmes polynomiaux, de la sommation et de l'intégration symboliques, de la factorisation des entiers et des polynômes sur un corps fini, ou encore du logarithme discret en cryptanalyse.

Mentionnons enfin qu'une large partie des ordinateurs du 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 (le système de classement PageRank utilisé par le moteur de recherche Google repose sur la recherche d'un vecteur propre d'une gigantesque matrice creuse).

2. Multiplication de matrices

Tout comme le produit d'entiers et de polynômes, la multiplication matricielle est une opération fondamentale en calcul formel, dont l'étude de complexité s'avère être un problème hautement non trivial.

L'algorithme naïf pour effectuer le produit d'une matrice de taille $m \times n$ par un vecteur utilise $O(mn)$ opérations arithmétiques : mn multiplications et $m(n-1)$ additions. Un résultat de Winograd affirme qu'en général on ne peut pas faire mieux : par exemple, pour les matrices réelles dont les coefficients sont algébriquement indépendants sur $\mathbb{Q}$, l'algorithme naïf *est optimal*. Puisque le produit matriciel peut s'interpréter comme une succession de produits matrice-vecteur, une question naturelle est de savoir si la multiplication *naïve* de deux matrices est aussi quasi-optimale.

La réponse à cette question, et les résultats les plus importants de cette section, classés par ordre chronologique de leur découverte, sont regroupés dans le théorème suivant.

Théorème 2 (la multiplication naïve n'est pas optimale)**.** *Soit $\mathbb{K}$ un corps commutatif. On peut multiplier deux matrices de $\mathcal{M}_n(\mathbb{K})$ en*

(a) $O(n^3)$ *opérations dans $\mathbb{K}$, par l'algorithme naïf.*

(b) $n^2\lceil \frac{n}{2} \rceil + 2n\lfloor \frac{n}{2} \rfloor \simeq \frac{1}{2}n^3 + n^2$ *multiplications dans $\mathbb{K}$.*

(c) $n^2\lceil \frac{n}{2} \rceil + (2n-1)\lfloor \frac{n}{2} \rfloor \simeq \frac{1}{2}n^3 + n^2 - \frac{n}{2}$ *multiplications dans $\mathbb{K}$ si la caractéristique de $\mathbb{K}$ est différente de 2 et si la division par 2 est gratuite.*

(d) $O(n^{\log_2(7)}) \simeq O(n^{2,81})$ *opérations dans $\mathbb{K}$.*

Ces résultats restent valables sur un anneau commutatif $\mathbb{A}$; pour (a) et (d) l'hypothèse de commutativité peut même être supprimée. La partie (b) est due à Winograd et fut historiquement la première amélioration de l'algorithme naïf. Elle réduit de moitié le nombre de multiplications, en augmentant en revanche le nombre d'additions. Cela constitue déjà un progrès pour une large classe d'anneaux $\mathbb{A}$ dans lesquels la multiplication est plus coûteuse que l'addition [2]. Par exemple, la technique du « scindage binaire » qui sera utilisée au Chapitre 13 pour le calcul d'un terme d'une récurrence, ou encore pour le calcul du développement décimal de e, requiert de multiplier des matrices contenant de gros nombres entiers.

L'amélioration (c) a été obtenue par Waksman, et implique qu'on peut effectuer le produit de deux matrices 2×2 en 7 multiplications. L'inconvénient commun des algorithmes de Winograd et de Waksman est l'utilisation de la commutativité de $\mathbb{K}$, qui devient un obstacle à une application récursive, ne permettant pas d'améliorer (la borne supérieure 3 sur) l'exposant ω_{mul}. Beaucoup

2. C'est par exemple le cas de $\mathbb{Z}$ et de $\mathbb{Z}[X]$, mais pas de $\mathbb{Q}$ ou de $\mathbb{Q}(X)$.

pensaient que tout résultat de type (b) et (c) serait optimal, au sens que $\frac{1}{2}n^3$ multiplications dans $\mathbb{K}$ seraient nécessaires pour le produit dans $\mathcal{M}_n(\mathbb{K})$. Ceci fut contredit par la découverte par Strassen de (d) ; ce résultat découle d'un fait d'une simplicité déconcertante, mais d'une portée considérable : deux matrices 2×2 peuvent être multipliées en 7 multiplications même si $\mathbb{K}$ n'est pas commutatif.

La non-optimalité de la multiplication matricielle naïve justifie l'introduction de la définition suivante.

Définition 2 (Fonction de multiplication matricielle)**.** On dit que l'application $\mathsf{MM} : \mathbb{N} \to \mathbb{N}$ est une *fonction de multiplication matricielle* (pour un corps $\mathbb{K}$) si :

- on peut multiplier des matrices de $\mathcal{M}_n(\mathbb{K})$ en au plus $\mathsf{MM}(n)$ opérations dans $\mathbb{K}$;
- MM est croissante et vérifie l'inégalité $\mathsf{MM}(n/2) \leq \mathsf{MM}(n)/4$ pour $n \in \mathbb{N}$.

Ainsi, le Théorème 2 montre que les fonctions $n \mapsto n^3$ ou $n \mapsto n^{\log_2(7)}$ sont (à des constantes près) des fonctions de multiplication matricielles. Comme dans le cas de la fonction de multiplication polynomiale $\mathsf{M}(\,)$ introduite au Chapitre 2, 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 matrices.

La seconde condition de la Définition 2 établit la cohérence avec la Définition 1 : si $\mathsf{MM}(n) = c \cdot n^\theta$, pour une certaine constante $c > 0$, alors θ doit être supérieur ou égal à 2. Cette condition permettra de montrer que dans des algorithmes de type « diviser pour régner » pour les matrices, le coût total est essentiellement celui du dernier appel récursif.

Le reste de la section est dédié à la preuve du Théorème 2.

2.1. **Multiplication naïve.** Commençons par décrire l'algorithme « cubique » de multiplication. Étant données deux matrices $A = (a_{i,j})_{i,j=1}^n$ et $X = (x_{i,j})_{i,j=1}^n$ de $\mathcal{M}_n(\mathbb{K})$, leur produit s'écrit $R = AX$ avec

$$r_{i,k} = \sum_{j=1}^{n} a_{i,j} x_{j,k}.$$

Le coût nécessaire pour obtenir un élément de R est donc de n multiplications et $n-1$ additions, de sorte que la complexité totale est en $O(n^3)$.

2.2. **Algorithme de Winograd.** Il est possible de diviser essentiellement par deux le nombre de multiplications scalaires suffisantes pour le calcul du produit AX. Ceci montre au passage que la multiplication matricielle naïve n'est pas optimale. Supposons pour simplifier que n est pair, $n = 2k$. L'algorithme de Winograd, repose sur l'identité suivante, héritière de celle de Karatsuba : si ℓ est un vecteur ligne $[a_1 \cdots a_n]$ et c est un vecteur colonne ${}^t[x_1 \cdots x_n]$, alors le produit scalaire $\langle \ell, c \rangle$ s'écrit

$$\langle \ell, c \rangle = (a_1 + x_2)(a_2 + x_1) + \cdots + (a_{2k-1} + x_{2k})(a_{2k} + x_{2k-1}) - \sigma(\ell) - \sigma(c), \tag{1}$$

où $\sigma(\ell) = a_1 a_2 + \cdots + a_{2k-1} a_{2k}$ et $\sigma(c) = x_1 x_2 + \cdots + x_{2k-1} x_{2k}$. Notons que $\sigma(\ell)$ se calcule en $k = n/2$ multiplications et $k - 1 = n/2 - 1$ additions.

Si maintenant $\ell_1, \ldots, \ell_n$ sont les lignes de A et $c_1, \ldots, c_n$ sont les colonnes de X, l'élément (i, j) du produit AX vaut $\langle \ell_i, c_j \rangle$. L'idée est alors de précalculer les quantités $\sigma(\ell_i)$ et $\sigma(c_i)$ pour $1 \leq i \leq n$; ce précalcul demande $n(\frac{n}{2} + \frac{n}{2}) = n^2$ multiplications et $n(\frac{n}{2} - 1 + \frac{n}{2} - 1) = n^2 - 2n$ additions. Une fois déterminés les $\sigma(\ell_i)$ et $\sigma(c_i)$, on peut calculer l'ensemble des éléments (i, j) de AX grâce à la formule (1) en $n^2 \cdot \frac{n}{2} = \frac{1}{2}n^3$ multiplications et $n^2 \cdot (n + (\frac{n}{2} - 1) + 2) = \frac{3}{2}n^3 + n^2$ additions. Au total, le coût de la multiplication de l'algorithme de Winograd est de $\frac{1}{2}n^3 + n^2$ multiplications et $\frac{3}{2}n^3 + 2n^2 - 2n$ additions. Ainsi, on a essentiellement transformé $n^3/2$ multiplications en additions, ce qui est appréciable.

2.3. **Algorithme de Waksman.** Pour effectuer le produit de matrices 2×2

$$R = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & t \end{bmatrix},$$

l'algorithme de Winograd utilise 8 multiplications, en écrivant

$$R = \begin{bmatrix} (a+z)(b+x) - ab - zx & (a+t)(b+y) - ab - ty \\ (c+z)(d+x) - cd - zx & (c+t)(d+y) - cd - ty \end{bmatrix}.$$

Il n'apporte donc aucun gain par rapport à l'algorithme naïf dans cette taille.

L'algorithme suivant, dû à Waksman, peut être vu comme une amélioration de l'algorithme de Winograd. Il effectue le produit de matrices 2×2 en 7 opérations, dès lors que dans le corps de base $\mathbb{K}$ l'élément 2 est inversible et que la division par 2 y est gratuite. L'idée est d'écrire les éléments $r_{i,j}$ de R comme

$$\begin{array}{rcl} r_{1,1} & = & \frac{1}{2}(A_1 - A_2) \\ r_{1,2} & = & \frac{1}{2}(B_1 - B_2) \\ r_{2,1} & = & \frac{1}{2}(C_1 - C_2) \\ r_{2,2} & = & \frac{1}{2}(D_1 - D_2) \end{array}$$

où

$$\begin{array}{ll} A_1 = (a+z)(b+x), & A_2 = (a-z)(b-x), \\ B_1 = (a+t)(b+y), & B_2 = (a-t)(b-y), \\ C_1 = (c+z)(d+x), & C_2 = (c-z)(d-x), \\ D_1 = (c+t)(d+y), & D_2 = (c-t)(d-y), \end{array}$$

et de faire observer que, grâce à l'identité

$$(C_1 + C_2) + (B_1 + B_2) = (A_1 + A_2) + (D_1 + D_2) = 2(ab + cd + xz + ty),$$

l'élément D_2 est une combinaison linéaire des sept autres produits $A_1, A_2, \ldots, D_1$, et donc n'a pas besoin d'être obtenu lui aussi par un produit.

L'algorithme qui s'en déduit, tout comme celui de Winograd, ne peut être utilisé de manière récursive : la preuve des formules sous-jacentes repose sur le fait que les éléments du corps de base $\mathbb{K}$ commutent ($bz = zb, bt = tb, \ldots$). Lors des appels récursifs, les objets à multiplier seraient eux-mêmes des matrices, pour lesquelles la loi de commutation n'est plus vérifiée. On dit donc que les algorithmes de Winograd et de Waksman sont des algorithmes *commutatifs*.

Exercice 1. Finir la preuve des parties (b) et (c) du Théorème 2, en généralisant les arguments précédents à des matrices de taille n, où $n \in \mathbb{N}$ n'est pas nécessairement pair.

2.4. **Algorithme de Strassen.** L'algorithme de Strassen fut le premier à descendre en-dessous de $O(n^3)$. Tout comme l'algorithme de Karatsuba pour les polynômes, il repose sur le gain d'une multiplication pour les matrices 2×2, qui devient un gain dans l'exposant lors d'une application récursive. La non-commutativité des matrices se traduit en revanche par une contrainte supplémentaire : afin de se prêter à un emploi récursif, le nouvel algorithme doit, à la différence de l'algorithme de Waksman, satisfaire à une contrainte de non-commutativité.

Soient donc A et X des matrices de taille 2 à multiplier par un algorithme non-commutatif. Intuitivement, cela se traduit par le fait que lors des multiplications utilisées par un tel algorithme, les éléments a, b, c, d de A doivent rester à gauche, et les éléments x, y, z, t de X à droite (cela n'était pas le cas pour les algorithmes de Winograd et de Waksman, qui *mélangent* les éléments de A et X).

L'algorithme naïf est bien non-commutatif, mais demande d'effectuer 8 multiplications. L'algorithme de Strassen que nous allons présenter revient à

- calculer des combinaisons linéaires des éléments de A et des éléments de X,
- effectuer **7** produits de ces combinaisons linéaires,
- recombiner les résultats pour obtenir le produit AX.

Explicitement, les formules à appliquer sont données dans les étapes 3 et 4 de l'algorithme présenté en Figure 1.

Effectuons maintenant le pas récursif. Quitte à rajouter des colonnes et des lignes nulles, on peut supposer que les matrices A et X sont de taille $n = 2^k$, avec $k \in \mathbb{N}$. On procède à une découpe en blocs de A et de X :

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \qquad X = \begin{bmatrix} x & y \\ z & t \end{bmatrix}.$$

Dans cette écriture a, b, c, d et x, y, z, t sont donc des matrices carrées de taille $n/2$.

Le point-clé est le suivant : grâce à la non-commutativité, les formules données ci-dessus pour le cas 2×2 s'appliquent toujours si a, b, c, d et x, y, z, t sont des matrices. Ainsi, elles permettent de ramener le produit en taille n à 7 produits en taille $n/2$, plus un certain nombre C (ici $C = 18$)

Multiplication matricielle rapide

Entrée : Deux matrices $A, X \in \mathcal{M}_n(\mathbb{K})$, avec $n = 2^k$.

Sortie : Le produit AX.

1. Si $n = 1$, renvoyer AX.
2. Décomposer $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ et $X = \begin{bmatrix} x & y \\ z & t \end{bmatrix}$, avec a, b, c, d et x, y, z, t dans $\mathcal{M}_{n/2}(\mathbb{K})$.
3. Calculer récursivement les produits
$$\left| \begin{array}{lcl} q_1 & = & a(x+z) \\ q_2 & = & d(y+t) \\ q_3 & = & (d-a)(z-y) \\ q_4 & = & (b-d)(z+t) \\ q_5 & = & (b-a)z \\ q_6 & = & (c-a)(x+y) \\ q_7 & = & (c-d)y. \end{array} \right.$$
4. Calculer les sommes
$$\left| \begin{array}{lcl} r_{1,1} & = & q_1 + q_5 \\ r_{1,2} & = & q_2 + q_3 + q_4 - q_5 \\ r_{2,1} & = & q_1 + q_3 + q_6 - q_7 \\ r_{2,2} & = & q_2 + q_7. \end{array} \right.$$
5. Renvoyer $\begin{bmatrix} r_{1,1} & r_{1,2} \\ r_{2,1} & r_{2,2} \end{bmatrix}$.

Figure 1: Algorithme de Strassen pour multiplier deux matrices.

d'additions en taille $n/2$, utilisées pour fabriquer les combinaisons linéaires des blocs de A et de X, et celles des produits q_i.

La complexité $\mathsf{S}(n)$ de l'algorithme de Strassen satisfait à la récurrence

$$\mathsf{S}(n) \leq 7\,\mathsf{S}\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right)^2.$$

En invoquant le lemme « diviser pour régner » avec les choix $m = 7, p = s = 2$, $\kappa = 1$ et $q = 4$, on déduit l'inégalité

$$\mathsf{S}(n) \leq \left(1 + \frac{C}{3}\right) n^{\log_2(7)},$$

qui prouve la partie (d) du Théorème 2.

L'idée de base de l'algorithme de Strassen est tout à fait générale : améliorer la multiplication en petite taille permet d'améliorer l'exposant.

Exercice 2. Supposons que l'on sache multiplier deux matrices de taille r à coefficients dans un corps $\mathbb{K}$ en r^θ multiplications non-commutatives dans $\mathbb{K}$ et un nombre arbitraire d'additions dans $\mathbb{K}$. Montrer que θ est un exposant faisable pour la multiplication des matrices à coefficients dans $\mathbb{K}$.

En pratique, il peut être intéressant de ne pas descendre récursivement jusqu'à la multiplication des matrices 1×1, mais d'arrêter la récursion à une taille $s > 1$ et d'utiliser l'algorithme naïf (ou tout autre algorithme, même commutatif!) en cette taille. Cela permet d'optimiser la constante du $O(\,)$.

Exercice 3. Soit n une puissance de 2. Établir un algorithme hybride, qui fait appel à l'algorithme de Strassen pour $n > 2^d$ et à l'algorithme naïf pour $n \leq 2^d$. Montrer que la complexité $\mathsf{MM}(n)$ de cet algorithme vérifie $\mathsf{MM}(n) \leq \mu(d) n^{\log_2(7)}$ pour tout $n \geq 2^d$, où $\mu(d)$ est une fonction qui dépend uniquement de d. Trouver la valeur de d qui minimise $\mu(d)$.

L'algorithme de Strassen s'applique aussi au cas où le corps de base $\mathbb{K}$ est remplacé par un anneau $\mathbb{A}$. Lorsque cet anneau $\mathbb{A}$ est une extension de $\mathbb{K}$, la question est d'estimer sa complexité en termes de nombre d'opérations dans $\mathbb{K}$.

Exercice 4. Soit $\mathbb{K}$ un corps et soient d et n deux entiers naturels strictement positifs. Estimer la complexité de l'algorithme de multiplication de Strassen appliqué à deux matrices carrées de taille n, dont les éléments sont des polynômes de degré au plus d de $\mathbb{K}[X]$.

Plus généralement, tout algorithme de multiplication dans $\mathcal{M}_n(\mathbb{K})$ utilisant $\mathsf{MM}(n) = O(n^\theta)$ opérations dans $\mathbb{K}$ fournit par application à $\mathbb{A} = \mathbb{K}[X]/(X^{d+1})$ un algorithme de multiplication de matrices polynomiales de degré d de complexité $\mathsf{MM}(n,d) = O(n^\theta\,\mathsf{M}(d))$ opérations de $\mathbb{K}$. Un algorithme de meilleure complexité permettant d'obtenir $\mathsf{MM}(n,d) = O(n^\theta d + n^2\,\mathsf{M}(d))$ sera presenté au Chapitre 6.

2.5. **Interprétation des formules de Strassen.** Il n'est pas immédiat de donner une intuition aux formules de Strassen (à la différence de celle de Karatsuba, qui repose de manière cachée sur l'évaluation de polynômes en $0, 1, +\infty$). Nous allons présenter un raisonnement qui permet de motiver les formules de Strassen et de les généraliser. L'explication appartient à Fiduccia ; elle est postérieure à la découverte de l'algorithme [3].

Le point de départ est l'observation que la recherche d'un algorithme non-commutatif pour effectuer le produit de matrices $R = AX$ avec

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \text{et} \qquad X = \begin{bmatrix} x & y \\ z & t \end{bmatrix}$$

revient à donner un algorithme pour la multiplication matrice-vecteur Mv, où

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \qquad \text{et} \qquad v = \begin{bmatrix} x \\ z \\ y \\ t \end{bmatrix}.$$

La suite de l'argument repose sur la remarque suivante : toute matrice (de taille arbitraire) de la forme

$$\begin{bmatrix} & & \\ m\alpha & & n\alpha \\ & & \\ m\alpha & & n\alpha \\ & & \end{bmatrix} \qquad \text{ou} \qquad \begin{bmatrix} & & \\ m\alpha & & m\alpha \\ & & \\ n\alpha & & n\alpha \\ & & \end{bmatrix}$$

où $m, n \in \mathbb{Z}$ et $\alpha \in \mathbb{K}$, dont tous les éléments, sauf les quatre marqués explicitement, sont nuls, peut être multipliée par un vecteur en une seule multiplication dans $\mathbb{K}$. On appelle une telle matrice *élémentaire*.

L'idée est alors de décomposer la matrice M en somme de 7 matrices élémentaires. On commence par soustraire à M les deux matrices élémentaires

$$E_1 = \begin{bmatrix} a & a & & \\ a & a & & \\ & & & \\ & & & \end{bmatrix} \qquad \text{et} \qquad E_2 = \begin{bmatrix} & & & \\ & & & \\ & & d & d \\ & & d & d \end{bmatrix},$$

correspondant aux éléments diagonaux de A. La matrice ainsi obtenue est

$$M_1 = \begin{bmatrix} & & b-a & \\ c-a & d-a & & \\ & & a-d & b-d \\ & & c-d & \end{bmatrix}.$$

3. Strassen lui-même ne se souvient pas exactement de l'origine de ses formules. Selon Landsberg, "*Strassen was attempting to prove, by process of elimination, that such an algorithm did not exist when he arrived at it.*" En tout cas, Strassen affirme "*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 ».

Par construction, elle admet la propriété que l'élément $(1,1)$ du deuxième bloc est l'opposé de l'élément $(2,2)$ du premier. En soustrayant à M_1 la matrice élémentaire

$$E_3 = \begin{bmatrix} & & \\ & d-a & a-d \\ & d-a & a-d \\ & & \end{bmatrix}$$

on obtient la matrice

$$M_2 = \begin{bmatrix} & b-a & & \\ c-a & & d-a & \\ & a-d & & b-d \\ & & c-d & \end{bmatrix},$$

qui se décompose comme $M_3 + M_4$, avec

$$M_3 = \begin{bmatrix} & b-a & \\ & & \\ & a-d & b-d \\ & & \end{bmatrix} \quad \text{et} \quad M_4 = \begin{bmatrix} & & \\ c-a & d-a & \\ & & c-d \\ & & \end{bmatrix}.$$

Or, puisque $a-d = (b-d)-(b-a)$ et $d-a = (c-a)-(c-d)$, chacune des matrices M_3 et M_4 s'écrit comme somme de 2 matrices élémentaires

$$M_3 = E_4 + E_5, \quad \text{où} \quad E_4 = \begin{bmatrix} & b-a & \\ & & \\ & a-b & \\ & & \end{bmatrix}, \quad E_5 = \begin{bmatrix} & & \\ & b-d & b-d \\ & & \end{bmatrix},$$

et

$$M_4 = E_6 + E_7, \quad \text{où} \quad E_6 = \begin{bmatrix} & & \\ c-a & c-a & \\ & & \end{bmatrix}, \quad E_7 = \begin{bmatrix} & d-c & \\ & & \\ & c-d & \end{bmatrix}.$$

En revenant à l'égalité ${}^t\begin{bmatrix} r_{1,1} & r_{2,1} & r_{1,2} & r_{2,2} \end{bmatrix} = \left(\sum_{i=1}^{7} E_i\right) \times {}^t v$ traduisant $R = AX$, on obtient finalement les formules données dans les étapes 3 et 4 en Figure 1.

Exercice 5. Généraliser l'algorithme de Strassen aux matrices de taille arbitraire. Plus précisément, étendre la preuve des formules de Strassen afin de montrer que le produit de deux matrices carrées de taille n peut s'effectuer en $n^3 - n(n-1)/2$ multiplications scalaires non-commutatives.

Est-ce que cela permet d'améliorer la borne de Strassen $\log_2(7)$ sur l'exposant de l'algèbre linéaire ?

2.6. **Peut-on faire mieux que** 2,81 **?** Strassen ayant prouvé que l'exposant ω_{mul} du produit de matrices était sous-cubique, la question naturelle est devenue de savoir quelle est sa véritable valeur.

Pour ce faire, tout au long des années 1970–1990, de nombreux outils ont été développés, les uns plus sophistiqués que les autres, et l'on fit preuve de beaucoup d'ingéniosité. Des notions nouvelles sont apparues, comme celles de *complexité bilinéaire*, de *rang tensoriel* et d'*algorithme approché*, utilisées de manière intensive notamment par Bini, Pan, Schönhage, Strassen, Winograd et d'autres. Ces notions ne seront pas abordées dans ce cours.

Le meilleur algorithme actuel, dû à Coppersmith et Winograd, est de complexité $O(n^{2,376})$. Il est cependant conjecturé que $\omega_{\text{mul}} = 2$, mais pour l'instant la preuve de cette conjecture semble hors de portée.

2.7. **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 d'exposant 2,77 dû à Pan et repris par Kaporin 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 défini modulo un entier de taille « raisonnable », les algorithmes de Winograd et de Waksman sont immédiatement rentables. L'algorithme

de Strassen devient ensuite meilleur pour des tailles de l'ordre de quelques dizaines (64 est une borne raisonnable).

L'immense majorité 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 estimations de complexité. En conséquence, dans leurs versions actuelles, il ne peuvent pas être rentables pour des tailles inférieures à des millions ou des milliards ...

3. Autres problèmes d'algèbre linéaire

Il est important de savoir multiplier les matrices, mais il est tout aussi utile de les inverser, calculer leur polynôme caractéristique, etc ...

3.1. **Élimination de Gauss.** Rappelons que l'algorithme classique d'élimination de Gauss permet de résoudre la plupart des problèmes classiques en algèbre linéaire de manière bien plus efficace que l'algorithmique naïve.

Définition 3 (forme échelonnée)**.** Une matrice est *échelonnée en ligne* lorsque

1. les lignes nulles sont toutes en dessous des lignes non nulles,
2. le premier coefficient non nul (appelé *pivot*) de chaque ligne non nulle est strictement à droite du premier coefficient non nul de la ligne précédente.

Une forme *échelonnée en ligne* d'une matrice A est une matrice échelonnée en ligne B équivalente à gauche à A (*i.e.* telle qu'il existe une matrice carrée inversible P avec $A = PB$).

L'algorithme de Gauss (sans pivot sur les colonnes) calcule, en n'utilisant que des opérations élémentaires sur les lignes (qui correspondent à des multiplications à gauche par des matrices inversibles), une forme échelonnée en ligne de A, sur laquelle le rang et le déterminant de A se lisent directement.

En effet, les lignes non nulles d'une forme échelonnée B forment une base de l'espace vectoriel engendré par les lignes de la matrice de départ A. Le nombre de lignes non nulles de B est égal au rang de A. Si A est carrée, son déterminant est égal au produit des éléments diagonaux de B.

Une variante de l'algorithme de Gauss, dite de *Gauss-Jordan*, produit même une *forme échelonnée réduite* de A : il s'agit d'une forme échelonnée de A, dont les pivots valent 1, les autres coefficients dans les colonnes des pivots étant nuls. Appliquée aux matrices augmentées $[A \,|\, b]$, resp. $[A \,|\, I_n]$, cette variante est utilisée pour la résolution du système $Ax = b$, resp. dans le calcul de l'inverse A^{-1}.

Une autre variante de l'algorithme d'élimination de Gauss permet de calculer une *décomposition* LUP qui, à son tour, permet de résoudre des systèmes linéaires, d'inverser des matrices, de calculer des déterminants, etc.

Définition 4 (matrices L, U, P et décomposition LUP)**.** Une matrice est dite *triangulaire supérieure*, resp. *triangulaire inférieure*, si les éléments situés en dessous (resp. au dessus) de la diagonale principale sont nuls. Une *matrice de permutation* est une matrice carrée P dont les coefficients valent 0 ou 1, et dont chaque ligne et colonne ne contient qu'un seul élément non nul. Une décomposition LU, resp. LUP, d'une matrice A est une factorisation de la forme $A = LU$, resp. $A = LUP$, avec L triangulaire inférieure, U triangulaire supérieure et P une matrice de permutations.

En résumé, nous allons admettre l'énoncé suivant.

Théorème 3 (élimination Gaussienne)**.** *Pour toute matrice $A \in \mathcal{M}_n(\mathbb{K})$, il est possible de calculer en $O(n^3)$ opérations dans $\mathbb{K}$*

- *le rang, le déterminant de A, et l'inverse A^{-1} si A est inversible,*
- *une base de solutions de $Ax = b$, pour tout b dans $\mathbb{K}^n$,*
- *une décomposition LUP, et une forme échelonnée réduite de A.*

3.2. **Résultat principal.** Les résultats importants de cette section sont résumés dans l'énoncé suivant.

Théorème 4 (l'élimination Gaussienne n'est pas optimale)**.** *Soit $\mathbb{K}$ un corps et soit θ un exposant faisable pour la multiplication des matrices à coefficients dans $\mathbb{K}$. Quelle que soit la matrice A de $\mathcal{M}_n(\mathbb{K})$, il est possible de calculer :*

(a) *le déterminant* $\det(A)$,

(b) *l'inverse* A^{-1} *(si elle existe),*

(c) *la solution* $x \in \mathbb{K}^n$ *du système* $Ax = b$, *pour tout* $b \in \mathbb{K}^n$, *si* A *est inversible,*

(d) *les polynômes caractéristique de* A,

(e) *une décomposition* LUP *de* A,

(f) *le rang* $\mathrm{rg}(A)$ *et une forme échelonnée réduite de* A,

(g) *une base du noyau de* A,

en $O(n^\theta)$ *operations dans* $\mathbb{K}$.

La preuve des parties (e), (f) et (g) dépasse le cadre de ce cours. En termes d'exposants, les assertions (a)–(d) impliquent les inégalités $\omega_{\det} \leq \omega_{\mathrm{polcar}} \leq \omega_{\mathrm{mul}}$ et $\omega_{\mathrm{inv}} \leq \omega_{\mathrm{mul}}$.

Dans la suite, nous allons prouver (a)–(d), sous des hypothèses simplificatrices. Nous montrerons également que $\omega_{\mathrm{mul}} \leq \omega_{\mathrm{inv}}$ et laisserons en exercice (Exercice 11) la preuve de $\omega_{\mathrm{inv}} \leq \omega_{\det}$. Utilisées conjointement, ces inégalités permettent de prouver le Théorème 1.

3.3. **La multiplication n'est pas plus difficile que l'inversion.** Soient A et B des matrices $n \times n$. On souhaite calculer $C = AB$. Pour cela, on pose

$$D = \begin{bmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{bmatrix}.$$

On a alors l'identité remarquable

$$D^{-1} = \begin{bmatrix} I_n & -A & AB \\ 0 & I_n & -B \\ 0 & 0 & I_n \end{bmatrix},$$

qui permet donc de ramener le produit en taille n à l'inversion en taille $3n$. Cela prouve notre affirmation et l'inégalité $\omega_{\mathrm{mul}} \leq \omega_{\mathrm{inv}}$.

Exercice 6. Soit $\mathbb{K}$ un corps et soit $\mathsf{T}(n)$ la complexité du produit de deux matrices triangulaires inférieures de taille $n \times n$ et à coefficients dans $\mathbb{K}$. Montrer qu'il est possible de multiplier deux matrices arbitraires de $\mathcal{M}_n(\mathbb{K})$ en $O(\mathsf{T}(n))$ opérations dans $\mathbb{K}$.

3.4. **L'inversion, le calcul de déterminant et la résolution de système ne sont pas plus difficiles que la multiplication.** Nous allons montrer qu'il est possible d'inverser des matrices $n \times n$ par un algorithme de type « diviser pour régner » en $O(n^\theta)$ opérations arithmétiques, quel que soit l'exposant faisable θ.

L'algorithme d'inversion matricielle présenté ici, dû à Strassen, est un algorithme récursif, qui peut s'interpréter comme un « pivot de Gauss par blocs ». Cet algorithme nécessite d'inverser certaines sous-matrices de A ; afin de garantir sa correction, nous allons faire l'hypothèse simplificatrice que *toutes ces matrices sont inversibles.* Le traitement du cas général (A matrice arbitraire) est plus délicat, et passe par le calcul efficace, toujours de type « diviser pour régner », d'une décomposition LUP de A.

Le point de départ est l'identité suivante (non-commutative !), dans laquelle on suppose $n = 2$ et $a \in \mathbb{K} \setminus \{0\}$:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ ca^{-1} & 1 \end{bmatrix} \times \begin{bmatrix} a & 0 \\ 0 & Z \end{bmatrix} \times \begin{bmatrix} 1 & a^{-1}b \\ 0 & 1 \end{bmatrix},$$

et où $Z = d - ca^{-1}b$ est le *complément de Schur* de a dans A.

Inversion matricielle rapide

Entrée : Une matrice « générique » $A \in \mathcal{M}_n(\mathbb{K})$, avec $n = 2^k$.

Sortie : Son inverse A^{-1}.

1. Si $n = 1$, renvoyer A^{-1}.
2. Décomposer $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, avec a, b, c, d dans $\mathcal{M}_{n/2}(\mathbb{K})$.
3. Calculer $e := a^{-1}$ récursivement.
4. Calculer $Z := d - ceb$.
5. Calculer $t := Z^{-1}$ récursivement.
6. Calculer $y := -ebt$, $z := -tce$, $x := e + ebtce$.
7. Renvoyer $\begin{bmatrix} x & y \\ z & t \end{bmatrix}$.

FIGURE 2: Algorithme de Strassen pour inverser une matrice.

Cette identité se déduit aisément grâce à la méthode du pivot de Gauss appliquée à A et permet d'obtenir la factorisation suivante

$$\begin{aligned} \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} &= \begin{bmatrix} 1 & -a^{-1}b \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} a^{-1} & 0 \\ 0 & Z^{-1} \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ -ca^{-1} & 1 \end{bmatrix} \\ &= \begin{bmatrix} a^{-1} + a^{-1}bZ^{-1}ca^{-1} & -a^{-1}bZ^{-1} \\ -Z^{-1}ca^{-1} & Z^{-1} \end{bmatrix}. \end{aligned} \tag{2}$$

Le point important est que l'identité (2) est « non-commutative » : elle reste valable dans tout anneau non-commutatif, pourvu que les éléments a et $Z = d - ca^{-1}b$ soient inversibles. Elle permet donc une application récursive.

L'algorithme qui en résulte est présenté en Figure 2. Dans chaque appel récursif l'algorithme effectue deux inversions, ainsi qu'un certain nombre de produits. Plus précisemment, avec les notations de la Figure 2, les 6 produits ce, bt, $ceb = (ce)b$, $ebt = e(bt)$, $(ebt)(ce)$ et $t(ce)$ sont suffisants. Cette analyse mène au résultat suivant.

Théorème 5. *Si toutes les sous-matrices rencontrées au cours de l'algorithme en Figure 2 sont inversibles, le coût de l'inversion de A est de $3\,\mathsf{MM}(n) + O(n^2)$.*

Démonstration. L'algorithme ci-dessus montre que la complexité $\mathsf{I}(n)$ de l'inversion satisfait à la récurrence

$$\mathsf{I}(n) \leq 2\,\mathsf{I}\left(\frac{n}{2}\right) + 6\,\mathsf{MM}\left(\frac{n}{2}\right) + Cn^2,$$

C étant une constante. Il suffit alors d'appliquer le lemme « diviser pour régner » avec les choix $m = p = s = 2$, $\kappa = 1$ et $q = 4$. □

Une première conséquence immédiate du Théorème 5 est que la résolution du système linéaire $Av = b$, pour $b \in \mathbb{K}^n$, et A inversible vérifiant les hypothèses du Théorème 5, peut s'effectuer également en $O(\mathsf{MM}(n)) = O(n^\theta)$ opérations dans $\mathbb{K}$.

Une légère adaptation de l'algorithme permet de calculer le déterminant de A en même temps que son inverse, en la même complexité. Pour cela, il suffit de remplacer les étapes 1, 3, 5 et 7 de l'algorithme en Figure 2 par les étapes 1′, 3′, 5′ et 7′ décrites ci-dessous.

1′. Si $n = 1$, renvoyer A^{-1} et A.

3′. Calculer $e := a^{-1}$ et $d_a := \det(a)$ récursivement.

5′. Calculer $t := Z^{-1}$ et $d_Z := \det(Z)$ récursivement.

7′. Renvoyer $\begin{bmatrix} x & y \\ z & t \end{bmatrix}$ et $d_a d_Z$.

À ce stade, nous avons démontré les parties (a)–(c) du Théorème 4 sous une hypothèse supplémentaire. Dans le cas où $\mathbb{K}$ est un sous-corps du corps des réels $\mathbb{R}$, on peut s'affranchir de cette hypothèse, en observant que, quelle que soit la matrice A inversible dans $\mathcal{M}_n(\mathbb{R})$, la matrice $B = {}^tA \cdot A$ vérifie les hypothèses du Théorème 5, et donc l'inverse de A peut se calculer grâce à l'égalité $A^{-1} = B^{-1} \cdot {}^tA$ en $O(\mathsf{MM}(n)) = O(n^\theta)$ opérations dans $\mathbb{K}$.

Il n'est pas difficile de prouver que si la matrice A est de type L (triangulaire inférieure) ou U (triangulaire supérieure) alors l'algorithme de Strassen calcule son inverse sans avoir besoin d'*aucune* hypothèse supplémentaire. Cette remarque est à la base d'un algorithme général d'inversion dû à Bunch et Hopcroft. Cet algorithme calcule d'abord une décomposition LUP d'une matrice inversible arbitraire [4] *sur un corps quelconque* $\mathbb{K}$ en $O(\mathsf{MM}(n))$ opérations, et arrive à en déduire le calcul d'inverse (et aussi la résolution de système) pour le même prix.

Concluons cette section avec la remarque que l'algorithme d'inversion de Strassen est à la base de toute une algorithmique pour les matrices structurées, qui sera présentée au Chapitre 11.

3.5. **Calcul du polynôme caractéristique.** Soit A une matrice de $\mathcal{M}_n(\mathbb{K})$. Dans cette section, nous présentons un algorithme efficace, dû à Keller-Gehrig, pour le calcul du polynôme caractéristique $\chi_A(X)$ de A. Nous ne décrirons en détail qu'un cas particulier très simple (et très fréquent). Plus exactement, nous ferons dans la suite l'hypothèse que $\chi_A(X) = X^n + p_{n-1}X^{n-1} + \cdots + p_0$ est irréductible dans $\mathbb{K}[X]$; en particulier, il coïncide avec le polynôme minimal de A.

L'algorithme repose de façon cruciale sur la propriété suivante.

Lemme 1. *Soit v un vecteur arbitraire de $\mathbb{K}^n \setminus \{0\}$ et soit $P \in \mathcal{M}_n(\mathbb{K})$ la matrice dont la j-ième colonne vaut $A^{j-1}v$, pour $1 \le j \le n$. Alors P est inversible et la matrice $P^{-1}AP$ est de type compagnon.*

Pour prouver ce lemme, on commence par observer que, grâce à l'hypothèse sur A, la famille $\{v_1, v_2, \ldots, v_n\}$, où $v_i = A^{i-1}v$, forme une base du $\mathbb{K}$-espace vectoriel $\mathbb{K}^n$. Dans cette base, la matrice de l'application linéaire $w \mapsto Aw$ est égale à la matrice compagnon $C = (c_{i,j})_{i,j=1}^n$ dont les éléments sont tous nuls, sauf les $c_{i+1,i}$ qui valent tous 1 pour $1 \le i \le n-1$, ainsi que $c_{i,n} = -p_{i-1}$, pour $1 \le i \le n$. Le théorème du changement de base fournit alors l'égalité $C = P^{-1}AP$, qui implique aussi que les polynômes caractéristiques de A et de C coïncident.

L'idée de l'algorithme de Keller-Gehrig est de déterminer la matrice P, pour ensuite construire la matrice C à l'aide d'une multiplication et d'une inversion, et de lire enfin les coefficients du polynôme caractéristique de A sur les éléments de la dernière colonne de C.

Du point de vue de la complexité, la seule étape coûteuse est la construction de la matrice P. En effet, la méthode directe qui consiste à calculer la *suite de Krylov* $v, Av, \ldots, A^{n-1}v$ par multiplications successives d'un vecteur par la matrice A a un coût cubique en n. La remarque cruciale est que l'on peut regrouper les produits matrice–vecteur en plusieurs produits matrice–matrice, de la façon suivante : on calcule en $O(n^2)$ opérations les vecteurs v, Av, on détermine ensuite par *exponentiation binaire* les $O(\log(n))$ matrices $A, A^2, A^4, A^8, \ldots$ et on termine le calcul des colonnes de P par le calcul des produits $A^2 \times [\, v \mid Av \,] = [\, A^2v \mid A^3v \,]$, puis $A^4 \times [\, v \mid \cdots \mid A^3v \,] = [\, A^4v \mid \cdots \mid A^7v \,]$, etc. Chaque produit de ce type est effectué à l'aide d'un produit matriciel en taille $n \times n$, en rajoutant artificiellement des colonnes nulles aux facteurs droits.

L'algorithme complet est présenté en Figure 3 (où la notation $\mathsf{Cat}(U, V)$ désigne l'opération de concaténation horizontale des matrices U et V). Nous venons de prouver :

Théorème 6. *Soit $\mathbb{K}$ un corps et soit $\theta \ge 2$ un exposant faisable pour la multiplication de matrices sur $\mathbb{K}$. Plus exactement, supposons qu'on dispose d'un algorithme de multiplication dans $\mathcal{M}_n(\mathbb{K})$ de complexité $\mathsf{MM}(n) = O(n^\theta)$. Si le polynôme caractéristique d'une matrice A de $\mathcal{M}_n(\mathbb{K})$ est irréductible dans $\mathbb{K}[X]$, l'algorithme en* Figure 3 *le calcule en $O(\mathsf{MM}(n)\log(n))$ opérations dans $\mathbb{K}$.*

Le calcul de la suite de Krylov sera utilisé au Chapitre 10 dans l'algorithme de Wiedemann pour la résolution des systèmes creux, et de façon moins transparente, au Chapitre 14 dans l'algorithme de Storjohann pour la résolution de systèmes linéaires à coefficients polynomiaux.

4. La détection de l'inversibilité est également faisable en $O(\mathsf{MM}(n))$ opérations, grâce à un algorithme qui calcule le rang de A en cette complexité ; cet algorithme dépasse notre cadre.

Algorithme de Keller-Gehrig

Entrée : Une matrice A de $\mathcal{M}_n(\mathbb{K})$, avec $n = 2^k$.

Sortie : Son polynôme caractéristique $\det(XI_n - A)$.

1. Choisir v dans $\mathbb{K}^n \setminus \{0\}$;
2. $M := A$; $P := v$;
3. Pour i de 1 à k faire
   $P := \mathsf{Cat}(P, MP)$;
   $M := M^2$;
4. $C := P^{-1}AP$;
5. Renvoyer $\left(X^n + p_{n-1}X^{n-1} + \cdots + p_0\right)$, où ${}^t[-p_0, \ldots, -p_{n-1}]$ est la dernière colonne de C.

FIGURE 3: Calcul du polynôme caractéristique par l'algorithme de Keller-Gehrig.

EXERCICES

Exercice 7 (Multiplication et division rapides de nombres complexes)**.**

1. Montrer qu'il est possible d'effectuer le produit matrice-vecteur
$$\begin{bmatrix} a & -b \\ b & a \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$
en utilisant seulement 3 multiplications scalaires.
2. En déduire que deux nombres complexes peuvent être multipliés en utilisant 3 multiplications réelles.
3. Montrer qu'on peut calculer le quotient de deux nombres complexes en utilisant au plus 7 multiplications et divisions réelles.
4. Donner un algorithme qui utilise seulement 6 opérations de ce type.

Exercice 8 (Multiplication rapide de quaternions)**.**

1. Soit $A = (a_{ij})$ une matrice $n \times n$ à coefficients dans $\mathbb{K}$, telle que $a_{ij} = \pm a_{ji}$ pour $i < j$. Soit v un vecteur arbitraire de $\mathbb{K}^n$. Si ℓ est le nombre d'éléments non-nuls de l'ensemble $\{a_{ij},\, i < j\}$, montrer que $n + \ell$ multiplications suffisent pour calculer le produit Av.
2. Montrer qu'on peut calculer le produit d'une matrice $n \times n$ et d'une matrice $n \times 2$ en utilisant $\frac{3}{2}n^2 + \frac{5}{2}n$ multiplications scalaires.

   Indication : Remplacer le produit $M \times [\ v_1 \mid v_2\ ]$ par $\begin{bmatrix} 0 & M \\ M & 0 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, puis décomposer la matrice $A = \begin{bmatrix} 0 & M \\ M & 0 \end{bmatrix}$ en somme de deux matrices vérifiant les hypothèses de la question précédente.
3. Un *quaternion* est une combinaison formelle $a \cdot 1 + b \cdot i + c \cdot j + d \cdot k$, avec $a, b, c, d \in \mathbb{R}$ et i, j, k éléments vérifiant la table de multiplication interne

$\cdot$	1	i	j	k
1	1	i	j	k
i	i	-1	k	$-j$
j	j	$-k$	-1	i
k	k	j	$-i$	-1

(par exemple, $i \cdot j = k$ et $k \cdot j = -i$). La multiplication $\cdot$ se prolonge par $\mathbb{R}$-linéarité à l'ensemble $\mathbb{H}$ des quaternions et en fait un corps non-commutatif.

Montrer que deux quaternions arbitraires peuvent être multipliés en utilisant seulement 10 multiplications réelles.

Exercice 9. Soit P un polynôme de degré au plus n à coefficients dans un corps $\mathbb{K}$. Soit $\theta > 2$ un réel tel que $O(n^\theta)$ opérations dans $\mathbb{K}$ suffisent pour multiplier deux matrices arbitraires de $\mathcal{M}_n(\mathbb{K})$.

1. Donner un algorithme pour évaluer P sur $\lceil \sqrt{n} \rceil$ éléments de $\mathbb{K}$ en $O(n^{\theta/2})$ opérations.
2. Si $A \in \mathcal{M}_n(\mathbb{K})$, montrer qu'on peut calculer la matrice $P(A)$ en $O(n^{\theta+1/2})$ opérations de $\mathbb{K}$.
3. Si $Q \in \mathbb{K}[x]$ est un autre polynôme de degré au plus n, montrer qu'il est possible de calculer les n premiers coefficients du polynôme $P(Q(x))$ en $O(n^{\frac{\theta+1}{2}})$ opérations dans $\mathbb{K}$.

Indication : Écrire P sous la forme $P_0(X) + P_1(X)X^d + P_2(X)(X^d)^2 + \cdots$, avec d bien choisi et $P_i(X)$ de degrés au plus $d-1$.

Exercice 10 (Inversion de Strassen des matrices polynomiales)**.** Soit $M(X) \in \mathcal{M}_n(\mathbb{K}[X])$ une matrice polynomiale carrée de taille n et de degré borné par d. Exhiber une borne supérieure pour la complexité (exprimée en termes d'opérations arithmétiques dans $\mathbb{K}$ et en fonction des deux paramètres n et d) de l'algorithme de Strassen pour le calcul de $M(X)^{-1}$, sous l'hypothèse que toutes les matrices rencontrées au cours de l'algorithme sont inversibles.

Exercice 11 (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, formalisons brièvement ce qu'on entend par *programme*. Dans notre contexte, un programme prend en entrée des indéterminées $Y_1, \ldots, Y_m$ sur 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

$$g_i : \ G_i = \mathsf{Arg}_i \, \mathrm{op}_i \, \mathsf{Arg}'_i,$$

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
$$G = H(Y_1, \ldots, Y_m, Y_i \text{ op } Y_j),$$
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
$$M \cdot {}^tN = {}^tN \cdot M = DI_n,$$
I_n étant la matrice identité de taille n, et ${}^tN$ 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.

6. En déduire que $\omega_{\text{inv}} \leq \omega_{\text{det}}$.

Notes

L'optimalité de l'algorithme naïf pour le produit matrice-vecteur mentionnée au début de la Section 2 est prouvée dans [58]. Les parties (b) et (c) du Théorème 2 sont dues à [56] et [55]. Le Théorème 4, dont le titre est emprunté à l'article de Strassen [50], montre que du point de vue de la complexité on peut faire mieux que la méthode du pivot de Gauss. C'est un résultat surprenant car Klyuyev et Kokovkin-Shcherbak [28] avaient prouvé que l'élimination Gaussienne est optimale si l'on se restreint à des opérations sur des lignes et des colonnes. C'est surtout un résultat qui a ouvert la voie à tout un domaine de recherche active, la théorie de la complexité bilinéaire. Les livres [41], [11, §15–16], [1, §7–10] et les articles de synthèse [40, 52] constituent de bonnes références sur ce sujet.

L'interprétation des formules de Strassen donnée en Section 2.5 et l'exercice 5 sont tirés de [16]. L'algorithme d'inversion des matrices génériques en Section 3.4 est dû à [50]. La remarque de Schönhage permettant de traiter l'inversion rapide d'une matrice réelle arbitraire est tirée de [46]. L'algorithme général d'inversion évoqué à la fin de la Section 3.4, ainsi que la preuve des égalités $\omega_{\text{inv}} = \omega_{\text{LUP}} = \omega_{\text{mul}}$ ont été donnés dans l'article [9]. La preuve de l'inégalité $\omega_{\text{inv}} \leq \omega_{\text{det}}$ esquissée dans l'exercice 11 est empruntée à [2]. La réduction présentée en Section 3.3 provient de [35]. La partie (f) du Théorème 4 est due à [27], et la partie (g) est due à [10]. Le fait que l'exposant ω_{mul} ne dépend que de la caractéristique du corps de base est prouvé dans [47].

L'algorithme décrit dans la Section 3.5 provient de l'article [27] ; il s'agit d'une version accélérée d'un algorithme classique de Krylov-Danilevsky [29, 15]. Il est possible d'adapter cet algorithme au cas où A est une matrice arbitraire. Plus précisément, l'algorithme de Keller-Gehrig calcule dans ce cas une base de $\mathbb{K}^n$ dans laquelle la matrice de l'endomorphisme de multiplication par A est triangulaire par blocs, avec des blocs diagonaux de type compagnon. Autrement dit, il procède à une mise en *forme de Frobenius faible*. Un rafinement de cet algorithme, dû à Giesbrecht [19], permet de calculer la *forme de Frobenius forte* (matrice diagonale par blocs de type compagnon) également en complexité $\tilde{O}(\mathsf{MM}(n))$; cet algorithme permet aussi le calcul, en la même complexité, du polynôme minimal et de tous les autres facteurs invariants de A. Les algorithmes de [19] sont probabilistes ; des versions déterministes ont été données dans [49]. L'article [27] propose également un algorithme différent de celui décrit dans la Section 3.5, calculant le polynôme caractéristique

d'une matrice *générique* en seulement $O(\mathsf{MM}(n))$ opérations ; voir l'exercice 13 en page 256. Une version (probabiliste) de même complexité et sans hypothèse de généricité a été proposée dans [43].

Les articles [19, 49] montrent que $\omega_{\text{Frobenius}} = \omega_{\text{mul}}$ et que l'évaluation d'un polynôme de degré d en une matrice de $\mathcal{M}_n(\mathbb{K})$ peut se faire en $\tilde{O}(\mathsf{MM}(n) + d)$ opérations dans $\mathbb{K}$, ce qui améliore le résultat (2) de l'exercice 9. Cet exercice est inspiré des articles [7] pour (1), [42] pour (2) et [8] pour (3) ; le résultat de (1) sera amélioré au Chapitre 6 et celui de (3) au Chapitre 4.

À l'heure actuelle, on ne sait pas si les exposants des problèmes suivants
- résoudre un système,
- calculer le rang,
- décider si une matrice est inversible,

sont eux aussi égaux à ω_{mul}. Grâce au Théorème 4, on sait que ces problèmes ne sont pas plus difficiles que la multiplication, mais peut-être sont-ils plus faciles ...

Un autre problème ouvert est le calcul du polynôme caractéristique par un algorithme déterministe de complexité $O(\mathsf{MM}(n))$.

Complexité des algorithmes commutatifs. Il est naturel de se demander s'il est possible d'obtenir des algorithmes *à la Winograd* plus efficaces, en regroupant les termes de (1) en paquets plus gros plutôt que deux par deux. La réponse est négative. Harter [20] a montré qu'il n'existe pas de généralisation de (1) de la forme

$$\langle \ell, c\rangle = \lambda(a_1, x_1, a_2, x_2, \ldots) + \mu(a_1, x_1, a_2, x_2, \ldots) + f(a_1, a_2, \ldots) + g(x_1, x_2, \ldots),$$

où λ et μ sont des formes linéaires et f et g des fonctions quelconques. Ja'Ja [24] a montré plus généralement qu'aucun algorithme commutatif ne peut apporter un gain d'un facteur supérieur à 2 sur l'algorithme naïf, et en particulier ne peut pas améliorer l'exposant 3. De ce point de vue, les algorithmes de Winograd et Waksman sont optimaux dans la classe des algorithmes commutatifs.

Algorithme de Strassen. Winograd [59, 17] a donné une version de l'algorithme de Strassen utilisant seulement $C = 15$ additions et soustractions (au lieu de $C = 18$). Cela permet de descendre la constante dans le $O(n^{\log_2(7)})$ de 7 à 6 pour l'algorithme en Figure 1. Cette constante a été abaissée à 3,92 dans [18, 17].

Dans le cas où la taille n de la matrice n'est pas une puissance de 2 ($2^{k-1} < n < 2^k$) il vaut mieux éviter d'appliquer l'algorithme de Strassen sur la matrice augmentée de $2^k - n$ lignes et colonnes nulles. Si n est pair, on effectuerait plutôt un découpage en 4 blocs de taille $n/2$. Si n est impair, on utiliserait la formule de Strassen sur des blocs de taille $(n-1)/2$ en gérant « à la main » les opérations sur la ligne et la colonne restantes. D'autres stratégies sont décrites dans [23].

Bornes inférieures vs. bornes supérieures. Le problème des bornes inférieures est très difficile. En taille $n = 2$, il a été montré [57, 22] que le résultat de Strassen est optimal du point de vue des multiplications : on ne peut pas effectuer le produit de deux matrices carrées de taille 2 en seulement 6 multiplications scalaires. Récemment Landsberg [32] a obtenu une généralisation de ce résultat, en utilisant des outils géométriques. Probert [44] a montré que la version de l'algorithme de Strassen due à Winograd est optimale également vis-à-vis du nombre total d'opérations arithmétiques : il n'existe aucun algorithme pour multiplier deux matrices 2×2 en 7 multiplications et seulement 14 additions et soustractions.

La taille $n = 2$ est la seule pour laquelle on connaît le nombre minimal de multiplications scalaires requises pour effectuer le produit de matrices carées $n \times n$. En taille $n = 3$, la meilleure borne inférieure connue à ce jour est 19 [6], alors que le meilleur algorithme commutatif utilise 22 multiplications [34] et le meilleur algorithme non-commutatif utilise 23 multiplications [31]. Pour une taille arbitraire, toutes les bornes inférieures connues sont seulement quadratiques en n : la meilleure, $\frac{5}{2}n^2 - 3n$, est due à Bläser [5]. Si le corps des scalaires est $\mathbb{R}$ ou $\mathbb{C}$, et dans un modèle restreint de complexité où les seules multiplications scalaires permises se font par des scalaires de module borné, Raz [45] a prouvé une borne inférieure en $\Omega(n^2 \log n)$.

Matrices de petite taille. Il existe un grand nombre d'autres schémas *à la Strassen* pour multiplier des matrices de petite taille. Un recensement des meilleurs algorithmes non-commutatifs connus pour le produit des matrices de petite taille a été effectué par Smith [48].

Par exemple, Sýkora [53] a montré qu'on peut multiplier deux matrices $n \times n$ en $n^3 - (n-1)^2$ produits non-commutatifs ; cet algorithme généralise à la fois ceux de Strassen et de Laderman. Des

techniques plus sophistiquées ont permis à Pan [37, 38] de montrer que le produit de matrices $n \times n$ peut s'effectuer en $n^3/3 + 9/2n^2 - n/3$ multiplications non-commutatives. Chacun de ces schémas, appliqué récursivement, donne un algorithme de produit matriciel sous-cubique. Par exemple, l'algorithme de [38] multiplie des matrices 48×48 en 47216 multiplications non-commutatives, ce qui mène à $\omega_{\text{mul}} < 2{,}781$.

À la recherche de $\omega_{\text{mul}} = 2$. L'idée initiale de Strassen de combiner une bonne multiplication en taille fixée et la puisssance de la récursivité pour améliorer l'exposant ω_{mul} n'a pas permis d'aller en dessous de 2,7. Les améliorations ultérieures reposent toutes sur une accumulation d'idées de plus en plus sophistiquées. Une première idée fut d'utiliser le produit des matrices rectangulaires. Un résultat de [36, 21] montre que si l'on dispose d'un algorithme pour multiplier deux matrices $(m \times n) \times (n \times p)$ en ℓ multiplications, alors on sait effectuer en ℓ multiplications chacun des produits en tailles $(n \times p) \times (p \times m)$ et $(p \times m) \times (m \times n)$. Cet énoncé est lié au théorème de transposition de Tellegen qui sera vu au Chapitre 12. Une décomposition par blocs permet alors d'obtenir un algorithme pour le produit de matrices carrées de taille mnp en ℓ^3 opérations, et d'obtenir ainsi la majoration $\omega_{\text{mul}} \leq 3\log(\ell)/\log(mnp)$. Ce résultat n'a permis d'améliorer la majoration de ω_{mul} que combiné avec une autre idée, des *algorithmes approchés* et *rang tensoriel limite* (*border rank* en anglais) introduite dans [4, 3]. L'article [4] donne un algorithme approché qui utilise 10 multiplications pour le produit matriciel en taille $(2 \times 2) \times (2 \times 3)$; cela implique $\omega_{\text{mul}} < \log_{12}(1000) < 2{,}77989$, mais les constantes dans les $O(\cdot)$ commençent à devenir astronomiques. D'autres idées pour obtenir des bornes de plus en plus fines sur ω_{mul} sont : l'utilisation des matrices creuses [47] qui aboutit à l'inégalité asymptotique pour les sommes directes de *matrices à trous* et à la majoration $\omega_{\text{mul}} < 2{,}548$; la théorie de la déformation des modules sur des groupes algébriques linéaires conduisant à la *méthode laser* de Strassen [51] et à la borne $\omega_{\text{mul}} < 2{,}4785$, et qui est à la base de la meilleure borne connue $\omega_{\text{mul}} < 2{,}376$ due à Coppersmith et Winograd [14].

Théorie vs. pratique. Théoriquement, le principal problème ouvert est $\omega_{\text{mul}} = 2$. D'un point de vue pratique, il n'est pas forcément intéressant de comparer les algorithmes en fonction de leur exposant, mais plutôt de répondre à des questions du type : quelle est la formule la plus simple qui mène à un exposant faisable $\theta < \log_2(7)$? Quel est l'algorithme le plus rapide pour une taille donnée ?

L'idée des algorithmes mentionnés en Section 2.7 est due à Pan [36, 39]. L'article [30] est le premier à avoir mis en avant leur aspect pratique ; les derniers progrès basés sur cette idée sont dûs à Kaporin [26, 25].

La recherche théorique pour améliorer les bornes sur ω_{mul} a demandé beaucoup d'ingéniosité (et de chance !), mais a mené à des algorithmes de plus en plus éloignés du calcul réel. Smith [48] adopte un point de vue pragmatique et propose d'automatiser la recherche d'algorithmes rapides en petite taille. Il ramène la recherche d'un algorithme qui calcule le produit de deux matrices de types $(a \times b)$ et $(b \times c)$ en m multiplications non-commutatives à un problème d'optimisation dans un espace à $(ab + bc + ca)m$ dimensions. Cela lui permet de retrouver tous les algorithmes connus pour les petites tailles et d'en découvrir des nouveaux. Ses programmes retrouvent les formules de Strassen en quelques secondes et celles de Laderman en quelques minutes.

Un point de vue différent est celui développé par Landsberg [33], qui montre que les questions ouvertes concernant la complexité de la multiplication matricielle se formulent convenablement en termes géométriques et théorie de la représentation.

Enfin, une autre piste actuellement explorée pour prouver (ou infirmer) que $\omega_{\text{mul}} = 2$ est celle de Cohn et Umans. Dans [13, 12], ils développent une approche unifiée permettant de ramener la multiplication matricielle à l'étude de la transformée de Fourier discrète dans des algèbres groupales associées à certains groupes finis, possédant de bonnes propriétés vis-à-vis de leurs représentations irréductibles. Ils retrouvent par ce biais $\omega_{\text{mul}} < 2{,}41$ et ramènent $\omega_{\text{mul}} = 2$ à des conjectures combinatoires et en théorie des groupes.

Déterminant et permanent. Le permanent d'une matrice $A \in \mathcal{M}_n(\mathbb{K})$ se définit

$$\text{perm}(A) = \sum_{\sigma \in \mathcal{S}_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Malgré la ressemblance entre cette définition et celle de $\det(A)$, on ne connaît pas d'algorithme de complexité polynomiale en n calculant $\mathrm{perm}(A)$ sur un corps $\mathbb{K}$ de caractéristique différente de 2. Valiant [54] a prouvé que le calcul du permanent pour les matrices $n \times n$ à coefficients dans $\{0, 1\}$ est un problème $\#P$-complet.

Bibliographie

[1] Abdeljaoued (J.) and Lombardi (H.). – *Méthodes matricielles : introduction à la complexité algébrique*. – Springer-Verlag, Berlin, 2004, *Mathématiques & Applications*, vol. 42, xvi+376p.

[2] Baur (W.) and Strassen (V.). – The complexity of partial derivatives. *Theoretical Computer Science*, vol. 22, 1983, pp. 317–330.

[3] Bini (D.). – Relations between exact and approximate bilinear algorithms. Applications. *Calcolo*, vol. 17, n1, 1980, pp. 87–97.

[4] Bini (D.), Capovani (M.), Romani (F.), and Lotti (G.). – $O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication. *Inform. Process. Lett.*, vol. 8, n5, 1979, pp. 234–235.

[5] Bläser (Markus). – A $\frac{5}{2}n^2$-lower bound for the multiplicative complexity of $n \times n$-matrix multiplication. In *STACS 2001 (Dresden)*, pp. 99–109. – Springer, Berlin, 2001.

[6] Bläser (Markus). – On the complexity of the multiplication of matrices of small formats. *J. Complexity*, vol. 19, n1, 2003, pp. 43–60.

[7] Borodin (A.) and Munro (I.). – Evaluation of polynomials at many points. *Information Processing Letters*, vol. 1, n2, 1971, pp. 66–68.

[8] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n4, 1978, pp. 581–595.

[9] Bunch (James R.) and Hopcroft (John E.). – Triangular factorization and inversion by fast matrix multiplication. *Math. Comp.*, vol. 28, 1974, pp. 231–236.

[10] Bürgisser (P.), Karpinski (M.), and Lickteig (T.). – Some computational problems in linear algebra as hard as matrix multiplication. *Comput. Complexity*, vol. 1, n2, 1991, pp. 131–155.

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

[12] Cohn (H.), Kleinberg (R.), Szegedy (B.), and Umans (C.). – Group-theoretic algorithms for matrix multiplication. In *FOCS'05*. pp. 379–388. – IEEE Computer Society, 2005.

[13] Cohn (Henry) and Umans (Christopher). – A group-theoretic approach to fast matrix multiplication. In *FOCS'03*. p. 438. – IEEE Computer Society, Washington, DC, USA, 2003.

[14] Coppersmith (Don) and Winograd (Shmuel). – Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, vol. 9, n3, 1990, pp. 251–280.

[15] Danilevskii (A. M). – The numerical solution of the secular equation. *Matem. sbornik*, vol. 44, n2, 1937, pp. 169–171. – (in Russian).

[16] Fiduccia (Charles M.). – On obtaining upper bounds on the complexity of matrix multiplication. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pp. 31–40. – Plenum, New York, 1972.

[17] Fischer (P. C.) and Probert (R. L.). – Efficient procedures for using matrix algorithms. In *Automata, languages and programming*, pp. 413–427. LNCS, Vol. 14. – Springer, Berlin, 1974.

[18] Fischer (Patrick C.). – Further schemes for combining matrix algorithms. In *Automata, languages and programming*, pp. 428–436. LNCS, Vol. 14. – Springer, Berlin, 1974.

[19] Giesbrecht (Mark). – Nearly optimal algorithms for canonical matrix forms. *SIAM J. Comput.*, vol. 24, n5, 1995, pp. 948–969.

[20] Harter (Richard). – The optimality of Winograd's formula. *Commun. ACM*, vol. 15, n5, 1972, p. 352.

[21] Hopcroft (J.) and Musinski (J.). – Duality applied to the complexity of matrix multiplication and other bilinear forms. *SIAM Journal on Computing*, vol. 2, 1973, pp. 159–173.

[22] Hopcroft (J. E.) and Kerr (L. R.). – On minimizing the number of multiplications necessary for matrix multiplication. *SIAM J. Appl. Math.*, vol. 20, 1971, pp. 30–36.

[23] Huss-Lederman (S.), Jacobson (E. M.), Tsao (A.), Turnbull (T.), and Johnson (J. R.). – Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing'96*. – IEEE Computer Society, Washington, DC, USA, 1996. 32 pp.

[24] Ja'Ja' (Joseph). – On the complexity of bilinear forms with commutativity. In *STOC'79*. pp. 197–208. – ACM, New York, NY, USA, 1979.

[25] Kaporin (I). – The aggregation and cancellation techniques as a practical tool for faster matrix multiplication. *Theoretical Computer Science*, vol. 315, n2-3, 2004, pp. 469–510.

[26] Kaporin (Igor). – A practical algorithm for faster matrix multiplication. *Numer. Linear Algebra Appl.*, vol. 6, n8, 1999, pp. 687–700.

[27] Keller-Gehrig (Walter). – Fast algorithms for the characteristic polynomial. *Theoret. Comput. Sci.*, vol. 36, n2-3, 1985, pp. 309–317.

[28] Klyuyev (V. V.) and Kokovkin-Shcherbak (N. I.). – On the minimization of the number of arithmetic operations for the solution of linear algebraic systems of equations. *USSR Computational Mathematics and Mathematical Physics*, vol. 5, 1965, pp. 25–43.

[29] Krylov (A. N.). – On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined. *Izv. Akad. Nauk SSSR*, vol. 7, n4, 1931, pp. 491–539. – (in Russian).

[30] Laderman (Julian), Pan (Victor), and Sha (Xuan He). – On practical algorithms for accelerated matrix multiplication. *Linear Algebra Appl.*, vol. 162/164, 1992, pp. 557–588.

[31] Laderman (Julian D.). – A noncommutative algorithm for multiplying 3×3 matrices using 23 muliplications. *Bull. Amer. Math. Soc.*, vol. 82, n1, 1976, pp. 126–128.

[32] Landsberg (J. M.). – The border rank of the multiplication of 2×2 matrices is seven. *J. Amer. Math. Soc.*, vol. 19, n2, 2006, pp. 447–459.

[33] Landsberg (J. M.). – Geometry and the complexity of matrix multiplication. *Bull. Amer. Math. Soc. (N.S.)*, vol. 45, n2, 2008, pp. 247–284.

[34] Makarov (O. M.). – An algorithm for multiplication of 3×3 matrices. *Zh. Vychisl. Mat. i Mat. Fiz.*, vol. 26, n2, 1986, pp. 293–294, 320.

[35] Munro (I.). – Problems related to matrix multiplication. In Rustin (R.) (editor), *Proceedings Courant Institute Symposium on Computational Complexity, October 1971*. pp. 137–151. – Algorithmics Press, New York, 1973.

[36] Pan (V.). – Computation schemes for a product of matrices and for the inverse matrix. *Uspehi Mat. Nauk*, vol. 27, n5(167), 1972, pp. 249–250.

[37] Pan (V. Ya.). – Strassen's algorithm is not optimal. Trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations. In *SFCS '78*. pp. 166–176. – IEEE Computer Society, Washington, DC, USA, 1978.

[38] Pan (V. Ya.). – New fast algorithms for matrix operations. *SIAM J. Comput.*, vol. 9, n2, 1980, pp. 321–342.

[39] Pan (V. Ya.). – New combinations of methods for the acceleration of matrix multiplication. *Comput. Math. Appl.*, vol. 7, n1, 1981, pp. 73–125.

[40] Pan (Victor). – How can we speed up matrix multiplication ? *SIAM Rev.*, vol. 26, n3, 1984, pp. 393–415.

[41] Pan (Victor). – *How to multiply matrices faster*. – Springer-Verlag, Berlin, 1984, *Lecture Notes in Computer Science*, vol. 179, xi+212p.

[42] Paterson (M. S.) and Stockmeyer (L. J.). – On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, vol. 2, n1, 1973, pp. 60–66.

[43] Pernet (Clément) and Storjohann (Arne). – Faster algorithms for the characteristic polynomial. In *ISSAC'07*, pp. 307–314. – ACM, New York, 2007.

[44] Probert (Robert L.). – On the additive complexity of matrix multiplication. *SIAM J. Comput.*, vol. 5, n2, 1976, pp. 187–203.

[45] Raz (Ran). – On the complexity of matrix product. *SIAM J. Comput.*, vol. 32, n5, 2003, pp. 1356–1369.

[46] Schönhage (A.). – Unitäre Transformationen grosser Matrizen. *Numer. Math.*, vol. 20, 1972/73, pp. 409–417.

[47] Schönhage (A.). – Partial and total matrix multiplication. *SIAM J. Comput.*, vol. 10, n3, 1981, pp. 434–455.

[48] Smith (Warren D.). – Fast matrix multiplication formulae – report of the prospectors. – 2002. Preprint, available at `http://www.math.temple.edu/~wds/prospector.pdf`.

[49] Storjohann (Arne). – Deterministic computation of the Frobenius form (extended abstract). In *42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001)*, pp. 368–377. – IEEE Computer Soc., Los Alamitos, CA, 2001.

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

[51] Strassen (V.). – Relative bilinear complexity and matrix multiplication. *J. Reine Angew. Math.*, vol. 375/376, 1987, pp. 406–443.

[52] Strassen (Volker). – Algebraic complexity theory. In *Handbook of theoretical computer science, Vol. A*, pp. 633–672. – Elsevier, Amsterdam, 1990.

[53] Sýkora (Ondrej). – A fast non-commutative algorithm for matrix multiplication. In *Mathematical foundations of computer science (Proc. Sixth Sympos., Tatranská Lomnica, 1977)*, pp. 504–512. Lecture Notes in Comput. Sci., Vol. 53. – Springer, Berlin, 1977.

[54] Valiant (L. G.). – The complexity of computing the permanent. *Theoret. Comput. Sci.*, vol. 8, n2, 1979, pp. 189–201.

[55] Waksman (Abraham). – On Winograd's algorithm for inner products. *IEEE Trans. Comput.*, vol. C-19, n4, 1970, pp. 360–361.

[56] Winograd (S.). – A new algorithm for inner-product. *IEEE Transactions on Computers*, vol. 17, 1968, pp. 693–694.

[57] Winograd (S.). – On multiplication of 2×2 matrices. *Linear Algebra and Appl.*, vol. 4, 1971, pp. 381–388.

[58] Winograd (Shmuel). – On the number of multiplications necessary to compute certain functions. *Comm. Pure Appl. Math.*, vol. 23, 1970, pp. 165–179.

[59] Winograd (Shmuel). – Some remarks on fast multiplication of polynomials. In *Complexity of sequential and parallel numerical algorithms (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1973)*, pp. 181–196. – Academic Press, New York, 1973.

CHAPITRE 4

Calculs rapides sur les séries

Résumé

La multiplication rapide de polynômes vue au Chapitre 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. La composition de séries est en général plus coûteuse que le produit, mais peut aussi tirer parti d'une multiplication rapide.

Ce chapitre 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 formelle tronquée à l'ordre N ». Ainsi, « calculer une série » voudra toujours dire en calculer les N premiers termes. Comme dans le Chapitre 2, $\mathsf{M}(N)$ dénote une borne supérieure sur le nombre d'opérations arithmétiques nécessaires pour multiplier deux polynômes de degré inférieur à N. L'efficacité des algorithmes sera mesurée par leur complexité arithmétique.

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. Ainsi, si Φ est une fonction C^1 de $\mathbb{R}$ dans $\mathbb{R}$, pour résoudre $\Phi(y) = 0$ dans $\mathbb{R}$, on choisit $y_0 \in \mathbb{R}$ et on itère

$$y_{k+1} = \mathcal{N}(y_k) = y_k - \frac{\Phi(y_k)}{\Phi'(y_k)}. \tag{1}$$

Cette itération est représentée en Figure 1. La solution serait exacte si Φ était une fonction linéaire ; dans le cas général, si y_0 est bien choisi (par exemple assez proche d'une racine simple), la suite y_k converge vers une limite y_∞. 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.

Dans le cadre formel, la fonction Φ est remplacée par une application des séries formelles dans les séries formelles. Dans bien des cas, l'itération de Newton converge, et le nombre de *coefficients* corrects est doublé à chaque itération, ce qui mène à une bonne complexité. La composition des séries reste une opération plus coûteuse, à laquelle est consacrée la fin du chapitre.

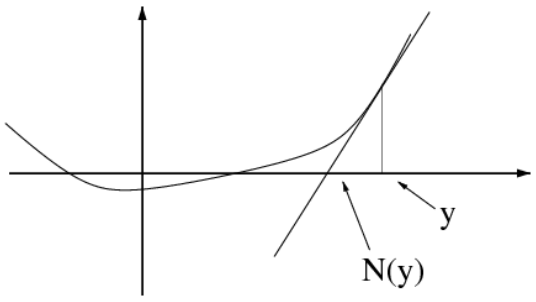


FIGURE 1: Opérateur de Newton de $\mathbb{R}$ dans $\mathbb{R}$.

1. Séries formelles

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

$$F(X) = \sum_{i\geq 0} f_i X^i.$$

Le coefficient f_i est appelé le $i$ième coefficient de $F(X)$, le coefficient f_0 est appelé terme constant de $F(X)$, et parfois noté $F(0)$.

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

$$\sum_{i\geq 0} f_i X^i \times \sum_{i\geq 0} g_i X^i = \sum_{i\geq 0} h_i X^i, \qquad \text{avec} \quad h_i = \sum_{j+k=i} f_j g_k,$$

la dernière somme étant finie.

Exemple 1. La série formelle

$$1 = 1\cdot X^0 + 0\cdot X + 0\cdot X^2 + \cdots,$$

où 1 est l'unité de $\mathbb{A}$, est élément neutre pour la multiplication de $\mathbb{A}[[X]]$. La formule donnant les coefficients du produit se réduit alors à un terme.

Exemple 2. Si $F = 1 + X$ et $G = 1 - X + X^2 - X^3 + \cdots$, alors le produit vaut $H = FG = 1$: le coefficient de X^n dans H vaut $1 - 1 = 0$ pour $n > 0$ et 1 sinon. La série G est donc l'inverse de F pour la multiplication, que l'on peut noter $(1 + X)^{-1}$.

Métrique. 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, quel que soit l'anneau de coefficients $\mathbb{A}$, 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 1. Vérifier que la fonction d définie ci-dessus est bien une distance.

Muni de cette distance, l'anneau des séries formelles forme un espace métrique *complet* (les suites de Cauchy convergent). En effet, dire qu'une suite (S_n) de séries est de Cauchy signifie qu'étant donné $k \in \mathbb{N}$, il existe un entier N tel que pour tous $m, n \geq N$, on ait $d(S_m, S_n) < 2^{-k}$, autrement dit après l'indice N, les k premiers termes des S_n sont fixés, ce sont ceux de la série limite.

Composition. Si F et G sont deux séries formelles, avec terme constant $G(0) = 0$, on définit la composition comme

$$F(G(X)) = f_0 + f_1 G(X) + f_2 G(X)^2 + \cdots$$

Les points de suspension signifient que l'on considère la limite des sommes H_n obtenues en arrêtant la somme au terme $f_n G(X)^n$. Cette limite existe puisque pour $m \geq n$, la distance obéit à $d(H_m, H_n) \leq 2^{-n}$, ce qui fait de $(H_n)_n$ une suite de Cauchy.

Inverse. Si F est de la forme $F = 1 + GX$ avec $G \in \mathbb{A}[[X]]$, alors F est inversible et son inverse est donné par

$$1 - GX + G^2X^2 - \cdots,$$

composition de $(1 + X)^{-1}$ par GX. La preuve découle de cette composition : $(1 + GX)(1 - GX + \ldots) = H(GX)$ où $H = (1 + X)(1 + X)^{-1} = 1$.

Si $a = F(0)$ est inversible, la série F se récrit $F = a(1 + GX)$ avec $G = a^{-1}(F - a)/X$, donc F est inversible, d'inverse $a^{-1}(1 + GX)^{-1}$.

Ces ingrédients mènent à la structure d'anneau de l'ensemble de séries formelles.

Proposition 1. *L'ensemble $\mathbb{A}[[X]]$ des séries formelles à coefficients dans $\mathbb{A}$ est un anneau commutatif unitaire. Ses éléments inversibles sont les séries de terme constant inversible.*

Démonstration. L'addition est commutative comme celle de $\mathbb{A}$. L'associativité et la distributivité du produit sont obtenues comme pour les polynômes. L'unité pour le produit est la série 1. La première partie du lemme est donc prouvée.

Si $F = \sum f_i X^i \in \mathbb{A}[[X]]$ est inversible, et $G = \sum g_i X^i$ est son inverse, alors l'extraction du coefficient de X^0 dans l'identité $FG = 1$ donne $f_0 g_0 = 1$ ce qui prouve qu'une série inversible a un terme constant inversible. □

Séries multivariées. Comme $\mathbb{A}[[X]]$ est un anneau commutatif unitaire, 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 $\mathbb{A}[[X, Y]]$.

Dérivation. La dérivée d'une série est définie formellement coefficient par coefficient via l'identité

$$\left(\sum_{i\geq 0} f_i X^i\right)' = \sum_{i\geq 0} (i+1) f_{i+1} X^i.$$

Les relations habituelles $(F+G)' = F' + G'$ et $(FG)' = F'G + FG'$ sont prouvées comme pour les polynômes. Dans le cas de séries en plusieurs variables, on utilise la notation des dérivées partielles $\partial F/\partial X$ pour désigner la dérivée par rapport à la variable X.

Troncatures. 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 chapitre 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 $N-1$, et même plus généralement $S = O(T^k)$ si $S = O(X^{k\,\mathrm{val}(T)})$.

Formule de Taylor. Le lien entre la composition et la dérivation est donné par la formule de Taylor. Si F, G, H sont trois séries formelles, avec $G(0) = H(0) = 0$, et les entiers $2, 3, \ldots, k-1$ sont inversibles dans $\mathbb{A}$, alors

$$F(G+H) = F(G) + F'(G)H + F''(G)\frac{H^2}{2!} + \cdots + O(H^k).$$

Pour cette identité, la commutativité de $\mathbb{A}$ est cruciale. La formule est classique pour les polynômes, et les coefficients de X^N dans les deux membres de la formule sont les mêmes que ceux de l'identité entre polynômes obtenue en considérant les troncatures de F, G et H modulo X^{N+1}, ce qui permet de conclure.

Intégration. Il s'agit de l'opération inverse. On suppose que les entiers sont inversibles dans $\mathbb{A}$ ($\mathbb{A}$ est une $\mathbb{Q}$-algèbre) et on définit alors

$$\int \sum_{i\geq 0} f_i X^i = \sum_{i\geq 0} f_i \frac{X^{i+1}}{i+1}.$$

Exemple 3. La série $\log(1+X)$ vaut

$$\log(1+X) = \int (1+X)^{-1} = X - \frac{1}{2}X^2 + \frac{1}{3}X^3 + \cdots$$

Inverse multiplicatif de séries formelles

Entrée : un entier $N > 0$, $F \bmod X^N$ une série tronquée ;

Sortie : $F^{-1} \bmod X^N$.

Si $N = 1$, alors renvoyer f_0^{-1}, où $f_0 = F(0)$.
Sinon,

1. Calculer récursivement l'inverse G de $F \bmod X^{\lceil N/2 \rceil}$;
2. Renvoyer $G + (1 - GF)G \bmod X^N$.

Figure 2: Inverse de série par itération de Newton.

2. La méthode de Newton pour le calcul d'inverses

Partant de la fonction $\Phi(y) = 1/y - f$, l'itération (1) devient

$$\mathcal{N}(g_k) = g_k + g_k^2(1/g_k - f) = g_k + g_k(1 - g_k f).$$

Cette itération, ou des variantes, permet le calcul d'inverses dans plusieurs contextes.

2.1. Convergence quadratique pour l'inverse d'une série formelle.

Lemme 1. *Soit $F \in \mathbb{A}[[X]]$ une série formelle de terme constant inversible et G une série telle que $G - 1/F = O(X^n)$ $(n \geq 1)$, alors la série*

$$\mathcal{N}(G) = G + G(1 - GF) \tag{2}$$

vérifie $\mathcal{N}(G) - 1/F = O(X^{2n})$.

Démonstration. Par hypothèse, on peut définir $H \in \mathbb{A}[[X]]$ par $1 - GF = X^n H$. Il suffit alors de récrire $F = G^{-1}(1 - HX^n)$ et d'inverser :

$$1/F = G(1 + HX^n + O(X^{2n})) = G(1 + HX^n) + G \cdot O(X^{2n}) = \mathcal{N}(G) + O(X^{2n}).$$

□

2.2. Algorithme.

Lemme 2. *L'algorithme d'inversion de la Figure 2 est correct.*

Démonstration. La preuve est une récurrence sur les entiers. Pour $N = 1$ la propriété est claire. Si la propriété est vraie pour $1 \leq N < k$, alors elle l'est pour $1 \leq \lceil k/2 \rceil < k$ donc le résultat de l'étape 1 vaut $F \bmod X^{\lceil N/2 \rceil}$. Alors, d'après le lemme 1, $G + (1 - GF)G = 1/F \bmod X^{2\lceil N/2 \rceil}$ et la troncature modulo X^N est donc bien correcte. □

2.3. Complexité.

Chaque itération coûte essentiellement deux multiplications, et l'ensemble est donc effectué en $O(\mathsf{M}(N))$ opérations d'après le lemme « divise pour régner ». Une estimation plus précise est donnée par la proposition suivante.

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

Démonstration. L'étape 2 de l'algorithme demande 2 multiplications modulo X^N, plus $C \cdot N$ opérations supplémentaires (additions, ...), C étant une constante que l'on ne va pas déterminer. Notons $\mathsf{N}(N)$ le coût du calcul modulo N ; on a alors

$$\mathsf{N}(N) \leq \mathsf{N}(\lceil N/2 \rceil) + 2\mathsf{M}(N) + CN.$$

L'application du lemme « diviser pour régner » donne alors

$$\mathsf{N}(N) \leq 4\mathsf{M}(N) + 2CN$$

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

Optimisations. Le coût de la Proposition 2 peut être encore divisé d'un facteur 2 :

Dans le produit $G(1-GF)$, le deuxième terme est de la forme $X^{\lceil N/2\rceil}R_N$, alors que G est de degré inférieur à $\lceil N/2\rceil$. Ainsi, seul le produit GR_N est nécessaire, et celui-ci ne demande que $\mathsf{M}(N/2)$ opérations.

D'autre part, le produit GF est de la forme $1+X^{\lceil N/2\rceil}R_N$. Il s'agit donc de calculer le produit d'un polynôme de degré $N/2$ par un polynôme de degré N, alors que l'on connaît à l'avance les $N/2$ premiers coefficients du produit. Dans cette situation, il est possible d'abaisser le coût du produit de $\mathsf{M}(N)$ à $\mathsf{M}(N/2)$, en utilisant un produit médian et des techniques non triviales de « transposition » de code abordées au Chapitre 12.

2.4. **Division de séries.** Le quotient H/F où H et F sont des séries et F est inversible, peut être obtenu comme le produit $H \times F^{-1}$. Ce calcul peut être amélioré d'un facteur constant. Il suffit pour cela de remplacer la dernière itération $G + (1-GF)G$ par $Y + (H-YF)G$ où $Y := HG \bmod X^{\lceil N/2\rceil}$. Cette astuce remplace un produit de la forme $(N/2, N)$ (pour GF), un produit de la forme $(N/2, N/2)$ (pour $(1-GF)G$) et un produit final (N, N) par deux produits $(N/2, N/2)$ (pour Y et $(H-YF)G$) et un produit $(N/2, N)$ pour calculer YF. L'économie est donc la différence $\mathsf{M}(N) - \mathsf{M}(N/2)$.

2.5. **Application à la division Euclidienne.** L'algorithme 2 est l'élément clé de la division Euclidienne rapide, traitée au Chapitre 5.

2.6. **Application au logarithme.** Si $F \in \mathbb{A}[[X]]$ est telle que $F(0) = 0$, on définit la série $\log(1+F)$ par composition avec $\log(1+X)$. Pour calculer cette série, il suffit d'utiliser l'identité

$$\log(1+F) = \int \frac{F'}{1+F}.$$

Exercice 2. Prouver cette identité à partir des définitions de log, $\int$ et de la dérivation données plus haut.

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

2.7. **Extension : inverse de matrices.** L'anneau $\mathbb{A}$ étant commutatif, il n'est pas possible de traiter directement des matrices de séries comme des séries de matrices. Une variante de l'itération (2) permet d'adapter l'inversion à ce cadre.

Lemme 3. *Soient A et B deux matrices carrées à coefficients dans $\mathbb{A}[[X]]$, telles que $A(0)$ soit inversible et $B - A^{-1} = O(X^n)$ $(n \geq 1)$. Alors la matrice*

$$\mathcal{N}(B) = B + (I - BA)B$$

vérifie $\mathcal{N}(B) - A^{-1} = O(X^{2n})$.

Démonstration. Comme ci-dessus, l'hypothèse $n \geq 1$ entraîne que le terme constant de B vaut $A(0)^{-1}$. Ensuite, la preuve est la même que pour l'inverse de séries : en définissant la matrice de séries $C := (I - BA)/X^n$, on peut récrire $A = B^{-1}(I - X^n C)$, et il vient

$$A^{-1} = (I + X^n C + O(X^{2n}))B = \mathcal{N}(B) + O(X^{2n}),$$

ce qui permet de conclure. □

L'algorithme correspondant est une variante immédiate de l'algorithme 2, où les multiplications sont déjà dans le bon sens pour ce cadre non-commutatif. La complexité de cet algorithme s'exprime à l'aide de la fonction $\mathsf{MM}(k, N)$ complexité du produit de matrices $k \times k$ de polynômes de degré au plus N. Le Corollaire 1 du Chapitre 6 montre que $\mathsf{MM}(k, N) = O(k^\theta N + k^2\mathsf{M}(N))$.

Proposition 3. *Soit A une matrice $k \times k$ à coefficients dans $\mathbb{A}[[X]]$, avec $A(0)$ inversible. La matrice $A^{-1} \bmod X^N$ peut être calculée en $O(\mathsf{MM}(k, N))$ opérations dans $\mathbb{A}$.*

Démonstration. Comme pour l'inverse ci-dessus, le coût est dominé à une constante près par celui du calcul de $B + (I - BA)B \bmod X^N$: la complexité vérifie l'inégalité $C(N) \leq C(\lceil N/2\rceil) + 2\mathsf{MM}(k, N)$ et le résultat se déduit de l'hypothèse $\mathsf{MM}(k, N) \leq \frac{1}{2}\mathsf{MM}(k, 2N)$. □

Résolution d'équation en série

Entrée : un entier $N > 0$, $\Phi \bmod (X^N, Y^N)$ une série tronquée avec $\Phi(0,0) = 0$ et $(\partial\Phi/\partial Y)(0,0)$ inversible.

Sortie : $F \bmod X^N$, $G = (\partial\Phi/\partial Y(X,F))^{-1} \bmod X^{\lceil N/2 \rceil}$, telles que $F(0) = 0$, $\Phi(X,F) = 0$.

Si $N = 1$, alors renvoyer 0, $((\partial\Phi/\partial Y)(0,0))^{-1}$.
Sinon,

1. Calculer récursivement $F \bmod X^{\lceil N/2 \rceil}$ et $G \bmod X^{\lceil \lceil N/2 \rceil / 2 \rceil}$;
2. Calculer $G := G + (1 - G\frac{\partial\Phi}{\partial Y}(X,F))G \bmod X^{\lceil N/2 \rceil}$;
3. Calculer $F := F - G\Phi(X,F) \bmod X^N$;
4. Renvoyer F, G.

Figure 3: Résolution de $\Phi(X,Y) = 0$ par itération de Newton.

3. Itération de Newton formelle et applications

3.1. Un résultat général.

Théorème 1 (Itération de Newton sur les séries)**.** *Soit $\Phi \in \mathbb{A}[[X,Y]]$ une série bivariée en X et Y, telle que $\Phi(0,0) = 0$ et $\frac{\partial\Phi}{\partial Y}(0,0)$ est inversible dans $\mathbb{A}$. Il existe alors une série $S \in \mathbb{A}[[X]]$, telle que $\Phi(X,S) = 0$ et $S(0) = 0$. Si F est une série telle que $S - F = O(X^n)$ $(n \geq 1)$, alors*

$$\mathcal{N}(F) = F - \frac{\Phi(X,F)}{\frac{\partial\Phi}{\partial Y}(X,F)}$$

vérifie $S - \mathcal{N}(F) = O(X^{2n})$.

Ce résultat est un résultat local en $(0,0)$: si ce point est solution, il s'étend en une courbe solution. Par translation, d'autres situations où la solution n'est pas en $(0,0)$ s'y ramènent. La première partie est une version du théorème des fonctions implicites pour les séries formelles.

Démonstration. L'itération est bien définie : $\frac{\partial\Phi}{\partial Y}(X,F)$ est inversible parce que $F(0) = S(0) = 0$ et que son terme constant $\frac{\partial\Phi}{\partial Y}(0,0)$ est inversible. De ce fait, on déduit aussi $\mathcal{N}(F) - F = O(\Phi(X,F))$.

La valuation de $\Phi(X,F)$ est doublée par l'itération :

$$\begin{aligned} \Phi(X,\mathcal{N}(F)) &= \Phi(X,F) + \frac{\partial\Phi}{\partial Y}(X,F)(\mathcal{N}(F) - F) + O((\mathcal{N}(F) - F)^2) \\ &= O(\Phi(X,F)^2). \end{aligned} \tag{3}$$

La limite S de la suite définie par $S_0 = S$ et $S_{k+1} = \mathcal{N}(S_k)$ existe donc (la suite est de Cauchy d'après l'égalité qui précède) et vérifie $\Phi(X,S) = \lim(\Phi(X,S_k)) = 0$. La vitesse de convergence vers S vient encore de la formule de Taylor. En effet, l'égalité

$$\Phi(X,F) = \Phi(X,S) + \frac{\partial\Phi}{\partial Y}(X,F)(S - F) + O((S - F)^2) \tag{4}$$

donne $S - F = O(\Phi(X,F))$. □

3.2. Algorithme.

Lemme 4. *L'algorithme de résolution détaillé en Figure 3 est correct.*

Démonstration. Pour $N = 1$, les valeurs renvoyées sont les termes constants de g et f. Si la propriété est vraie pour $1 \leq k < N$, alors l'étape 1 renvoie $(\partial\Phi/\partial Y(X,f))^{-1} \bmod X^{\lceil N/2 \rceil}$ et $f \bmod X^{\lceil N/2 \rceil}$. D'après le lemme 1, l'étape 2 calcule $(\partial\Phi/\partial Y(X,f))^{-1} \bmod X^{N/2}$. L'étape 3 calcule alors $f \bmod X^N$: les formules (3) et (4) montrent que seuls les $N/2$ premiers coefficients de $\partial\Phi/\partial Y$ sont utiles pour doubler la précision. □

3.3. **Applications.** Un exemple d'application est fourni par l'inverse de la section précédente avec $\Phi(X, Y) = f - 1/(f(0)^{-1} + Y)$.

La complexité de l'algorithme de la Figure 3 dépend en particulier de celle du calcul de Φ et de $\partial\Phi/\partial Y$. Les applications les plus directes de la méthode de Newton sont résumées dans les théorèmes suivants.

Théorème 2. *Dans le cas où la série Φ du Théorème 1 est un polynôme de degré $O(1)$, les N premiers termes de sa solution série telle que $S(0) = 0$ sont obtenus en $O(\mathsf{M}(N))$ opérations dans $\mathbb{A}$.*

Démonstration. L'argument est le même que pour la complexité de l'inverse. Pour Φ polynôme de degré $O(1)$, les évaluations de Φ et de Φ' coûtent $O(\mathsf{M}(N))$, le reste de l'argument est inchangé. □

Dans le cas où Φ est un polynôme, un algorithme de meilleure complexité $O(N)$ (donc linéaire au lieu de quasi-linéaire !) sera présenté au Chapitre 8.

Théorème 3. *Soit f une série de $\mathbb{A}[[X]]$. On peut calculer en $O(\mathsf{M}(N))$ opérations dans $\mathbb{A}$ 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 également que $2, 3, \ldots, N-1$ sont des éléments inversibles de $\mathbb{A}$.

Démonstration. La preuve pour l'inverse et le logarithme provient des sections 2.3 et 2.6. L'exponentielle est traitée dans la section qui vient, et la puissance se déduit des précédentes par

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

Une application remarquable de cette identité évidente apparaît dans la section 3.7.2 sur l'inverse compositionnel. □

3.4. **Exponentielle.** Si $F \in \mathbb{A}[[X]]$ est telle que $F(0) = 0$, on définit la série $\exp(F)$ par

$$\exp(F) = 1 + F + F^2/2! + F^3/3! + \cdots$$

lorsque 2,3,... sont inversibles dans $\mathbb{A}$.

Pour calculer cette série, l'idée est d'appliquer une méthode de Newton avec

$$\Phi(Y) = F - \log Y, \quad Y(0) = 1.$$

Exercice 3. Montrer que les conditions du Théorème 1 sont vérifiées.

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

$$E_{k+1} = \mathcal{N}(E_k) = E_k + E_k(F - \log E_k).$$

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

Exercice 4. 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 5. 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 $\mathbb{K}$ 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 $p_i = \alpha_1^i + \cdots + \alpha_d^i$. Ce sont des éléments de $\mathbb{K}$, puisqu'ils sont fonctions symétriques des racines de P. Leur utilisation efficace repose sur la proposition suivante.

Proposition 4. *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. Si $P = c\prod_\alpha (X-\alpha)$, la décomposition en éléments simples

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

où α décrit les zéros de P comptés avec multiplicités, montre que les coefficients du développement en puissances de X^{-1} de XP'/P sont les p_i. Pour se ramener à des calculs de séries formelles, on introduit par exemple une nouvelle variable par $X = 1/T$. 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 4. *Soient P et Q deux polynômes unitaires de $\mathbb{K}[X]$, de degré d. Alors, les polynômes*

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

peuvent être calculés en $O\big(\mathsf{M}(d^2)\big)$ 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 sont présentés au Chapitre 7. 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.

Les polynômes constituent une structure de données bien commode pour coder des nombres algébriques et calculer avec : les 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-ième 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 multipliant les 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-ième coefficient par $i!$ pour obtenir la série génératrice des sommes de Newton du polynôme $P \otimes Q$, 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 (Théorème 1) sont vérifiées pour la fonction

$$\Phi(X, Y) = F(Y) - X.$$

L'opérateur de Newton correspondant est donc

$$\mathcal{N}(G) = G - \frac{F(G) - X}{F'(G)}.$$

La dérivation de $F(G) = X$ donne $F'(G)G' = 1$ et cet opérateur se simplifie en

$$\mathcal{N}(G) = G - G' \times (F(G) - X).$$

Le coût est alors dominé soit par celui du produit, soit plus généralement par celui de la composition par F (voir Section 4).

Exercice 6. La k-ième 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 s'écrit $a_i = f_i(1/\pi)$, où f_i est un polynôme impair de degré $2i+1$. Borner le nombre d'opérations dans $\mathbb{Q}$ nécessaires pour calculer $f_1, \dots, f_N$, pour N grand.

3.7.2. *Le N-ième terme.* Bien qu'en général le coût de l'inverse compositionnel 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-ième 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 $T \in \mathbb{Q}[[X]]$ des arbres généraux étiquetés (appelés aussi *arbres de Cayley*) vérifie l'équation

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

Ses coefficients sont obtenus en $O(\mathsf{M}(N))$ opérations dans $\mathbb{Q}$ en calculant d'abord ceux 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 utilisée en analyse d'algorithmes passe par le calcul du développement de T au voisinage du point singulier $\exp(-1)$, où T 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 T solution de $T = X\exp(T)$ de l'exemple précédent. L'asymptotique mentionnée plus haut donne alors la formule de Stirling.

3.8. **Systèmes.** Le théorème 1 s'étend à des systèmes d'équations.

Théorème 5. *Soient $\Phi = (\Phi_1, \ldots, \Phi_k)$ des séries en $k+1$ indéterminées X et $Y = (Y_1, \ldots, Y_k)$, telles que $\Phi(0) = 0$ et la matrice jacobienne $(\frac{\partial \Phi}{\partial Y})$ est inversible dans $\mathbb{A}$ en 0. Alors, le système*

$$\Phi_1(X, Y_1, \ldots, Y_k) = \cdots = \Phi_k(X, Y_1, \ldots, Y_k) = 0$$

admet une solution $S = (S_1, \ldots, S_k) \in \mathbb{A}[[X]]^k$ telle que $S(0) = 0$. Si $F = (F_1, \ldots, F_k)$ est tel que $S - F = O(X^n)$ $(n \geq 1)$, alors

$$\mathcal{N}(F) = \begin{pmatrix} F_1 \\ \vdots \\ F_k \end{pmatrix} - \left(\frac{\partial \Phi}{\partial Y}(X, F_1, \ldots, F_k)\right)^{-1} \cdot \begin{pmatrix} \Phi_1(X, F_1, \ldots, F_k) \\ \vdots \\ \Phi_k(X, F_1, \ldots, F_k) \end{pmatrix}$$

vérifie $S - \mathcal{N}(F) = O(X^{2n})$.

La preuve est la même que celle du théorème 1, la formule de Taylor pour les fonctions de plusieurs variables s'exprimant alors à l'aide de la matrice jacobienne.

Lemme 5. *L'algorithme 3 fonctionne sans changement pour un système Φ satisfaisant aux hypothèses du théorème 5.*

Démonstration. La preuve est la même, en observant que l'étape 2 est écrite de telle sorte que le lemme 3 s'applique : cette étape calcule l'inverse de la matrice jacobienne à la précision requise et la dernière étape effectue un produit matrice-vecteur pour en déduire la correction de la nouvelle itération. □

Corollaire 1 (Systèmes polynomiaux)**.** *Sous les mêmes hypothèses, si $\Phi_i \in \mathbb{A}[[X]][Y_1, \ldots, Y_k]$ pour $1 \leq i \leq k$, alors les solutions séries à précision N peuvent être calculées en $O(\mathsf{MM}(k, N))$ opérations dans $\mathbb{A}$.*

Démonstration. L'étape 2 demande des produits de matrices $k \times k$ de polynômes de degré $\lceil N/2 \rceil$, et l'étape 3 demande un produit matrice-vecteur en précision N. La complexité de l'application récursive est donc dominée par celle de l'étape 2, le résultat est donc le même que pour l'inverse de matrices de séries. □

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 $F(G(X))$. À la différence des algorithmes vus dans les sections précédentes, il s'avère que les algorithmes connus n'atteignent pas une complexité quasi-optimale (c'est-à-dire linéaire en N, à des facteurs logarithmiques près).

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

$$F(G(X)) = f_0 + G(f_1 + G(f_2 + \ldots)) + O(X^N).$$

Exercice 7. 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 $O(\sqrt{N}(\mathsf{M}(N) + \mathsf{MM}(\sqrt{N}))) = O(\sqrt{N}(\mathsf{M}(N) + N^{\theta/2}))$, où θ est un exposant faisable pour la complexité du produit de matrices, présenté au Chapitre 3. La série F est d'abord écrite

$$F(X) = F_0(X) + X^k F_1(X) + \cdots + X^{k\lceil N/k \rceil} F_{\lceil N/k \rceil}(X) + O(X^N),$$

où les polynômes F_i en nombre $1 + \lceil N/k \rceil$ comportent chacun k termes. Ensuite, on calcule les puissances $G^2, \ldots, G^k$ 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 $k \times k$ — la première devient un vecteur colonne d'environ N/k^2 blocs, la seconde un vecteur ligne d'environ N/k blocs —, ce produit est effectué en au plus $O(N^2 k^{\theta-3})$ 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 de la composition au même ordre que $\mathsf{M}(N)$, il est possible de faire sensiblement mieux que la méthode de 4.2, en évitant la multiplication matricielle, grâce à un algorithme sophistiqué dû à Brent & Kung, détaillé dans cette section.

Théorème 6. *Si* $2, 3, \ldots, \lceil \sqrt{N \log N} \rceil$ *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

$$F(G_1 + X^m G_2) = F(G_1) + F'(G_1)X^m G_2 + F''(G_1)\frac{X^{2m}G_2^2}{2!} + \cdots, \tag{5}$$

où G_1 est un polynôme de degré inférieur à m ; le choix de m est ajusté plus loin pour minimiser la complexité.

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

Lemme 6. *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 (déjà utilisée en page 135) :

$$F' \circ G = \frac{(F \circ G)'}{G'}.$$

L'hypothèse sur $G'(0)$ permet de garantir l'inversion de la série du dénominateur. □

L'utilisation répétée de cette idée dans la formule (5) 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 avec un *polynôme*, donnée par le lemme suivant.

Lemme 7. *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 fournit l'estimation $C(F(G_1)) = O(m\mathsf{M}(N)\log N)$. 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. Sinon, il suffit de considérer 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 théorème « diviser pour régner » conclut la preuve du lemme. □

Exercices

Exercice 8 (Exponentielle de série)**.** Soit s une série de terme constant nul et $n = 2^p$. On étudie l'itération

$$\begin{aligned} z_k &= z_{k-1} + z_{k-1}(1 - y_{k-1}z_{k-1}) \mod X^{2^{k-1}}, \\ y_k &= y_{k-1} - y_{k-1} \int z_k(y'_{k-1} - s'y_{k-1}) \mod X^{2^k} \end{aligned}$$

avec $y_0 = 1 = z_0$.

1. Montrer que $y_p - \exp(s) = O(X^n)$;
2. Estimer la complexité du calcul des n premiers coefficients de la série $\exp(s)$ par cette itération ;
3. (Plus difficile) interpréter cette itération comme l'itération de Newton associée à l'opérateur $Y \mapsto Y' - s'Y$.

Exercice 9 (Composition avec l'exponentielle)**.** Soit $f(x) \in \mathbb{Q}[[X]]$ une série à coefficients rationnels. Le but de cet exercice est de montrer que les N premiers coefficients de la série $S(X) = f(e^X - 1)$ peuvent être calculés en $O\big(\mathsf{M}(N)\log N\big)$ opérations arithmétiques dans $\mathbb{Q}$.

1. Montrer que $S(X) - A(e^X - 1) = O(X^N)$, où $A(X)$ est l'unique polynôme tel que $f(X) = A(X) + O(X^N)$.
2. On pose $B(X) = A(X - 1)$; donner un algorithme pour calculer les coefficients de B à partir de ceux de A en $O\big(\mathsf{M}(N)\log N\big)$ opérations dans $\mathbb{Q}$.
3. La transformée de Laplace formelle $\mathcal{L}$ est définie sur $\mathbb{Q}[[X]]$ comme l'application $\mathbb{Q}$-linéaire telle que $\mathcal{L}(X^k) = k!\,X^k$, $k \in \mathbb{N}$. Si $B(X) = b_0 + b_1X + \cdots + b_{N-1}X^{N-1}$, montrer que
$$\mathcal{L}\big(B(e^X)\big) = \sum_{i=0}^{N-1} \frac{b_i}{1 - iX}.$$
4. Donner un algorithme pour calculer les N premiers coefficients de $B(e^X)$ à partir de ceux de $B(X)$ en $O\big(\mathsf{M}(N)\log N\big)$ opérations dans $\mathbb{Q}$.
5. Donner finalement un algorithme calculant les N premiers coefficients de S à partir de ceux de f en $O\big(\mathsf{M}(N)\log N\big)$ opérations dans $\mathbb{Q}$.

Exercice 10 (Composition de séries avec arcsinus)**.** Soit $\mathbb{K}$ un corps de caractéristique nulle, et soit $F \in \mathbb{K}[X]$ une série formelle de terme constant nul. Écrire un algorithme à base d'itérations de Newton permettant de calculer les N premiers termes de la série composée $(\arcsin \circ F)$, à partir des N premiers termes de la série F, en $O(\mathsf{M}(N))$ opérations arithmétiques dans $\mathbb{K}$.

Notes

Lorsque Newton décrit sa méthode de résolution des équations [16], il traite non seulement de la recherche de solutions numériques, mais aussi de la recherche de solutions séries, et va jusqu'à montrer l'usage de sa méthode pour les solutions de polynômes à coefficients des séries (la première partie de notre théorème 2, sans la complexité). La formule d'inversion évoquée en page 135 a été découverte par Lagrange en 1768 [14].

L'utilisation de l'itération de Newton en informatique est très ancienne. L'inversion de matrice de la section 2.7 remonte à Schulz [20] pour des matrices réelles. L'importance de la méthode de Newton pour le calcul rapide avec des séries formelles a été soulignée par Lipson [15] : "*In a symbolic mathematics system setting, we feel that Newton's method offers a unified and easily implemented algorithm for solving a wide variety of power series manipulation problems that can be cast into a root-finding mold. Our experience attests to the practicality of Newton's method. We feel that it is a great algebraic algorithm.*"

Du point de vue de la complexité, Sieveking [21] a donné l'algorithme d'inversion rapide de séries (notre Figure 2), et Kung [13] a observé qu'il s'agissait d'une itération de Newton et il a amélioré la constante dans la complexité. Ensuite, Brent [7] a montré comment l'exponentielle et le logarithme s'obtiennent également en $O(\mathsf{M}(N))$ opérations. Les algorithmes de la section 3.5 sont dus à Schönhage [19] ; l'application aux sommes et produits composés en section 3.6 est beaucoup plus récente et se trouve dans [4]. La méthode de Newton permet également la résolution de systèmes linéaires à coefficients polynomiaux (Chapitre 14) et la résolution d'équations ou de systèmes *différentiels* (Chapitre 15). L'idée d'incorporer le dividende dans la dernière itération de Newton, afin de gagner un facteur constant pour la division de séries (Section 2.4), est due à [11].

Les algorithmes non triviaux de composition décrits en Section 4 sont dus à Brent et Kung [6]. L'idée de l'algorithme *pas de bébés, pas de géants* en Section 4.2 remonte à [17]. La composition $F \circ G \bmod X^N$ peut s'effectuer en complexité quasi-optimale $O(\mathsf{M}(N)\log N)$ si G est un polynôme [6], ou plus généralement lorsque G est une série algébrique [22]. Des compositions avec d'autres fonctions particulières peuvent aussi être calculées en $O(\mathsf{M}(N)\log N)$, ou parfois $O(\mathsf{M}(N))$, voir [5]. Savoir si cela est encore possible pour des séries quelconques est un grand problème ouvert.

Dans le cas où la caractéristique p de $\mathbb{A}$ est finie, Bernstein [2] a montré qu'il est possible de descendre à une complexité quasi-optimale en la précision N des séries, pour le prix d'une dépendence linéaire en p. Cet algorithme est intéressant lorsque la caractéristique p est petite. Une avancée récente importante est due à Kedlaya et Umans [12], qui ont donné le premier algorithme de complexité binaire quasi-linéaire à la fois en N et en $\log(p)$, pour le calcul de $F \circ G \bmod X^N$, lorsque F et G sont des séries à coefficients dans le corps fini $\mathbb{F}_p$.

Ritzmann [18] a montré que la composition des séries entières peut s'effectuer en complexité binaire quasi-optimale : si $F, G \in \mathbb{Z}[X]$ ont des coefficients bornés par ℓ en module, alors il est possible de calculer $F \circ G \bmod X^N$ en $O(\mathsf{M}_{\mathbb{Z}}(N^2 \log N \log \ell))$ opérations binaires. L'article [18] montre également que le problème de la composition de séries à coefficients dans un corps $\mathbb{K}$ est d'une difficulté équivalente à celui du calcul du N-ième coefficient de $F \circ G$. L'article [6] montre que le problème de l'inverse compositionnel leur est également équivalent.

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. Dans le modèle de multiplication polynomiale par FFT, les meilleurs constantes devant $\mathsf{M}(N)$ sont dues à Harvey [9, 10] : $\frac{13}{9}$ pour l'inverse, $\frac{4}{3}$ pour la racine carrée, $\frac{13}{6}$ pour l'exponentielle. Les articles de synthèse [1, 3] décrivent diverses techniques utilisées pour éliminer certains calculs redondants dans la méthode de Newton.

L'exercice 8 est inspiré par [8], et l'exercice 9 est tiré de [5].

Bibliographie

[1] Bernstein (D. J.). – Removing redundancy in high-precision Newton iteration. – Preprint, available from `http://cr.yp.to/fastnewton.html#fastnewton-paper`.

[2] Bernstein (Daniel J.). – Composing power series over a finite ring in essentially linear time. *Journal of Symbolic Computation*, vol. 26, n3, 1998, pp. 339–341.

[3] Bernstein (Daniel J.). – Fast multiplication and its applications. In *Algorithmic number theory : lattices, number fields, curves and cryptography*, pp. 325–384. – Cambridge Univ. Press, 2008.

[4] Bostan (Alin), Flajolet (Philippe), Salvy (Bruno), and Schost (Éric). – Fast computation of special resultants. *Journal of Symbolic Computation*, vol. 41, n1, January 2006, pp. 1–29.

[5] Bostan (Alin), Salvy (Bruno), and Schost (Éric). – Power series composition and change of basis. In *ISSAC'08*, pp. 269–276. – ACM, New York, 2008.

[6] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n4, 1978, pp. 581–595.

[7] Brent (Richard P.). – Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity*, pp. 151–176. – Academic Press, New York, 1976. Proceedings of a Symposium held at Carnegie-Mellon University, Pittsburgh, Pa., 1975.

[8] Hanrot (Guillaume) and Zimmermann (Paul). – Newton iteration revisited. – Available at `http://www.loria.fr/~zimmerma/papers`.

[9] Harvey (David). – Faster algorithms for the square root and reciprocal of power series. *Mathematics of Computation*. – To appear. Available at `http://arxiv.org/abs/0910.1926/`.

[10] Harvey (David). – Faster exponentials of power series. – Preprint, available from `http://arxiv.org/abs/0911.3110`.

[11] Karp (A. H.) and Markstein (P.). – High-precision division and square root. *ACM Transactions on Mathematical Software*, vol. 23, n4, December 1997, pp. 561–589.

[12] Kedlaya (Kiran S.) and Umans (Christopher). – Fast modular composition in any characteristic. In *FOCS '08 : Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*. pp. 146–155. – IEEE Computer Society, Washington, DC, USA, 2008.

[13] Kung (H. T.). – On computing reciprocals of power series. *Numerische Mathematik*, vol. 22, 1974, pp. 341–348.

[14] Lagrange (Joseph-Louis). – Nouvelle méthode pour résoudre les équations littérales par le moyen des séries. *Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Berlin*, vol. 24, 1768, pp. 251–326.

[15] Lipson (J. D.). – Newton's Method : A Great Algebraic Algorithm. In *Proceedings ACM Symposium of Symbolic and Algebraic Computation*. pp. 260–270. – ACM Press, 1976.

[16] Newton (Isaac). – *La méthode des fluxions, et les suites infinies*. – de Bure aîné, 1740. Traduction française par G. Buffon du texte de 1671 de Newton. Disponible en ligne à `http://gallica.bnf.fr`.

[17] Paterson (M. S.) and Stockmeyer (L. J.). – On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, vol. 2, n1, 1973, pp. 60–66.

[18] Ritzmann (P.). – A fast numerical algorithm for the composition of power series with complex coefficients. *Theoretical Computer Science*, vol. 44, 1986, pp. 1–16.

[19] Schönhage (A.). – *The fundamental theorem of algebra in terms of computational complexity*. – Technical report, Univ. Tübingen, 1982. 73 pages.

[20] Schulz (G.). – Iterative Berechnung der reziproken Matrix. *Zeitschrift für angewandte Mathematik und Physik*, vol. 13, 1933, pp. 57–59.

[21] Sieveking (M.). – An algorithm for division of powerseries. *Computing*, vol. 10, 1972, pp. 153–156.

[22] van der Hoeven (Joris). – Relax, but don't be too lazy. *Journal of Symbolic Computation*, vol. 34, n6, 2002, pp. 479–542.

CHAPITRE 5

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 sont utilisés au Chapitre 6 pour l'évaluation multipoint et l'interpolation et au Chapitre 7 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 F/G *à l'infini* a ses premiers termes donnés par Q, puisque 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 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 $(T^m F(1/T))/(T^n G(1/T)) \bmod T^{m-n+1}$ 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 2 du Chapitre 4 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.

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. Développement de fractions rationnelles

Si $\mathbb{A}$ est un anneau commutatif, 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 habituelles 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 1. 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 au Chapitre 4. 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 le théorème suivant. Le problème inverse, tout aussi fondamental, consiste à retrouver la forme rationnelle à partir des premiers termes de la série rationnelle, et est traité efficacement au Chapitre 9.

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-ième 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-ième puissance de x dans $\mathbb{B}$. Or, cela peut se faire en $O(\log N)$ multiplications par exponentiation binaire 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 du Théorème 3. 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 éléments 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 en 1202 par Leonardo Pisano (mieux connu sous le pseudonyme de Fibonacci) 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 2. 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 [1] : 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 [2].

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 ») [3] 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 3. Soit M une matrice de taille $d \times d$ à coefficients dans $\mathbb{A}$, de polynôme caractéristique $\chi_M(X) = \det(XI_d - M)$, soient 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 4. 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.

Exercice 5. Si $P \in \mathbb{A}[X]$ est de degré d, alors la suite $(P(n))_{n\geq 0}$ est une srlcc de polynôme caractéristique $(X-1)^{d+1}$.

---

1. Le journal *The Fibonacci Quarterly* est entièrement dédié à l'étude de leurs propriétés.
2. 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.
3. On pourra admirer $F_{1\,000\,000}$ à l'url `http://www.upl.cs.wisc.edu/~bethenco/fibo/`

3.2.1. *Exponentiation binaire.* L'approche naïve pour le calcul du N-ième 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

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

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 6. Montrer que le N-ième 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 $O(\mathsf{MM}(d) \log N) = O(d^\theta \log N)$ opérations dans $\mathbb{A}$, où $2 \leq \theta < 2,38$ (voir le Chapitre 3). 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 6) 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 linéaires, c'est-à-dire solutions de récurrences linéaires à coefficients constants. Ce lien est précisé dans le résultat suivant

Lemme 2. *Soit $A(X) = \sum_{n \geq 0} a_n X^n$ la série génératrice de la suite $(a_n)_{n \geq 0}$. Soit $P \in \mathbb{K}[X] \setminus \{0\}$ de degré d et soit $\bar{P} = P(1/X)X^d$ son polynôme réciproque. Les assertions suivantes sont équivalentes :*

(i) P est un polynôme caractéristique de (a_n) ;

(ii) $A(X) = N_0/\bar{P}$ pour un certain $N_0 \in \mathbb{K}[X]$ avec $\deg(N_0) < d$.

De plus, si P est le polynôme minimal de (a_n), alors $d = \max\{1 + \deg(N_0), \deg(\bar{P})\}$ et $\operatorname{pgcd}(N_0, \bar{P}) = 1$.

Exercice 7. Prouver ce lemme.

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

Le lemme 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. Au vu du Théorème 3, cela entraîne le résultat suivant.

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

Exercice 8. 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 d'Hadamard $\mathbf{a} \star_{\mathbf{H}} \mathbf{b} = (a_n b_n)_n$ est une srlcc de polynôme caractéristique égal au produit composé $P \otimes Q$ défini au Chapitre 4 ;

3. la suite $\left(\sum_{i=0}^{n} \binom{n}{i} a_i b_{n-i}\right)_n$ est une srlcc de polynôme caractéristique égal à la somme composée $P \oplus Q$ définie au Chapitre 4.

Ils se calculent donc tous en bonne complexité (quasi-linéaire en la taille de la sortie).

Exercice 9. Prouver les assertions précédentes.

Exercice 10. Lorsque les coefficients de la récurrence sont des entiers, étudier la complexité binaire de tous les algorithmes traités dans ce chapitre.

3.4. **Application : tests de primalité.** Une application du calcul rapide d'un terme d'une récurrence est une famille de tests probabilistes 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 11. 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 12. 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 13. 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 clé est que *le quotient de la division euclidienne de deux polynômes ne dépend que de leurs coefficients de poids fort*. Cette remarque est également à la base du calcul rapide de pgcd, étudié au Chapitre 7.

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 au Chapitre 12.

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 complète.

L'exercice 3 est le point clé de la méthode de Wiedemann pour la résolution de systèmes linéaires creux, traitée au Chapitre 10.

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 6) 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, exposé au Chapitre 14.

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 dans l'exercice 13, 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 13 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, n2, 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. 33, 1987, pp. 67–108.

[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, n1, 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. In *Thirteenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 90–96. – 1972.

[8] Montgomery (Peter L.). – Modular multiplication without trial division. *Mathematics of Computation*, vol. 44, n170, 1985, pp. 519–521.

[9] Shoup (V.). – A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In *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.

CHAPITRE 6

Calculs modulaires, évaluation et interpolation

Résumé

Le concept d'algorithme modulaire est central en calcul formel. Il permet de pallier le phénomène d'explosion des expressions intermédiaires. Des cas particuliers importants sont l'évaluation multipoint et l'interpolation, pour lesquels il existe des algorithmes rapides qui peuvent être vus comme des généralisations de la FFT.

1. Introduction

Un ingrédient essentiel en calcul formel est l'utilisation de différents types de représentation 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 Chapitre 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 au Chapitre 5) 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 *modulos* 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 garantir l'unicité du résultat et la correction de ce schéma, il suffit que les modulos soient suffisamment *nombreux et indépendants*. Typiquement, s'il s'agit d'entiers, ils peuvent 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 d'*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 $n \times n$, dont les éléments sont des polynômes de degré 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-ième 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 éléments 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 modulos. Dans certaines situations, le programmeur a le choix des modulos. Dans l'exemple précédent, on peut choisir comme modulos 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 lesquelles, même si elle est faisable, la FFT n'est pas encore rentable ; ce choix est alors déconseillé. Dans d'autres situations, le choix des modulos est intrinsèquement imposé par le problème à traiter ; c'est le cas pour le calcul de la factorielle exposé au Chapitre 13.

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 $a_0, \dots, a_{n-1}$ 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 :

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

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

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

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 $\mathbf{V} = \left(a_{i-1}^{j-1}\right)_{i,j=1}^{n}$ est la matrice de Vandermonde associée aux points a_i, dont le déterminant vaut $\det(\mathbf{V}) = \prod_{i<j} \left(a_j - a_i\right)$, alors le problème d'interpolation se traduit par la résolution en l'inconnue $\mathbf{p}$ du système linéaire $\mathbf{Vp} = \mathbf{b}$, 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 $\det(\mathbf{V})$ et donc aussi celle de la matrice $\mathbf{V}$.

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

Cette discussion suggère un algorithme naïf pour l'interpolation, produisant l'unique solution $\mathbf{p} = \mathbf{V}^{-1}\mathbf{b}$ du système $\mathbf{Vp} = \mathbf{b}$ à l'aide du pivot de Gauss en $O(n^3)$ opérations dans $\mathbb{A}$, ou encore en $O(\mathsf{MM}(n)) = O(n^\theta)$ opérations ($2 \leq \theta < 3$), en utilisant l'algorithmique matricielle rapide (Chapitre 3). De manière analogue, l'évaluation multipoint de P en les a_i se traduit par le produit matrice-vecteur $\mathbf{Vp}$, 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 coûteuse 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 chapitre est de montrer qu'il est possible d'atteindre une 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 (« Évaluation interpolation »)**.** *On peut effectuer l'évaluation et l'interpolation sur n points vérifiant l'hypothèse* (H) *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 forment une progression géométrique de raison inversible dans $\mathbb{A}$.*

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 aux Chapitres 7 et 9 pour le calcul rapide de pgcd ou d'approximants de Padé-Hermite ; 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 donnés des polynômes $P, m_1, \ldots, m_r$ dans $\mathbb{A}[X]$, avec $\deg(P) < n = \sum_i \deg(m_i)$, 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, trouver $P \in \mathbb{A}[X]$ de degré inférieur à $n = \sum_i \deg(m_i)$ tel que

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

Les techniques de ce chapitre 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 modulos entiers a_i et où l'on cherche un $P \in \mathbb{Z}$ 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 chapitre est organisée comme suit : dans la Section 3 est décrite 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.

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 et a_i les points d'interpolation vérifiant l'hypothèse (H). 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 $A_i(a_j) = 0$. 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 effectuer 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.

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 Chapitre 7 pour la définition du résultant.

Interpolation de Lagrange

Entrée : $a_0, \dots, a_{n-1} \in \mathbb{A}$ vérifiant (H) et $b_0, \dots, b_{n-1}$ 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 $i = 0, \dots, n-1$ faire
   $A \leftarrow A \cdot (X - a_i)$
3. Pour $i = 0, \dots, n-1$ 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 un coût quadratique. □

4. Algorithmes rapides

4.1. **Les idées.** L'idée fondamentale de l'algorithme d'évaluation multipoint rapide est 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 d'associer à P 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 cette construction, le choix suivant s'impose :

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

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

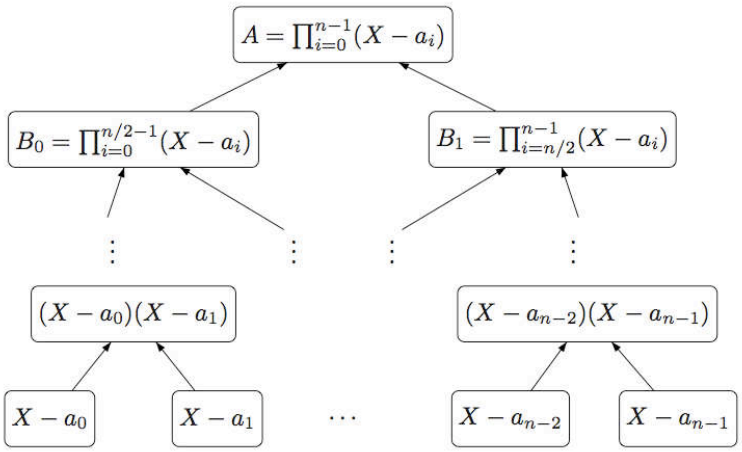
On a donc ramené le calcul en degré n à deux calculs en degré $n/2$, plus deux divisions euclidiennes ; on va ensuite itérer ce processus. En bout de course, on retrouve les valeurs $P(a_i)$, car celles-ci peuvent également s'écrire $P \bmod (X - a_i)$.

Remarquons que les polynômes par lesquels on effectue les divisions euclidiennes ne sont pas « gratuits » : il convient de les (pré)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 $X - a_0$.
- Sinon, 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 B_0 et B_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 $B_0 B_1$ et dont les fils sont $\mathcal{B}_0$ et $\mathcal{B}_1$.

Graphiquement, on a la représentation suivante :



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.

ArbreSousProduits

Entrée : $a_0, \ldots, a_{n-1}$ dans $\mathbb{A}$, avec n de la forme $n = 2^\kappa$ ($\kappa \in \mathbb{N}$).

Sortie : l'arbre $\mathcal{B}$ des sous-produits associé aux points a_i.

1. Si $n = 1$, renvoyer $X - a_0$
2. Calculer (par un appel récursif) $\mathcal{B}_0 = \mathsf{ArbreSousProduits}(a_0, \ldots, a_{n/2-1})$
3. Calculer (par un appel récursif) $\mathcal{B}_1 = \mathsf{ArbreSousProduits}(a_{n/2}, \ldots, a_{n-1})$
4. Calculer $A = B_0 B_1$, où B_0 et B_1 sont les racines de $\mathcal{B}_0$ et de $\mathcal{B}_1$.
5. Renvoyer l'arbre de racine A et dont les fils sont $\mathcal{B}_0$ et $\mathcal{B}_1$.

FIGURE 1: Algorithme rapide pour le calcul de l'arbre des sous-produits.

Proposition 2. *L'algorithme* ArbreSousProduits *calcule tous les polynômes contenus dans l'arbre* $\mathcal{B}$ *pour* $O(\mathsf{M}(n)\log n)$ *opérations arithmétiques 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 en invoquant le théorème « diviser pour régner ». □

4.3. **Évaluation multipoint rapide.** L'algorithme d'évaluation multipoint 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

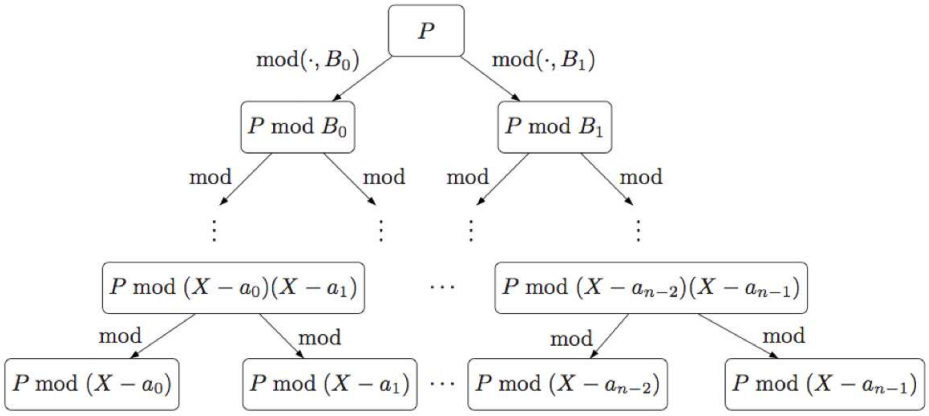
Entrée : $a_0, \ldots, a_{n-1}$ dans $\mathbb{A}$, avec n de la forme $n = 2^\kappa$ ($\kappa \in \mathbb{N}$),
l'arbre $\mathcal{B}$ des sous-produits associé aux points a_i,
P dans $\mathbb{A}[X]$, avec $\deg P < n$.

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

1. Si $n = 1$, 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

Figure 2: Algorithme rapide d'évaluation multipoint par division répétée.

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 peut s'effectuer en $O(\mathsf{M}(n))$ opérations de $\mathbb{A}$ (Chapitre 5). Nous obtenons le résultat de complexité suivant :

Proposition 3. *La complexité de l'algorithme* EvaluationRapide (Figure 2) *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 $\mathsf{T}(n)$:

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

et le résultat en découle grâce au théorème « diviser pour régner ». □

Exercice 2. Si $\omega \in \mathbb{A}$ est une racine primitive n-ième de l'unité, montrer que l'algorithme FFT vu au Chapitre 2 est un cas particulier de l'algorithme en Figure 2.

Exercice 3. Estimer la complexité de la variante de l'algorithme EvaluationRapide dans laquelle l'arbre des sous-produits n'est pas précalculé, les polynômes modulo lesquels s'effectuent les divisions étant calculés à la volée.

4.4. **Sommes de fractions.** Un problème intermédiaire à traiter avant de résoudre efficacement le problème d'interpolation est celui du calcul rapide 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

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

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. $(d_0, \ldots, d_{n-1}) \leftarrow \mathsf{EvaluationRapide}(A', (a_0, \ldots, a_{n-1}))$
3. $(c_0, \ldots, c_{n-1}) \leftarrow (b_0/d_0, \ldots, b_{n-1}/d_{n-1})$
4. $R \leftarrow \mathsf{SommesFractions}(c_0/(X - a_0), \ldots, c_{n-1}/(X - a_{n-1}))$
5. Renvoyer le numérateur de R.

FIGURE 3: Algorithme d'interpolation rapide.

Il était apparu que pour effectuer l'interpolation, il suffisait de 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

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

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 :

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

d'où l'on déduit le résultat suivant, en faisant encore une fois appel au théorème « 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* $\mathbb{A}$.

4.5. **Interpolation.** 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 $c_i = b_i/A_i(a_i)$. Le calcul itératif des constantes c_i reviendrait à évaluer n polynômes différents, chacun en un point, et aurait donc une complexité trop élevée, quadratique en n. La proposition suivante permet de remplacer ces n évaluations par une évaluation multipoint d'un unique polynôme en n points, ce qui mène à un gain de complexité.

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

Démonstration. On dérive A, ce qui fournit l'égalité :

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

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.

Proposition 6. *La complexité de l'algorithme* InterpolationRapide *en Figure 3 est* $O(\mathsf{M}(n)\log n)$ *opérations de* $\mathbb{A}$.

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

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 $\{1, q, \dots, q^{n-1}\}$ en $O(\mathsf{M}(n))$ opérations.

Proposition 7. *Soit* q *un élément inversible de* $\mathbb{A}$, *et* $a_i = q^i$ *pour* $0 \le i < n$. *Alors :*

1. *On peut évaluer tout polynôme* $P \in \mathbb{A}[X]$ *de degré strictement inférieur à* n *sur les points* $a_0, \dots, a_{n-1}$ *en* $O(\mathsf{M}(n))$ *opérations dans* $\mathbb{A}$.
2. *Si les points* (a_i) *vérifient l'hypothèse* (H) *on peut effectuer l'interpolation en les points* $a_0, \dots, a_{n-1}$ *en* $O(\mathsf{M}(n))$ *opérations dans* $\mathbb{A}$.

Démonstration. Soit $P = p_0 + p_1 X + \cdots + p_{n-1}X^{n-1}$. Pour $i = 0, \dots, 2n-2$, on introduit les nombres triangulaires $t_i = i(i-1)/2$ et la suite $\beta_i = q^{t_i}$; pour $i = 0, \dots, n-1$ on construit $\gamma_i = p_i/\beta_i$. Tous les éléments q^{t_i}, γ_i et β_i peuvent être calculés en $O(n)$ opérations, en utilisant l'égalité $q^{t_{i+1}} = q^i q^{t_i}$. L'observation clé est que $ij = t_{i+j} - t_i - t_j$ et donc $q^{ij} = \beta_i^{-1}\beta_j^{-1}\beta_{i+j}$. L'algorithme d'évaluation repose alors sur la formule

$$P(q^i) = \sum_{j=0}^{n-1} p_j q^{ij} = \beta_i^{-1} \cdot \sum_{j=0}^{n-1} \gamma_j \beta_{i+j},$$

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

Pour l'interpolation, on adapte l'algorithme InterpolationRapide en Figure 3, de façon à économiser un facteur log en tirant parti de la structure des points $a_i = q^i$.

On commence par remarquer que *la racine* $A(X)$ de l'arbre des sous-produits peut être calculée en $O(\mathsf{M}(n))$ opérations dans $\mathbb{A}$, par un algorithme *qui ne nécessite pas la construction de tout l'arbre des sous-produits*, donc différent de l'algorithme donné en Figure 1. En effet, puisque les points a_i forment une suite géométrique, il est possible de calculer récursivement $A = \prod_{i=0}^{n-1}(X - a_i)$ par *un seul appel récursif* calculant $B_0 = \prod_{i=0}^{n/2-1}(X - a_i)$, suivi du calcul de $B_1 = \prod_{i=n/2}^{n-1}(X - a_i)$ à partir de B_0 par une homothétie de coût $O(n)$, et enfin du calcul du produit $A = B_0 B_1$.

Par ailleurs, observons que dans l'algorithme InterpolationRapide, le calcul de tout l'arbre des sous-produits a été utilisé *uniquement* pour l'évaluation multipoint de A'. Or, par la première partie de la Proposition 7, l'évaluation de A' sur les points a_i peut s'effectuer en $O(\mathsf{M}(n))$ opérations.

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 $1/(a - X) = \sum_{i \ge j} a^{-j-1}X^j$ 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 $q^{-1}, q^{-2}, \dots, q^{-n}$ étant en progression géométrique, on sait évaluer C sur ces points en $O(\mathsf{M}(n))$. □

Exercice 4. Montrer que tout l'arbre des sous-produits associé aux points $a_i = q^i$, $i = 0, \dots, n-1$, peut être calculé en $\mathsf{M}(n) + O(n \log n)$ opérations dans $\mathbb{A}$.

Une approche évaluation-interpolation reposant sur la Proposition 7 permet d'obtenir un bon algorithme pour la multiplication des matrices polynomiales.

Corollaire 1. *Soit $\mathbb{K}$ un corps possédant au moins $2d$ éléments. Il est possible de multiplier deux matrices polynomiales de $\mathcal{M}_n(\mathbb{K}[X])$ de degré au plus d en $\mathsf{MM}(n,d) = O(d\,\mathsf{MM}(n) + n^2\,\mathsf{M}(d))$ opérations dans $\mathbb{K}$.*

Exercices

Exercice 5 (Calcul rapide de sommes de puissances)**.** Soient $x_1, x_2, \ldots, x_n$ des éléments de $\mathbb{K}$ et soit $P_k = \sum_{i=1}^{n} x_i^k$ pour $k \geq 1$. Montrer que, à partir de la donnée des $x_1, \ldots, x_n$, il est possible de calculer :

1. tous les P_k dont les indices $k \leq n$ sont une puissance de 2 en $O(n \log(n))$ opérations dans $\mathbb{K}$.
2. tous les $P_1, \ldots, P_n$ en $O(\mathsf{M}(n) \log(n))$ opérations dans $\mathbb{K}$.

Exercice 6. 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.

Exercice 7. 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 $\mathbb{A}$.

Exercice 8. Soit $\mathbb{K}$ un corps, soient $m_1, m_2 \in \mathbb{K}[X]$ des polynômes premiers entre eux, de degrés au plus n, et soient $v_1, v_2 \in \mathbb{K}[X]$ de degrés inférieurs à n. Donner un algorithme de complexité $O(\mathsf{M}(n) \log n)$ qui calcule $f \in \mathbb{K}[X]$ de degré inférieur à $2n$ tel que $f = v_1 \bmod m_1$ et $f = v_2 \bmod m_2$. [Indication : utiliser un pgcd étendu.]

En déduire un algorithme de type « diviser pour régner » pour l'interpolation polynomiale en degré n, de complexité $O(\mathsf{M}(n) \log^2 n)$.

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 [15] et prouvé par Pan [16]. 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 en n.

Une forme particulière du théorème des restes chinois a été enoncée il y a plus de 2000 ans par le mathématicien chinois Sun Tsu. Le traitement moderne est dû à Gauss [9]. 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 $x \bmod I \mapsto (x \bmod I_1, \ldots, x \bmod I_r)$. La formule d'interpolation de Lagrange est due à Waring [23]. Les avantages pratiques de l'arithmétique modulaire et les premières applications en informatique du théorème des restes chinois ont été mis en évidence dans les années 1950 par Svoboda et Valach [21], et indépendamment par Garner [8] et Takahasi et Ishibashi [22].

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 [19]. Horowitz [11] 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 [11].

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 [13, 3], reposant sur les algorithmes quasi-optimaux pour la division de [13, 20]. Montgomery [14] 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 chapitre vient de [17, 2]. À l'origine, cet algorithme a permis de montrer qu'une DFT_ω, pour ω racine primitive n-ième de l'unité, peut être effectuée

en $O(n \log n)$ opérations même si n n'est pas une puissance de 2. L'algorithme d'interpolation est une adaptation de [12]. 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 [20] a prouvé une borne inférieure de type $n \log n$. Shoup et Smolensky [18] 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 de tels algorithmes existent dans les cas particuliers où les a_i forment une progression arithmétique ou géométrique [6].

L'exercice 8 provient de [10] ; historiquement, ce fut le premier algorithme rapide pour les restes chinois. Les exercices 6 et 7 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, n4, 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, n3, 1974, pp. 366–386.

[4] Borodin (A.) and Munro (I.). – Evaluation of polynomials at many points. *Information Processing Letters*, vol. 1, n2, 1971, pp. 66–68.

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

[6] Bostan (Alin) and Schost (Éric). – Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, vol. 21, n4, 2005, pp. 420–446.

[7] Fiduccia (C. M.). – Polynomial evaluation via the division algorithm : The fast Fourier transform revisited. In *4th ACM Symposium on Theory of Computing*, pp. 88–93. – 1972.

[8] Garner (Harvey L.). – The residue number system. In *IRE-AIEE-ACM'59 Western joint computer conference*. pp. 146–153. – ACM, NY, USA, 1959.

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

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

[11] Horowitz (E.). – A fast method for interpolation using preconditioning. *Information Processing Letters*, vol. 1, n4, 1972, pp. 157–163.

[12] Mersereau (Russell M.). – An algorithm for performing an inverse chirp z-transform. *IEEE Trans. Acoust. Speech Signal Processing*, vol. ASSP-22, n5, 1974, pp. 387–388.

[13] Moenck (R. T.) and Borodin (A.). – Fast modular transforms via division. In *Thirteenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 90–96. – 1972.

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

[15] 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. – 1954.

[16] Pan (V. Ya.). – Methods of computing values of polynomials. *Russian Mathematical Surveys*, vol. 21, n1, 1966, pp. 105–136.

[17] 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.

[18] Shoup (Victor) and Smolensky (Roman). – Lower bounds for polynomial evaluation and interpolation problems. *Computational Complexity*, vol. 6, n4, 1996/97, pp. 301–311.

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

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

[21] Svoboda (A.) and Valach (M.). – Operátorové obvody (Operational circuits). *Stroje na Zpracování Informací (Information Processing Machines)*, vol. 3, 1955, pp. 247–295.

[22] Takahasi (H.) and Ishibashi (Y.). – A new method for exact calculation by a digital computer. *Information Processing in Japan*, 1961, pp. 28–42.

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

CHAPITRE 7

Pgcd et résultant

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. Le pgcd et le résultant peuvent être calculés en complexité quasi-optimale.

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 n'en 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 des 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 chapitre se termine en Section 3 sur un algorithme de complexité quasi-optimale pour le pgcd dans $\mathbb{K}[X]$, exploitant la multiplication polynomiale rapide.

Dans ce chapitre, $\mathbb{A}$ 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 chapitre $\mathrm{pgcd}(A, B)$ 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 $a \in \mathbb{A}$, $b \in \mathbb{A} \setminus \{0\}$, 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 est appelé *reste* de la division euclidienne de a par b.

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

$$A = a m_1^{k_1} \cdots m_s^{k_s}, \qquad B = b m_1^{\ell_1} \cdots m_s^{\ell_s},$$

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}.

Figure 1: L'algorithme d'Euclide.

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 en Figure 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.

Théorème 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 Chapitre 4 (p. 141), le calcul naïf 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 est supérieur à celui de A, la première étape ne coûte pas d'opération arithmétique, et la borne ci-dessus s'applique au reste du calcul. □

Exercice 2. Énoncer et prouver l'analogue entier du résultat du Théorème 1.

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 que 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 tailles. Dans de nombreuses applications, c'est davantage de ces *cofacteurs* U et V que du pgcd lui-même qu'on a besoin. On parle alors de calcul de pgcd *étendu*. Ce calcul intervient par exemple de manière importante dans la manipulation des nombres algébriques (Exemple 3 ci-dessous), dans le calcul dans des algèbres quotient $\mathbb{K}[X]/(P)$ (Exemples 2 et 4 ci-dessous, cf. aussi la Section 1.4 en page 142), et dans le développement rapide des séries algébriques (Chapitre 8). Il intervient également dans d'autres algorithmes qui ne seront pas abordés dans ce cours, par exemple pour le calcul des « restes chinois rapides », pour la factorisation de polynômes dans $\mathbb{Z}[X]$ ou $\mathbb{Q}[X]$ par des techniques de type « Hensel », ou pour l'intégration symbolique (à la Hermite).

Mais, avant tout, les calculs de pgcd étendu permettent d'effectuer des *inversions modulaires*. Lorsque A et B sont premiers entre eux, (c'est-à-dire si G est un élément inversible de A), alors l'élément V est un inverse de B modulo A.

Exemple 2. La relation de Bézout pour $a + bX$ et $1 + X^2$ dans $\mathbb{R}[X]$ 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, alors $\mathbb{A}/(A)$ 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
$$\frac{13X - 20}{169} U - A = 1,$$
- 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.

AEE(A, B)

Entrée : A et B dans $\mathbb{A}$.
Sortie : Un pgcd G 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 :
   $(Q_i, R_{i+1}) :=$ QuotRem(R_{i-1}, R_i) ; # division euclidienne
   $U_{i+1} := U_{i-1} - Q_i U_i$; $V_{i+1} := V_{i-1} - Q_i V_i$;
   $i := i + 1$
3. Renvoyer $R_{i-1}, U_{i-1}, V_{i-1}$.

Figure 2: L'algorithme d'Euclide étendu.

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

$$33 \bmod 7 = 5, \qquad 3 \times 5 - 2 \times 7 = 1, \qquad 3 \times 25 \bmod 7 = 5.$$

Exercice 3. Soit p un nombre premier et soit a un entier non multiple de p. Donner un algorithme pour le calcul de l'inverse de a modulo p utilisant le petit théorème de Fermat ($a^{p-1} = 1 \bmod p$) et l'exponentiation binaire. Analyser la complexité binaire de cet algorithme et comparer avec l'approche par 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

$$R_{i-1} = Q_i R_i + R_{i+1}$$

donne la relation

$$R_{i+1} = R_{i-1} - Q_i R_i = (U_{i-1} - Q_i U_i)A + (V_{i-1} - Q_i V_i)B,$$

qui pousse à définir U_{i+1} par $U_{i-1} - Q_i U_i$ et V_{i+1} par $V_{i-1} - Q_i V_i$. À partir de cette relation, 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

$$A = a_m X^m + \cdots + a_0, \quad (a_m \neq 0), \qquad \text{et} \qquad B = b_n X^n + \cdots + b_0, \quad (b_n \neq 0),$$

appartiennent à $\mathbb{A}[X]$, cette matrice, carrée de taille $m+n$, s'écrit

$$\mathsf{Syl}(A,B) = \begin{pmatrix} a_m & a_{m-1} & \dots & a_0 & & & \\ & a_m & a_{m-1} & \dots & a_0 & & \\ & & \ddots & \ddots & & \ddots & \\ & & & a_m & a_{m-1} & \dots & a_0 \\ b_n & b_{n-1} & \dots & b_0 & & & \\ & b_n & b_{n-1} & \dots & b_0 & & \\ & & \ddots & \ddots & & \ddots & \\ & & & b_n & b_{n-1} & \dots & b_0 \end{pmatrix}$$

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 $\mathsf{Syl}(A,B)$ (cf. Définition 3, page 115).

Proposition 1. *Soient A, B deux polynômes de $\mathbb{K}[X]$. 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 $X - 3$.

Définition 1. 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 4. Calculer le discriminant de $aX^2 + bX + c$ en prenant le déterminant de la matrice de Sylvester.

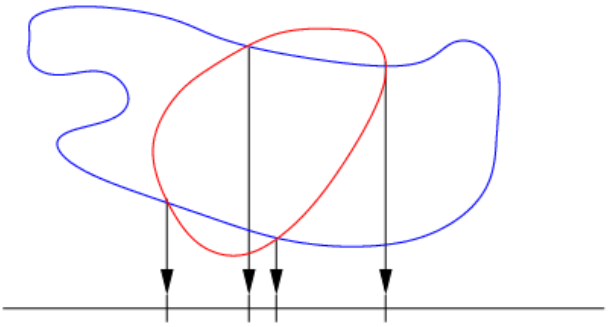


Figure 3: Le résultant calcule des projections.

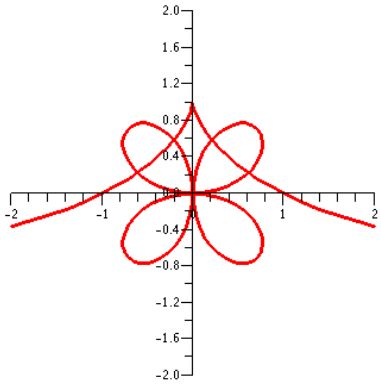


Figure 4: Les deux courbes de l'exemple 8.

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 plus générale pour un nombre arbitraire d'équations et d'inconnues, la *résolution géométrique*, qui dépasse le cadre de ce cours.

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 4, 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.

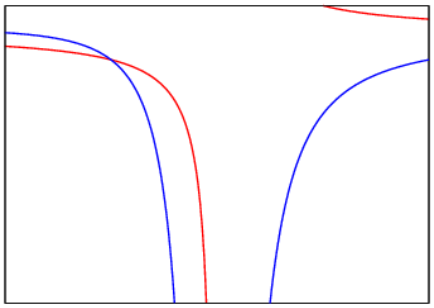


FIGURE 5: Les courbes de l'exemple 9.

Le résultat général est le suivant.

Proposition 2. *Soient $A = a_m Y^m + \cdots$ 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 des termes de tête a_m et b_n.*

Démonstration. Ce résultat est une conséquence immédiate du théorème 2 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.

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 fournit bien 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 des numérateurs de $X - A(T)$ et de $Y - B(T)$.

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 et calcul quadratique.** L'essentiel des propriétés du résultant est contenu dans le théorème suivant.

Théorème 2. *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

$$\begin{aligned}\mathrm{Res}(A,B) &= a^n b^m \prod_{i,j}(\alpha_i - \beta_j) = (-1)^{mn} b^m \prod_{1\le j\le n} A(\beta_j) \\ &= a^n \prod_{1\le i\le m} B(\alpha_i) = (-1)^{mn}\,\mathrm{Res}(B,A).\end{aligned}$$

Démonstration. Il suffit de prouver la première égalité. Les trois suivantes en sont des conséquences immédiates.

Le facteur $a^n b^m$ est une conséquence de la multilinéarité du déterminant. On peut donc se restreindre au 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 $\mathbb{A}$ étant intègre, il possède un corps de fractions 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 3. *Il existe U et V dans $\mathbb{A}[X]$ tels que le résultant de A et de B s'écrive* $\mathrm{Res}(A,B) = UA + VB$.

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 le résultant $\mathrm{Res}(A,B)$ (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 $\mathbb{K}[X]$. La proposition 3 montre que cette combinaison se fait « sans division », c'est-à-dire avec des coefficients dans $\mathbb{A}[X]$.

En utilisant la définition 1 et les résultats du Chapitre 3, le résultant de deux polynômes A, B de $\mathbb{K}[X]$ peut se calculer naïvement en $O(\mathsf{MM}(n)) = O(n^\theta)$ opérations dans $\mathbb{K}$, où $n = \max(\deg(A),\deg(B))$ et où $2 \le \theta \le 3$ est un exposant faisable pour la multiplication matricielle.

Un algorithme plus efficace, de complexité seulement quadratique, s'obtient comme conséquence du Théorème 2 et est laissé en exercice.

Exercice 5 (Algorithme de type Euclide pour le résultant)**.** Soient A et B deux polynômes de $\mathbb{K}[X]$ de degrés $m = \deg(A)$ et $n = \deg(B)$.

1. Soit $A = QB + R$ la division euclidienne dans $\mathbb{K}[X]$ de A par B. Si $r = \deg(R)$, et si b_n est le coefficient dominant de B, montrer que
$$\mathrm{Res}(A, B) = (-1)^{mn} \cdot b_n^{m-r} \cdot \mathrm{Res}(B, R).$$
2. Écrire un algorithme de type Euclide (*i.e.* à base de divisions répétées), qui calcule le résultant $\mathrm{Res}(A, B)$ en $O(mn)$ opérations dans $\mathbb{K}$.

2.4. **Calcul avec des nombres algébriques.** L'idée que les polynômes sont de 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 4. *Soient $A = \prod_i (X - \alpha_i)$ et $B = \prod_j (X - \beta_j)$ des polynômes de $\mathbb{K}[X]$. Alors*
$$\mathrm{Res}_X(A(X), B(T - X)) = \prod_{i,j} (T - (\alpha_i + \beta_j)),$$
$$\mathrm{Res}_X(A(X), B(T + X)) = \prod_{i,j} (T - (\beta_j - \alpha_i)),$$
$$\mathrm{Res}_X(A(X), X^{\deg B} B(T/X)) = \prod_{i,j} (T - \alpha_i \beta_j),$$
$$\mathrm{Res}_X(A(X), T - B(X)) = \prod_{i} (T - B(\alpha_i)).$$

Démonstration. C'est une application directe du Théorème 2. □

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 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'algorithmes rapides (de complexité quasi-optimale) 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 (voir l'exercice 9). Cet algorithme n'est pas quasi-optimal. En revanche, pour les trois premières opérations de la Proposition 4, des algorithmes quasi-optimaux à base de multiplication rapide de séries existent, ils ont été présentés au Chapitre 4 (Section 3.6, page 134).

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 lesquels 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*.

Définition 2. 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

$$b^{\deg A - \deg B + 1} A = \overline{Q}B + \overline{R},$$

où $\overline{Q}, \overline{R} \in \mathbb{A}[X]$ avec $\deg \overline{R} < \deg \overline{Q}$.

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 13 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 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 mêmes 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-résultants 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.

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}.

FIGURE 6: L'algorithme des sous-résultants.

Proposition 5. *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.

Calcul de la paramétrisation. Les sous-résultants permettent de finir la « résolution » (commencée en page 164) 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.

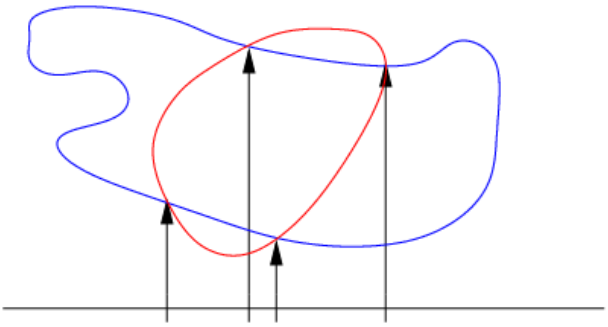


Figure 7: Paramétrisation des solutions d'un système algébrique par les racines d'un résultant.

3. Algorithme d'Euclide rapide

Le point central de cette section est la présentation d'un algorithme rapide pour le pgcd, qui est un algorithme récursif à base de multiplications et de divisions euclidiennes rapides. 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. Nous décrivons maintenant l'algorithme dit du « 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 $R_N = \operatorname{pgcd}(A, B)$ le dernier non nul d'entre eux. D'après les égalités en page 162, 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.

Notons qu'une étape de l'algorithme d'Euclide classique peut elle aussi s'écrire sous une forme matricielle. En effet, si Q_i est le quotient de la division de R_{i-1} par R_i, on peut écrire :

$$\begin{bmatrix} 0 & 1 \\ 1 & -Q_i \end{bmatrix} \begin{bmatrix} R_{i-1} \\ R_i \end{bmatrix} = \begin{bmatrix} R_i \\ R_{i+1} \end{bmatrix}.$$

Ainsi, la matrice $M_{A,B}$ est un produit de matrices élémentaires de ce type. Elle est inversible.

3.1. 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 :

- $\deg R_j \geq \frac{n}{2}$;
- $\deg R_{j+1} < \frac{n}{2}$.

On a vu en page 162 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 voulues 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).

Euclide via HGCD

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 $M_{R_{j+1},R_{j+2}}MM_{\text{hgcd}}$.

FIGURE 8: Algorithme d'Euclide rapide via le demi-pgcd rapide.

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 8.

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 une 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 en Section 3.2.

Proposition 6. *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 qui font la transition 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 pour l'étape 4 une division euclidienne rapide (cf. Théorème 1 en page 1). 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 à la récurrence :

$$\mathsf{B}(n) \leq \mathsf{B}(n/2) + \mathsf{H}(n) + C\mathsf{M}(n),$$

C étant une constante. Le théorème « 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 uniquement à ce dernier.

3.2. **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 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

$$X^4 + 2X^3 + 5X^2 + 3X + 11.$$

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. Soient $M = \mathsf{HGCD}(f, g)$, 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 $C' = A' \bmod B'$, 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. Soient $\ell = 2m - \deg B'$, b et c les quotients respectifs de B' et C' par X^ℓ.
   ℓ est de l'ordre de $\frac{n}{4}$, de sorte de b et c on des degrés de l'ordre $\frac{n}{2}$.
7. Soit $M'' = \mathsf{HGCD}(b, c)$. Renvoyer la matrice $M''M'M$.

Figure 9: Algorithme du demi-pgcd rapide.

Voyons comment cette idée permet d'obtenir notre algorithme. Étant donnés 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 éléments de cette matrice ont degré $\frac{n}{4}$). La remarque ci-dessus permet de montrer qu'on obtient ainsi une matrice de transition vers les restes de A et B eux-mêmes.

On applique donc cette matrice aux polynômes initiaux ; on obtient des restes B' et C' 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 9 permettent d'assurer la correction de l'algorithme.

Proposition 7. *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
$$\mathsf{H}(n) \leq 2\mathsf{H}(n/2) + C\mathsf{M}(n),$$
où C est une constante. La conclusion découle comme d'habitude du lemme « diviser pour régner ». □

3.3. **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.

Théorème 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 $X^{n-2\ell}$. □

3.4. **Le cas des 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 4. *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.

Exercices

Exercice 6 (Factorisation sans carré)**.** Soit $\mathbb{K}$ un corps de caractéristique nulle. Si $F \in \mathbb{K}[X]$ se factorise en irréductibles comme $\prod_i f_i^{\alpha_i}$, le polynôme $\overline{F} = \prod_i f_i$ est appelé la *partie sans carré de* F. Si $n = \deg(F)$, montrer qu'on peut calculer les coefficients de la partie sans carré de F à partir des coefficients de F en $\mathcal{O}(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$.

Exercice 7. Soient $f, g \in \mathbb{K}[X]$ des polynômes unitaires.

1. Soit $N \in \mathbb{N} \setminus \{0\}$. Montrer que l'unique polynôme unitaire de $\mathbb{K}[X]$ dont les racines sont les puissances N-ièmes des racines de f peut être obtenu par un calcul de résultant.
2. Si f est le polynôme minimal d'un nombre algébrique α, montrer qu'on peut déterminer un polynôme annulateur de $g(\alpha)$ à l'aide d'un résultant.
3. Calculer le polynôme minimal sur $\mathbb{Q}$ de $\alpha = \sqrt[3]{4} + \sqrt[3]{2} + 1$.

Exercice 8. Soit $f \in \mathbb{K}[X]$ un polynôme unitaire de degré $d \geq 1$. Pour $N \geq 1$, on note $G_N(f)$ l'unique polynôme unitaire de degré d dont les racines sont les puissances N-ièmes des racines de f.

(a) Exprimer $G_N(f)$ à l'aide d'un résultant de polynômes à deux variables.

(b) Justifier l'appartenance à $\mathbb{K}[X]$ du polynôme $G_N(f)$.

(c) Utiliser (a) pour donner un algorithme pour le calcul de $G_N(f)$; estimer sa complexité.

(d) Montrer que $G_2(f)$ peut se calculer en $\mathcal{O}(\mathsf{M}(d))$ opérations dans $\mathbb{K}$.

(e) Si N est une puissance entière de 2, montrer qu'on peut calculer $G_N(f)$ en $\mathcal{O}(\mathsf{M}(d)\log(N))$ opérations dans $\mathbb{K}$.

Exercice 9 (Résultants bivariés)**.** Soient $F, G \in \mathbb{K}[X, Y]$ ayant des degrés en X et en Y bornés par un entier $D \geq 1$. Montrer que le polynôme $R(Y) := \mathrm{Res}_X(F, G)$ est de degré au plus $2D^2$. Donner un algorithme de type évaluation-interpolation pour le calcul des coefficients de R et estimer son coût en opérations dans $\mathbb{K}$.

Notes

L'introduction du résultant et des sous-résultants remonte à Bézout et Sylvester [2, 32, 33]. Ces notions ont été revisitées en calcul formel dans les années 1965 par Collins [7, 8, 9], Brown et Traub [4, 5]. L'article de synthèse [35] retrace leur histoire. De nombreuses propriétés des résultants et de l'algorithme d'Euclide sont établies de manière élémentaire à l'aide des fonctions symétriques dans [18]. On peut consulter les livres [34, 10, 16], ou [17] pour une approche un peu plus complète.

L'idée de base l'algorithme de demi-pgcd (le quotient de la division euclidienne de deux entiers ne dépend que de leurs chiffres de poids fort) est due à Lehmer [19]. Knuth [15] en a fait le premier algorithme sous-quadratique, qui calcule le pgcd de deux entiers de taille binaire n en

complexité binaire $O(\mathsf{M}_{\mathbb{Z}}(n)\log^4(n))$. Schönhage [26] a montré comment abaisser cette complexité à $O(\mathsf{M}_{\mathbb{Z}}(n)\log(n))$. L'algorithme de [26] a été adapté au cas polynomial par Moenck [23], mais seulement pour le cas où la suite des restes est normale. Brent, Gustavson et Yun [12, 3] ont donné un algorithme de type LKS (Lehmer-Knuth-Schönhage) pour le cas polynomial général, de complexité arithmétique $O(\mathsf{M}(n)\log(n))$ en degré n. Cet algorithme utilise $O(n\log n)$ opérations non scalaires. Strassen [30, 31] a montré que $\Omega(n\log n)$ opérations non scalaires sont aussi nécessaires, du moins pour une entrée générique, donc l'algorithme LKS est optimal de ce point de vue.

Schwartz [28] a montré que le résultant de deux polynômes peut se calculer également en $O(\mathsf{M}(n)\log(n))$ opérations arithmétiques. Dans [27], Schönhage a donné un algorithme (probabiliste) pour le calcul rapide du pgcd de deux polynômes de $\mathbb{Z}[X]$. Pour deux polynômes de degré borné par n et coefficients d'au plus ℓ chiffres, l'algorithme a une complexité binaire $\tilde{O}(n(n+\ell))$, il est donc quasi-optimal (en la taille binaire de l'entrée) lorsque $\ell > n$.

L'exercice 6 est inspiré par [37], qui montre qu'il est possible de déterminer toute la factorisation sans carré en la même complexité. L'article [38] prouve que les problèmes du calcul du pgcd et de la factorisation sans carré sont équivalents du points de vue de la complexité.

Il est très difficile de trouver une référence complète, lisible et sans erreur sur les algorithmes de pgcd rapide. Le chapitre 11 de [34] fournit une bonne partie des résultats techniques implicitement utilisés dans la section 3, mais l'algorithme qu'il propose est erroné. Les notes du livre [36] nous ont beaucoup inspirés, mais elles contiennent des erreurs pour le pgcd entier. On pourra également consulter des articles plus récents [25, 29, 24].

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).

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. Le résultant différentiel de deux équations différentielles a été défini par Berkovich et Tsirulik dans [1]. Chardin [6] a défini les sous-résultants de deux équations différentielles. Li [21, 22] a étendu la définition aux polynômes de Ore. On trouve dans [11, 20] des algorithmes efficaces pour le calcul du pgcd de deux opérateurs différentiels ; ces algorithmes ont été étendus aux sous-résultants dans [21, 22]. Dans [6], Chardin a donné une expression du résultant de deux équations différentielles en termes de leurs solutions. Cela a été généralisé au cadre des polynômes de Ore par Hong [13, 14].

Bibliographie

[1] Berkovich (L. M.) and Tsirulik (V. G.). – Differential resultants and some of their applications. *Differentsial′nye Uravneniya*, vol. 22, 1986, pp. 750–757, 914. – English translation in : Differential Equations, Plenum Publ. Corp., Vol. 22, no. 5, pp. 530–536. NY Plenum 1986.

[2] Bézout (É.). – Recherches sur le degré des équations résultantes de l'évanouissement des inconnues. *Histoire de l'académie royale des science*, 1764, pp. 288–338.

[3] Brent (Richard P.), Gustavson (Fred G.), and Yun (David Y. Y.). – Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, vol. 1, 1980, pp. 259–295.

[4] Brown (W. S.). – On Euclid's algorithm and the computation of polynomial greatest common divisors. In *SYMSAC '71 : Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*. pp. 195–211. – ACM, New York, NY, USA, 1971.

[5] Brown (W. S.) and Traub (J. F.). – On Euclid's algorithm and the theory of subresultants. *J. Assoc. Comput. Mach.*, vol. 18, 1971, pp. 505–514.

[6] Chardin (Marc). – Differential resultants and subresultants. In *Fundamentals of computation theory (Gosen, 1991)*, pp. 180–189. – Springer, Berlin, 1991.

[7] Collins (G. E.). – Polynomial remainder sequences and determinants. *Amer. Math. Monthly*, vol. 73, 1966, pp. 708–712.

[8] Collins (George E.). – Subresultants and reduced polynomial remainder sequences. *J. Assoc. Comput. Mach.*, vol. 14, 1967, pp. 128–142.

[9] Collins (George E.). – The calculation of multivariate polynomial resultants. *J. Assoc. Comput. Mach.*, vol. 18, 1971, pp. 515–532.

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

[11] Grigoriev (D. Yu.). – Complexity of factoring and calculating the GCD of linear ordinary differential operators. *J. Symbolic Comput.*, vol. 10, n1, 1990, pp. 7–37.

[12] Gustavson (Fred G.) and Yun (David Y. Y.). – Fast algorithms for rational Hermite approximation and solution of Toeplitz systems. *IEEE Trans. Circuits and Systems*, vol. 26, n9, 1979, pp. 750–755.

[13] Hong (Hoon). – Ore principal subresultant coefficients in solutions. *Appl. Algebra Engrg. Comm. Comput.*, vol. 11, n3, 2001, pp. 227–237.

[14] Hong (Hoon). – Ore subresultant coefficients in solutions. *Appl. Algebra Engrg. Comm. Comput.*, vol. 12, n5, 2001, pp. 421–428.

[15] Knuth (Donald E.). – The analysis of algorithms. In *Actes du Congrès International des Mathématiciens (Nice, 1970), Tome 3*, pp. 269–274. – Gauthier-Villars, Paris, 1971.

[16] 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.

[17] Lang (Serge). – *Algebra*. – Springer-Verlag, New York, 2002, third edition, *Graduate Texts in Mathematics*, vol. 211, xvi+914p.

[18] 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.

[19] Lehmer (D. H.). – Euclid's Algorithm for Large Numbers. *Amer. Math. Monthly*, vol. 45, n4, 1938, pp. 227–233.

[20] Li (Z.) and Nemes (I.). – A modular algorithm for computing greatest common right divisors of Ore polynomials. In *ISSAC'97*. pp. 282–289. – ACM, New York, 1997.

[21] Li (Ziming). – A subresultant theory for Ore polynomials with applications. In *ISSAC'98*. pp. 132–139. – ACM, New York, 1998.

[22] Li (Ziming). – Greatest common right divisors, least common left multiples, and subresultants of Ore polynomials. In *Mathematics mechanization and applications*, pp. 297–324. – Academic Press, San Diego, CA, 2000.

[23] Moenck (R. T.). – Fast computation of GCDs. In *Fifth Annual ACM Symposium on Theory of Computing (Austin, Tex., 1973)*, pp. 142–151. – Assoc. Comput. Mach., New York, 1973.

[24] Möller (Niels). – On Schönhage's algorithm and subquadratic integer GCD computation. *Math. Comp.*, vol. 77, n261, 2008, pp. 589–607.

[25] Pan (V. Y.) and Wang (X.). – Acceleration of Euclidean algorithm and extensions. In Mora (Teo) (editor), *ISSAC'02*. pp. 207–213. – ACM, New York, 2002.

[26] Schönhage (A.). – Schnelle Berechnung von Kettenbruchentwicklugen. *Acta Inform.*, vol. 1, 1971, pp. 139–144.

[27] Schönhage (A.). – Probabilistic computation of integer polynomial GCDs. *J. Algorithms*, vol. 9, n3, 1988, pp. 365–371.

[28] Schwartz (J. T.). – Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, vol. 27, n4, 1980, pp. 701–717.

[29] 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.

[30] Strassen (V.). – The computational complexity of continued fractions. In *SYMSAC'81*. pp. 51–67. – ACM, New York, NY, USA, 1981.

[31] Strassen (V.). – The computational complexity of continued fractions. *SIAM J. Comput.*, vol. 12, n1, 1983, pp. 1–27.

[32] Sylvester (James Joseph). – On rational derivation from equations of coexistence, that is to say, a new and extended theory of elimination, *1839–40*. In *The Collected mathematical papers of James Joseph Sylvester*, pp. 40–53. – Baker, H. F., New York, 1839.

[33] Sylvester (James Joseph). – A method of determining by mere inspection the derivatives from two equations of any degree. *Philos. Mag.*, vol. 16, n101, 1840, pp. 132–135.

[34] von zur Gathen (Joachim) and Gerhard (Jürgen). – *Modern computer algebra*. – Cambridge University Press, New York, 2003, 2nd edition, xiv+785p.

[35] von zur Gathen (Joachim) and Lücking (Thomas). – Subresultants revisited. *Theoret. Comput. Sci.*, vol. 297, n1-3, 2003, pp. 199–239.

[36] Yap (Chee). – *Fundamental Problems in Algorithmic Algebra*. – Oxford Univ. Press, 2000.

[37] Yun (David Y.Y.). – On square-free decomposition algorithms. In *SYMSAC'76*. pp. 26–35. – ACM, New York, NY, USA, 1976.

[38] Yun (David Y.Y.). – On the equivalence of polynomial GCD and squarefree factorization problems. In *Proc. of the MACSYMA Users' Conference*, pp. 65–70. – 1977.

CHAPITRE 8

Algorithmique des séries D-finies

Résumé

Les séries D-finies se manipulent plus rapidement que les séries arbitraires. L'équation différentielle linéaire les définissant fournit une structure de données adaptée sur laquelle plusieurs opérations utiles sont algorithmiques. En particulier, la multiplication de deux séries D-finies s'effectue en temps linéaire.

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

1.1. **Définitions.** Si $\mathbb{K}$ désigne un corps, l'anneau des séries formelles à coefficients dans $\mathbb{K}$ est noté $\mathbb{K}[[X]]$. Ses principales propriétés ont été présentées au Chapitre 4. Son corps des fractions, noté $\mathbb{K}((X))$, est égal à $\mathbb{K}[[X]][1/X]$. Ses éléments sont appelés des séries de Laurent formelles. C'est une algèbre non seulement sur $\mathbb{K}$, mais aussi sur le corps des fractions rationnelles $\mathbb{K}(X)$.

Définition 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 m suffisamment grand, $A, A', \ldots, A^{(m)}$ sont liées, et une relation de liaison entre ces dérivées est une équation différentielle linéaire.

Définition 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 à 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 $\mathbb{K} = \mathbb{Q}$.

1.2. **Méthode naïve.** Le résultat qu'il s'agit de considérer d'un point de vue algorithmique est le suivant. Il s'agit d'un analogue du Lemme 2 au Chapitre 5.

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

$$q_0(X)A^{(m)}(X) + \cdots + q_m(X)A(X) = 0 \tag{2}$$

une équation différentielle qui l'annule. En notant $[X^n]f(X)$ le coefficient de X^n dans la série $f(X)$ avec la convention que ce coefficient est nul pour $n < 0$, on a pour $n \geq 0$

$$[X^n]f'(X) = (n+1)[X^{n+1}]f(X), \qquad [X^n]X^k f(X) = [X^{n-k}]f(X). \tag{3}$$

Par conséquent, l'extraction du coefficient de X^n de (2) fournit une récurrence linéaire sur les a_n valide pour tout $n \geq 0$ avec la convention $a_k = 0$ pour $k < 0$. Pour obtenir une récurrence de la forme (1) il faut décaler les indices de $n_0 := \max_{0\leq i\leq m}(\deg q_i + i - m)$ s'il est strictement positif. Les équations obtenues alors pour les indices moindres fournissent des contraintes linéaires sur les premiers coefficients a_n pour qu'ils correspondent aux coefficients d'une série solution de (2).

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 $(\frac{d}{dX})^k$ signifie que l'opérateur $\frac{d}{dX}$ 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. □

Exemple 1. L'équation $A' - X^k A = 0$ ($k \in \mathbb{N}$) donne la récurrence linéaire $(n+1)a_{n+1} - a_{n-k} = 0$ valide pour tout $n \geq 0$, avec la convention que les a_n d'indice négatif sont nuls. On en déduit que a_0 est libre, puis les contraintes $a_1 = \cdots = a_k = 0$, et les coefficients suivants sont fournis par la récurrence décalée $(n+k+1)a_{n+k+1} - a_n = 0$, valide pour $n \geq 0$. On reconnaît ainsi les coefficients de $a_0 \exp(X^{k+1}/(k+1))$.

Exercice 1. Retrouver la récurrence satisfaite par la suite $1/n!$ à partir de l'équation $A' = A$, et réciproquement.

Exercice 2. Calculer une équation différentielle linéaire satisfaite par la série génératrice des nombres de Fibonacci.

Exercice 3. 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 §1.4.

1.3. **Exemples d'applications.**

Exemple 2. 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 la méthode de Newton, par exemple pour résoudre le polynôme. Cette idée peut aussi être combinée avec la méthode de scindage binaire du Chapitre 13.

Exemple 3. Le cas particulier des récurrences d'ordre 1 donne lieu aux suites *hypergéométriques*, qui jouent un rôle important dans les algorithmes de sommation symbolique.

1.4. **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 qui exploitent la multiplication rapide.

1.4.1. *Cas d'un opérateur donné en $\theta = \frac{d}{dX}$.* 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 $\theta = \frac{d}{dX}$ (θ est parfois appelé *l'opérateur d'Euler*), alors la conversion en récurrence est assez facile : partant de

$$\sum_{\substack{0\le j\le m\\ 0\le i\le d}} a_{i,j} X^j \theta^i,$$

les relations (3) donnent l'opérateur de récurrence

$$\sum a_{i,j} S_n^{-j} n^i = \sum a_{i,j} (n-j)^i S_n^{-j}.$$

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 $O(m\,\mathsf{M}(d))$ 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 $O(\mathsf{M}(d))$ opérations, par exemple en utilisant la somme composée (Chapitre 4).

1.4.2. *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

$$\sum_{j=0}^{m+d} X^j \sum_{i=0}^{d} a_{i,j} X^i \partial^i$$

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 $O(\mathsf{M}(d))$ opérations à partir des coefficients a_i, et donc les coefficients de P en $O(\mathsf{M}(d)\log d)$ opérations, par interpolation rapide, en utilisant les techniques du Chapitre 6.

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*

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

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 Chapitre 13.

2. Propriétés de clôture

Les séries D-finies jouissent de propriétés de clôture. En ce sens, elles sont un analogue différentiel des nombres algébriques. Ces propriétés de clôture généralisent celles sur les suites récurrentes linéaires à coefficients *constants* (srlcc) évoquées en Section 3.3 du Chapitre 5.

2.1. **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 $\mathbb{K}$.*

Une conséquence algorithmique fondamentale de ce théorème est le fait que les séries D-finies peuvent être multipliées plus rapidement que les séries arbitraires, en complexité *linéaire*.

Corollaire 1. *Soient $A, B \in \mathbb{K}[[X]]$ deux séries D-finies, solutions d'équations différentielles d'ordre $O(1)$ à coefficients polynomiaux de degré $O(1)$. On peut calculer le produit $AB \bmod X^N$ en $O(N)$ opérations dans $\mathbb{K}$.*

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 $C = AB$ de deux séries D-finies A et B. Par la formule de Leibniz, toutes les dérivées de C s'écrivent comme combinaisons linéaires de produits entre une dérivée $A^{(i)}$ de A et une dérivée $B^{(j)}$ de B. Les dérivées de A et de B étant engendrées par un nombre fini d'entre elles, il en va de même pour les produits $A^{(i)}B^{(j)}$, ce qui prouve la D-finitude de C. □

Exercice 4. 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. Le calcul se ramène donc ultimement à la résolution d'un système linéaire à coefficients dans $\mathbb{K}[X]$, ce qui peut se faire efficacement grâce à l'algorithme de Storjohann (Chapitre 14).

Exemple 4. 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 5. De la même manière, montrer que $\sin^2 X + \cos^2 X = 1$, avec et sans calcul.

2.2. **Produit d'Hadamard.**

Corollaire 2. *Si $A = \sum_{n\geq 0} a_n X^n$ et $B = \sum_{n\geq 0} b_n X^n$ sont deux séries D-finies, alors leur produit d'Hadamard*

$$A \odot B = \sum_{n\geq 0} a_n b_n X^n$$

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 5. Les polynômes orthogonaux 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 :

$$\sum_{n\geq 0} H_n(X)H_n(Y)\frac{Z^n}{n!} = \frac{\exp\left(\frac{4Z(XY - Z(X^2+Y^2))}{1-4Z^2}\right)}{\sqrt{1-4Z^2}}.$$

Exemple 6. Si l'intégrale

$$I(t) = \int_{-\infty}^{\infty} e^{-xt^2} f(x)\, dx,$$

peut être calculée à partir du développement de Taylor de f en intervertissant sommation et intégration, alors $I(t)$ est D-finie si f l'est. Il en va de même pour la transformée de Laplace ou la transformée de Fourier.

Exercice 6. Montrer que dans les mêmes conditions, f est D-finie si I l'est, et donner un algorithme pour calculer une équation différentielle linéaire satisfaite par f à partir d'une équation satisfaite par I.

3. 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 le Chapitre 7), 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 7. Les dénombrements d'arbres mènent naturellement à des équations algébriques sur les séries génératrices. Ainsi, la série génératrice $C(X)$ des nombres de Catalan (nombre d'arbres binaires à n sommets internes) vérifie

$$C = 1 + XC^2;$$

la série génératrice $M(X)$ des nombres de Motzkin (nombre d'arbres unaires-binaires à n sommets internes) vérifie

$$M = 1 + XM + XM^2.$$

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 7. Trouver une formule explicite des nombres de Catalan. Récrire les fonctions Γ qui pourraient apparaître dans le résultat en termes de factorielles, puis de coefficient binomial.

Exercice 8. À l'aide d'un système de calcul formel, calculer une récurrence linéaire satisfaite par les coefficients de la série $Y(X)$ solution de

$$Y = 1 + XY + XY^7.$$

Les mêmes arguments que ci-dessus mènent à une autre propriété de clôture des séries D-finies.

Corollaire 3. *Si F est une série D-finie et A une série algébrique sans terme constant, alors $F \circ A$ est D-finie.*

La preuve consiste à observer que les dérivées successives de $F \circ A$ s'expriment comme combinaisons linéaires des $F^{(i)}(A)A^j$ pour un nombre fini de dérivées de F (par D-finitude) et de puissances de A (par la même preuve que pour le théorème 4). Cette preuve fournit encore un algorithme.

Exemple 8. À l'aide d'un système de calcul formel, calculer une récurrence linéaire satisfaite par les coefficients du développement en série de Taylor de

$$\exp\left(\frac{1-\sqrt{1-4X}}{2}\right).$$

Corollaire 4. *Si A est une série algébrique, alors* $\exp \int A$ *est D-finie.*

Exercice 9. Faire la preuve.

4. 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. *Soient A et F deux séries de $\mathbb{K}[[X]]$.*

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 A algébrique de genre supérieur ou égal à 1, alors F et $A \circ f$ sont D-finies si et seulement si F est algébrique.*

Exercice 10. Prouver le sens « si » de ces trois propriétés.

Exercices

Exercice 11. Parmi les suites données par les termes généraux suivants, indiquer celles qui sont P-récursives et celles qui ne le sont pas :

$$\frac{1}{\binom{2n}{n}}, \qquad \cos n, \qquad 2^{2^n}, \qquad H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}, \qquad \lfloor \log n \rfloor.$$

Parmi les fonctions dont la valeur en x est donnée par les termes suivants, indiquer celles qui sont D-finies et celles qui ne le sont pas :

$$\exp\Big(\sqrt[3]{1-x}\Big), \qquad \prod_{i\geq 1} \frac{1}{1-x^i}, \qquad \sum_{i\geq 0} x^{2^i}, \qquad \sum_{n\geq 0} F_n^3 \frac{x^n}{n!}, \qquad \tan(x).$$

(F_n désigne le n-ième nombre de Fibonacci, dont la suite vérifie $F_{n+2} = F_{n+1} + F_n$ et $F_1 = F_0 = 1$.)

Exercice 12.

1. Soit $(a_n)_{n\geq 0}$ une suite P-récursive d'éléments d'un corps de caractéristique 0. Montrer que les suites extraites $(a_{2n})_{n\geq 0}$ et $(a_{2n+1})_{n\geq 0}$ sont également P-récursives.
2. Soit $(g_n)_{n\geq 0}$ une suite dont la série génératrice exponentielle est

$$\sum_{n\geq 0} g_n \frac{X^n}{n!} = \frac{\exp(-X/2 - X^2/4)}{\sqrt{1-X}}.$$

   Montrer que la suite $(g_n)_{n\geq 0}$ satisfait à une récurrence linéaire homogène, d'ordre au plus 3 et à coefficients polynomiaux de degrés au plus 2. La récurrence n'est pas demandée explicitement.
3. Si $N > 0$, quel est le coût binaire, en fonction de N, du calcul de g_N ?

Exercice 13 (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 $\big\{e^{\alpha_1 X}, Xe^{\alpha_1 X}, \ldots, X^{d_1-1}e^{\alpha_1 X}, \ldots,\ e^{\alpha_k X}, Xe^{\alpha_k X}, \ldots, X^{d_k-1}e^{\alpha_k X}\big\}$.

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 14 (Puissance symétrique d'équation différentielle)**.** Soient m et d deux entiers naturels et soient $a(X), b(X)$ deux polynômes dans $\mathbb{Q}[X]$ de degrés au plus d.

1. Montrer qu'il existe une équation différentielle $\mathcal{E}_m$ linéaire homogène d'ordre $m+1$, à coefficients dans $\mathbb{Q}[X]$, qui admet $\phi(X)^m$ comme solution, quelle que soit la solution $\phi(X)$ de l'équation différentielle

$$(\mathcal{E}_1) \qquad y''(X) + a(X)y'(X) + b(X)y(X) = 0.$$

   On admettra que pour toute base (ϕ_1, ϕ_2) de solutions de $\mathcal{E}_1$, les fonctions $\phi_1^i \phi_2^{m-i}$, $i = 0, \ldots, m$ sont linéairement indépendantes.
2. Montrer que si $a = 0$, alors $\mathcal{E}_2 : y'''(X) + 4b(X)y'(X) + 2b'(X)y(X) = 0$.

3. Expliciter un algorithme pour calculer $\mathcal{E}_m$, en ramenant le calcul final à un calcul sur des matrices de polynômes.
4. Estimer la complexité de cet algorithme en nombres d'opérations dans $\mathbb{Q}$ en fonction de m et d.
5. Pour m fixé, on associe à une fonction y de X des fonctions L_k par les valeurs initiales
$$L_0(X) = y(X), \quad L_1(X) = y'(X)$$
et la relation de récurrence
$$L_{k+1}(X) = L'_k(X) + ka(X)L_k(X) + k(m-k+1)b(X)L_{k-1}(X).$$
(a) Montrer que lorsque $y = \phi^m$, pour $0 \leq k \leq m$,
$$L_k(X) = m(m-1)\dots(m-k+1)\phi^{m-k}(X)\phi'(X)^k,$$
et $L_{m+1}(X) = 0$.
(b) Montrer que $L_{m+1}(X) = 0$ n'est autre que l'équation $\mathcal{E}_m$.
(c) En déduire des bornes sur les degrés des coefficients de $\mathcal{E}_m$ et un algorithme de complexité $O(m^3\mathsf{M}(d))$ pour le calcul de $\mathcal{E}_m$.
6. * Montrer que si $a, b \in \mathbb{Q}$, alors $\mathcal{E}_m$ est à coefficients constants et qu'on peut calculer $\mathcal{E}_m$ en $O(\mathsf{M}(m)\log m)$ opérations dans $\mathbb{Q}$.

Notes

Les propriétés de clôture des séries D-finies ont été décrites avec leurs applications par Stanley dans [13] ainsi que dans son livre [14], et par Lipshitz dans [9].

L'utilisation des séries D-finies pour les séries algébriques en combinatoire est exploitée à de nombreuses reprises par Comtet dans son livre [5], 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 [7]. Quelques années plus tard, il est encore retrouvé par Tannery [15] dans sa thèse, dont le manuscrit est remarquablement clair et disponible sur le web (à l'url `http://gallica.bnf.fr`). Pour le calcul du développement de séries algébriques, il faut tenir compte de l'ordre assez élevé des équations obtenues [4], ce qui rend cet algorithme utile pour des très grandes précisions, l'itération de Newton du Chapitre 4 étant préférable pour des précisions modérées.

Les limitations de la dernière section ne sont pas très connues. Elles sont dues à Harris et Sibuya pour la première [8] et à Singer pour les deux autres [11].

En ce qui concerne les algorithmes et les implantations, la plupart des algorithmes présentés dans ce chapitre sont implantés dans le package `gfun` de Maple [10]. L'algorithme rapide de la section 1.4 provient essentiellement de [3], qui donne une variante légèrement plus efficace (d'un facteur constant).

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] Bostan (Alin), Chyzak (Frédéric), Lecerf (Grégoire), Salvy (Bruno), and Schost (Éric). – Differential equations for algebraic functions. In *ISSAC'07*. pp. 25–32. – ACM Press, 2007.

[5] Comtet (L.). – *Analyse Combinatoire.* – PUF, Paris, 1970. 2 volumes.

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

[7] 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.

[8] Harris (William A.) and Sibuya (Yasutaka). – The reciprocals of solutions of linear ordinary differential equations. *Advances in Mathematics*, vol. 58, 12, 1985, pp. 119–132.

[9] Lipshitz (L.). – D-finite power series. *Journal of Algebra*, vol. 122, 12, 1989, pp. 353–373.

[10] 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, 12, 1994, pp. 163–177.

[11] Singer (Michael F.). – Algebraic relations among solutions of linear differential equations. *Transactions of the American Mathematical Society*, vol. 295, 12, 1986, pp. 753–763.

[12] Sloane (N. J. A.). – *The On-Line Encyclopedia of Integer Sequences*. – 2006. Published electronically at `http://www.research.att.com/~njas/sequences/`.

[13] Stanley (R. P.). – Differentiably finite power series. *European Journal of Combinatorics*, vol. 1, 12, 1980, pp. 175–188.

[14] Stanley (Richard P.). – *Enumerative combinatorics*. – Cambridge University Press, 1999, vol. 2, xii+581p.

[15] 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`.

CHAPITRE 9

Approximants de Padé et de Padé-Hermite

Résumé

L'algorithme d'Euclide étendu permet le calcul d'approximants de Padé. Plus généralement, il permet la reconstruction des fractions rationnelles. Les approximants de Padé-Hermite sont une généralisation des approximants de Padé. Leur calcul peut s'effectuer grâce à un algorithme qui peut être vu comme une généralisation de l'algorithme d'Euclide étendu. L'approximation de Padé-Hermite rend possible la reconstruction d'équations linéaires à coefficients polynomiaux reliant des séries formelles.

Un premier problème abordé dans ce chapitre est le calcul d'approximants de Padé. Plus généralement, on s'intéresse à la *reconstruction rationnelle*, dont un autre cas particulier important est l'interpolation des fractions rationnelles. En Section 1, nous ramenons la reconstruction rationnelle à l'algorithme d'Euclide étendu, et l'appliquons à la reconnaissance d'une suite récurrente linéaire à partir de ses premiers termes (algorithme de Berlekamp-Massey). Les applications en sont nombreuses : pour la résolution de systèmes différentiels à coefficients constants (Chapitre 15), pour le calcul de polynômes minimaux de matrices creuses (Chapitre 10), pour la résolution de systèmes linéaires à coefficients polynomiaux (Chapitre 14).

Un second problème, traité en Section 2, est le calcul d'approximants de Padé-Hermite. Il peut s'effectuer grâce à un algorithme qui généralise celui pour le calcul des approximants de Padé. L'approximation de Padé-Hermite permet d'effectuer la reconstruction de polynômes annulant une série algébrique, et d'opérateurs différentiels annulant une série D-finie. Il permet également de deviner une récurrence à coefficients polynomiaux d'ordre arbitraire à partir des premiers termes de la suite.

1. Reconstruction rationnelle

Définition 1. Soit $\mathbb{K}$ un corps, $A \in \mathbb{K}[X]$ un polynôme de degré $n > 0$ et $B \in \mathbb{K}[X]$ de degré $< n$. Pour un $k \in \{1, \ldots, n\}$ fixé, la *reconstruction rationnelle de B modulo A* est la recherche d'un couple de polynômes $(R, V) \in \mathbb{K}[X]^2$ vérifiant :

$$\text{(RR)} \qquad \operatorname{pgcd}(V, A) = 1, \quad \deg(R) < k, \quad \deg(V) \leq n - k \quad \text{et} \quad \frac{R}{V} \equiv B \bmod A.$$

On parle d'*approximation de Padé* lorsque $A = X^n$, et d'*interpolation de Cauchy* lorsque $A = \prod_i (X - u_i)$ avec $u_1, \ldots, u_n \in \mathbb{K}$ deux à deux distincts.

Remarque. Si $k = n$, alors clairement $(R, V) = (B, 1)$ est une solution du problème (RR).

Si $k < n$, il est possible que (RR) n'admette aucune solution. C'est le cas par exemple en prenant $n = 3, k = 2$ et $A = X^3, B = X^2 + 1 \in \mathbb{K}[X]$. En effet, si $V(X) = aX + b$ avec $b \neq 0$, alors $R \equiv (aX + b)(X^2 + 1) = bX^2 + aX + b \bmod X^3$, ce qui est incompatible avec $\deg(R) \leq 1$.

Par ailleurs, si (RR) admet une solution (R, V), alors la fraction rationnelle R/V est forcément unique. En effet, si $(R_1, V_1) \in \mathbb{K}[X]^2$ est une autre solution de (RR), alors $R_1/V_1 \equiv R/V \bmod A$, donc A divise $R_1V - V_1R$. Or, le polynôme $R_1V - V_1R$ ayant un degré strictement inférieur à celui de A, il doit être identiquement nul. Donc les fractions R/V et R_1/V_1 coïncident.

Un problème plus simple. Si $R, V \in \mathbb{K}[X]$ sont tels que (R, V) est solution du problème (RR), alors (R, V) vérifie aussi le problème plus simple

$$\text{(RRS)} \qquad \deg(R) < k, \quad \deg(V) \leq n - k \quad \text{et} \quad R \equiv VB \bmod A,$$

où, à la différence de (RR), on a mis de côté la contrainte sur le pgcd de A et de V.

Remarquons tout de suite que le problème (RRS) admet *toujours* une solution non triviale $(R, V) \neq (0, 0)$: en effet, il se traduit en termes d'algèbre linéaire en un système linéaire homogène

ayant $k+(n-k+1)=n+1$ inconnues (les coefficients de R et de V) et n équations. Par ailleurs, la solution R/V du problème (RRS) est encore unique (même idée de preuve que pour (RR) : si A divise à la fois $R-VB$ et R_1-V_1B, alors il divise également la combinaison linéaire $RV_1-R_1V=(R-VB)V_1-(R_1-V_1B)V$).

Lien avec le pgcd étendu. Nous allons prouver en §1.1 que le problème (RRS) peut être résolu en utilisant l'algorithme d'Euclide étendu AEE en page 162. Nous allons en déduire ensuite une procédure de décision et calcul pour (RR).

L'intuition du lien entre le problème (RRS) et le calcul du pgcd étendu s'appuie sur la remarque simple suivante : la congruence $R \equiv VB \bmod A$ équivaut à l'existence d'un polynôme U tel que $R=UA+VB$; or, cette dernière égalité est une *relation de type Bézout*.

Pour préciser un peu cette remarque, traitons brièvement le problème (RRS) dans le cas particulier $k=1$. Si A et B sont premiers entre eux, alors on prend $R=1$ et la relation de Bézout $R=UA+VB$ avec $\deg(V)<\deg(A)=n$ fournit la réponse. Sinon, on prend $R=0$, et comme le ppcm de A et B a un degré $<m+n$, le polynôme $V=\mathrm{ppcm}(A,B)/B$ est de degré au plus $n-1$ et vérifie $VB=0 \bmod A$.

1.1. **Calcul de la reconstruction rationnelle.** Rappelons que l'algorithme d'Euclide étendu (AEE, page 162) calcule une suite de restes successifs dont le degré décroît, ainsi qu'une suite de cofacteurs (U_i,V_i). Les éléments de la i-ième itération vérifient l'identité de Bézout $U_iA+V_iB=R_i$. L'écriture matricielle de l'itération

$$\begin{bmatrix} U_i & V_i \\ U_{i+1} & V_{i+1} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q_i \end{bmatrix} \times \begin{bmatrix} U_{i-1} & V_{i-1} \\ U_i & V_i \end{bmatrix}$$

permet de déduire aisément que $U_iV_{i+1}-V_iU_{i+1}=(-1)^i$ quel que soit i, et en particulier que U_i et V_i sont premiers entre eux. Cela entraîne également que $\mathrm{pgcd}(R_i,V_i)=\mathrm{pgcd}(A,V_i)$.

La clé du résultat principal de cette section (Théorème 1 ci-dessous) est le lemme suivant, qui montre que les cofacteurs V_i dans l'algorithme AEE ont des degrés modérés et bien maîtrisés.

Lemme 1. *Soient $n_0:=\deg(A)>n_1:=\deg(B)>n_2>\ldots>n_\ell$ les degrés des restes R_i dans $\mathsf{AEE}(A,B)$. Alors :*

$$\deg(V_i)=n-n_{i-1} \qquad \textit{pour} \qquad 1\le i\le \ell+1. \tag{1}$$

Démonstration. On procède par récurrence complète sur i. L'initialisation est claire, puisque $\deg(V_1)=0$. Prouvons maintenant que si l'égalité (1) est vérifiée pour $1\le i\le k$, alors elle l'est aussi pour $i=k+1$.

On part de l'observation que $\deg(V_{k-1})=n-n_{k-2}<n-n_{k-1}=\deg(V_k)$, qui implique que le degré de $V_{k+1}=V_{k-1}-Q_kV_k$ est égal à $\deg(Q_kV_k)=\deg(Q_k)+\deg(V_k)=(n_{k-1}-n_k)+(n-n_{k-1})=n-n_k$. □

Remarque. Dans une situation « générique », les quotients successifs Q_i ont tous degré 1, et donc $n_{i+1}=n_i-1$ (la suite des restes est appelée *normale*), si bien que le lemme précédent implique l'égalité $\deg(V_i)=i-1$.

Nous sommes en mesure de prouver que le problème de reconstruction rationnelle (RR) peut être résolu à l'aide de l'algorithme AEE.

Théorème 1. *Soit $A\in\mathbb{K}[X]$ de degré $n>0$ et soit $B\in\mathbb{K}[X]$ de degré $<n$. Soit $k\in\{1,2,\ldots,n\}$ et soit (R_j,U_j,V_j) la j-ième ligne dans $\mathsf{AEE}(A,B)$, où j est choisi minimal tel que $\deg(R_j)<k$.*

1. *Il existe une solution $(R,V)\neq(0,0)$ de (RRS), à savoir $(R,V)=(R_j,V_j)$. Si, de plus, $\mathrm{pgcd}(R_j,V_j)=1$, alors (R,V) est aussi solution de (RR).*
2. *Si (RR) admet une solution et si $R/V\in\mathbb{K}(X)$ en est une forme irréductible, alors il existe une constante $\alpha\in\mathbb{K}\setminus\{0\}$ telle que $R=\alpha R_j$ et $V=\alpha V_j$.*

Le problème (RR) admet donc une solution si et seulement si $\mathrm{pgcd}(A,V_j)=1$.

Démonstration. Par construction, R_j vaut bien $U_jA+V_jB\equiv V_jB$ modulo A. Par ailleurs, le Lemme 1 montre que le degré de V_j vaut $n-\deg(R_{j-1})$ et est donc borné par $n-k$ (par la minimalité de j). Donc $(R,V)=(R_j,V_j)$ vérifie bien (RRS).

$\mathsf{ARR}(A, B, k)$

Entrée : $A, B \in \mathbb{K}[X]$ avec $n = \deg(A) > \deg(B)$, et $k \in \{1, \ldots, n\}$.
Sortie : Une solution (R, V) de (RR), ou 0 si une telle solution n'existe pas.

1. $R_0 := A$; $U_0 := 1$; $V_0 := 0$; $R_1 := B$; $U_1 := 0$; $V_1 := 1$; $i := 1$.
2. Tant que $\deg(R_i) \geq k$, faire :
   $(Q_i, R_{i+1}) := \mathsf{QuotRem}(R_{i-1}, R_i)$; # la division euclidienne
   $U_{i+1} := U_{i-1} - Q_i U_i$; $V_{i+1} := V_{i-1} - Q_i V_i$;
   $i := i + 1$
3. Si $\mathrm{pgcd}(A, V_i) = 1$, renvoyer (R_i, V_i) ; sinon renvoyer 0.

FIGURE 1: Algorithme de reconstruction rationnelle.

De plus, le $\mathrm{pgcd}(A, V_j)$ est égal au $\mathrm{pgcd}(R_j, V_j)$. Par conséquent, si ce dernier vaut 1, alors V_j est inversible modulo A et donc $(R, V) = (R_j, V_j)$ vérifie aussi (RR).

L'unicité (à une constante près) est plus délicate à prouver. Supposons que (R, V) est une solution de (RR). Alors R s'écrit $UA + VB$ pour un certain $U \in \mathbb{K}[X]$. Nous allons prouver que $(R, V) = (\alpha R_j, \alpha V_j)$ pour un certain $\alpha \in \mathbb{K}[X] \setminus \{0\}$.

Commençons par montrer l'égalité $U_j V = U V_j$. Supposons le contraire et considérons le système linéaire

$$\begin{pmatrix} U_j & V_j \\ U & V \end{pmatrix} \times \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} R_j \\ R \end{pmatrix}.$$

Le système étant supposé de Cramer, A s'écrit comme quotient de deux déterminants

$$A = \begin{vmatrix} R_j & V_j \\ R & V \end{vmatrix} \Big/ \begin{vmatrix} U_j & V_j \\ U & V \end{vmatrix}.$$

L'hypothèse $\deg(R_j) < k \leq \deg(R_{j-1})$ et le Lemme 1 entraînent que le degré du membre droit de l'égalité précédente est majoré par

$$\begin{aligned} \deg(R_j V - R V_j) &\leq \max\{k - 1 + \deg(V), \deg(R) + n - \deg(R_{j-1})\} \\ &\leq \max\{n - 1, (k - 1) + n - \deg(R_{j-1})\} = n - 1, \end{aligned}$$

ce qui contredit $\deg(A) = n$. Nous avons donc bien l'égalité $U_j V = U V_j$. Celle-ci implique que V_j divise $U_j V$, et puisque U_j et V_j sont premiers entre eux, il s'ensuit que V_j divise V. En écrivant $V = \alpha V_j$, avec $\alpha \in \mathbb{K}[X]$, on obtient $U V_j = V U_j = \alpha U_j V_j$, et donc $U = \alpha U_j$. Enfin, $R = UA + VB = \alpha(U_j A + V_j B) = \alpha R_j$.

Les polynômes R et V étant premiers entre eux, il s'ensuit que α est une constante de $\mathbb{K} \setminus \{0\}$, et que donc $\mathrm{pgcd}(R_j, V_j) = 1$. □

L'algorithme qui s'ensuit est donné en Figure 1 ; à noter la ressemblance avec l'algorithme AEE.

1.2. **Approximants de Padé.** On rappelle que si $n > 0$, $k \in \{1, 2, \ldots, n\}$ et si $B \in \mathbb{K}[X]$ est de degré $< n$, alors un *approximant de Padé pour B de type* $(k-1, n-k)$ est une fraction rationnelle $R/V \in \mathbb{K}[X]$ telle que

$$X \nmid V, \quad \deg(R) < k, \quad \deg(V) \leq n - k \quad \text{et} \quad \frac{R}{V} \equiv B \bmod X^n. \tag{2}$$

Le Théorème 1 (pour $A = X^n$) entraîne la conséquence suivante.

Corollaire 1. *Soit $B \in \mathbb{K}[X]$ de degré $< n$. Soit (R_j, U_j, V_j) la j-ième ligne dans $\mathsf{AEE}(X^n, B)$, où j est choisi minimal tel que $\deg(R_j) < k$. Alors :*

1. *L'équation (2) admet une solution si et seulement si $\mathrm{pgcd}(R_j, V_j) = 1$.*
2. *Si $\mathrm{pgcd}(R_j, V_j) = 1$, alors R_j/V_j est l'unique approximant de Padé pour B de type $(k - 1, n - k)$.*

L'algorithme qui s'en déduit est un cas particulier de l'algorithme ARR, en prenant $A = X^n$ et en remplaçant le test $\mathrm{pgcd}(A, V_i) = 1$ de l'étape (3) par $V_i(0) \neq 0$.

Berlekamp-Massey

Entrée : Une borne $N \in \mathbb{N}$ sur le degré du polynôme minimal de la suite $(a_n)_{n\geq 0}$ et les $2N$ premiers termes $a_0, \ldots, a_{2N-1} \in \mathbb{K}$.

Sortie : Le polynôme générateur minimal de $(a_n)_{n\geq 0}$.

1. $A := a_0 + a_1 X + \cdots + a_{2N-1} X^{2N-1}$.
2. Calculer $R, V \in \mathbb{K}[X]$ la solution de (3) telle que $V(0) = 1$.
3. $d := \max\{1 + \deg(R), \deg(V)\}$. Renvoyer $\bar{V} := V(1/X)X^d$.

Figure 2: L'algorithme de Berlekamp-Massey.

1.3. **Algorithme de Berlekamp-Massey.** L'algorithme décrit dans cette section permet de deviner des récurrences à coefficients constants d'ordre arbitraire à partir de termes consécutifs de la suite.

On se donne dans un corps $\mathbb{K}$ les $2N$ premiers éléments d'une suite récurrente linéaire $(a_n)_{n\geq 0}$ pour laquelle on sait qu'il existe un polynôme caractéristique de degré $d \leq N$, c'est-à-dire un polynôme

$$f(X) = f_d X^d + \cdots + f_0, \quad \text{tel que} \quad f_d a_{n+d} + \cdots + f_0 a_n = 0, \quad \text{pour tous} \quad n \geq 0.$$

Le problème est de calculer le polynôme minimal (*i.e.* le polynôme caractéristique de degré minimal) de $(a_n)_{n\geq 0}$. Le Lemme 2 (page 145) montre que ce calcul équivaut à la résolution du problème de Padé

$$\frac{R}{V} \equiv A \bmod X^{2N}, \quad X \not| \ V, \ \deg(R) < N, \ \deg(V) \leq N \ \text{ et } \ \operatorname{pgcd}(R, V) = 1. \tag{3}$$

En effet, il implique que $(R, V) = (N_0, \bar{P})$ est solution de (3). L'algorithme qui s'en déduit est donné en Figure 2.

1.4. **Interpolation rationnelle de Cauchy.** L'interpolation des fractions rationnelles, appelée aussi *interpolation de Cauchy*, est une généralisation naturelle de l'interpolation polynomiale (de Lagrange).

Soient $k \in \{1, \ldots, n\}$, $v_0, \ldots, v_{n-1} \in \mathbb{K}$ et soient $u_0, \ldots, u_{n-1}$ des points distincts de $\mathbb{K}$. On cherche une fraction rationnelle $R/V \in \mathbb{K}(X)$ avec $R, V \in \mathbb{K}[X]$ satisfaisant aux contraintes

$$V(u_i) \neq 0, \quad \frac{R(u_i)}{V(u_i)} = v_i \quad \text{pour } 0 \leq i < n, \quad \deg(R) < k, \ \deg(V) \leq n - k. \tag{4}$$

Le problème (4) est un cas particulier de reconstruction rationnelle. En effet, si l'on note $B \in \mathbb{K}[X]$ l'unique polynôme de degré $< n$ tel que $B(u_i) = v_i$, alors (4) équivaut à la reconstruction rationnelle de B modulo $A := (X - u_0) \cdots (X - u_{n-1})$.

Le Théorème 1 implique la procédure de décision et de calcul suivante : on calcule le polynôme interpolant B et le polynôme $A = (X - u_0) \cdots (X - u_{n-1})$. On calcule les éléments (R_j, V_j) de la j-ième ligne de $\mathsf{AEE}(A, B)$ avec j minimal tel que $\deg(R_j) < k$. Si $\operatorname{pgcd}(A, V_j) = 1$, alors R_j/V_j est l'unique forme canonique de la fraction rationnelle cherchée. Sinon, le problème (4) n'a pas de solution.

Une application importante de l'interpolation rationnelle est la reconnaissance de récurrences linéaires d'ordre 1 à coefficients polynomiaux.

Corollaire 2 (Devinette de récurrences hypergéométriques)**.** *Soit (a_n) une suite d'éléments non nuls de $\mathbb{K}$ vérifiant la récurrence $p_1(n)a_{n+1} + p_0(n)p_n = 0$, dont les coefficients (inconnus) $p_0(X)$ et $p_1(X)$ sont des polynômes de $\mathbb{K}[X]$, premiers entre eux. Si l'on connaît une borne d sur le degré de p_0 et de p_1 telle que $p_0(j) \neq 0$ pour $0 \leq j \leq 2d + 1$, alors on peut déterminer p_0 et p_1 par un calcul d'interpolation rationnelle, à partir des $2d + 1$ premiers termes de la suite (a_n).*

1.5. **Algorithmes rapides.** En combinant les résultats du Théorème 1 ci-dessus, du Théorème 3 en page 172, et du Chapitre 6 pour le calcul rapide des polynômes A et B définis en Section 1.4, on déduit le résultat de complexité suivant.

Théorème 2. *Soient $A \in \mathbb{K}[X]$ de degré $n > 0$, $B \in \mathbb{K}[X]$ de degré $< n$ et $k \in \{1, 2, \ldots, n\}$. Soient $v_0, \ldots, v_{n-1} \in \mathbb{K}$, et soient $u_0, \ldots, u_{n-1}$ des points distincts de $\mathbb{K}$. Il est possible de calculer en $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$:*

1. *une solution du problème reconstruction rationnelle* (RR) *;*
2. *un approximant de Padé pour B de type $(k-1, n-k)$;*
3. *une fraction rationnelle $F = R/V \in \mathbb{K}(X)$ telle que $\deg(V) \leq n-k$, $\deg(R) < k$, et $F(u_i) = v_i$ pour $0 \leq i < n$.*

Le cas entier. L'analogue pour les entiers de la reconstruction rationnelle est également calculable en complexité quasi-optimale. Si n et B sont deux entiers, $0 \leq B < n$, et étant donné $k \leq n$, il s'agit de calculer des entiers R et V tels que

$$\text{pgcd}(n, V) = 1 \quad \text{et} \quad B = \frac{R}{V} \mod n, \quad |R| < k, \quad 0 \leq V \leq \frac{n}{k}. \tag{5}$$

Théorème 3. *À partir de k, n et B, on peut calculer R et V satisfaisant à* (5) *en $O(\mathsf{M}_{\mathbb{Z}}(\log n)\log\log n)$ opérations binaires.*

Le cas des récurrences. Une conséquence utile du Théorème 2 à la reconnaissance rapide des suites est contenue dans le corollaire suivant. Sa généralisation aux cas des suites P-récursives arbitraires sera traitée en Section 2.

Corollaire 3. *Soit (a_n) une suite d'éléments de $\mathbb{K}$ vérifiant*
- *soit une récurrence linéaire à coefficients constants d'ordre au plus N ;*
- *soit une récurrence linéaire d'ordre 1 à coefficients polynomiaux de degré au plus N.*

À partir des premiers $2N+1$ termes de la suite, on peut retrouver les coefficients de la récurrence en $O(\mathsf{M}(N)\log N)$ opérations dans $\mathbb{K}$.

2. Approximants de Padé-Hermite

Définition 2. Soit $\mathbf{F} = {}^t(f_1, \ldots, f_n)$ un vecteur de $n \geq 1$ séries formelles de $\mathbb{K}[[X]]$, et soit $\mathbf{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$. Un vecteur non nul $\mathbf{P} = (P_1, \ldots, P_n)$ de polynômes de $\mathbb{K}[X]$ est appelé *approximant de Padé-Hermite de type* $\mathbf{d}$ *de* $\mathbf{F}$ si :

1. La valuation $\text{val}(\mathbf{P}\cdot\mathbf{F})$ de la série $\mathbf{P}\cdot\mathbf{F} = \sum_{i=1}^{n} P_i f_i$ est au moins égale à $\sigma := \sum(d_i+1) - 1$;
2. $\deg(P_i) \leq d_i$ pour tout $1 \leq i \leq n$.

L'entier σ est alors appelé *l'ordre de l'approximant*.

Exemple 1. Dans la terminologie de la section précédente, si $R/V \in \mathbb{K}(X)$ est un approximant de Padé de type $(k, n-k)$ de $B \in \mathbb{K}[[X]]$, alors (R, V) est un approximant de Padé-Hermite pour $(-1, B)$, de type $(k, n-k)$.

Exercice 1. Soient A, B deux polynômes de $\mathbb{K}[X]$, avec $n = \deg(A) > \deg(B)$. Montrer que (R, V) est solution du problème de reconstruction rationnelle (RRS) défini en page 187 si et seulement s'il existe un polynôme $U \in \mathbb{K}[X]$ tel que (R, V, U) soit un approximant de Padé-Hermite de $(-1, B, A)$ de type $(k-1, n-k, n-1)$.

Le premier résultat de cette section montre l'existence des approximants de Padé-Hermite et fournit également un premier algorithme pour leur calcul.

Théorème 4. *Tout vecteur de séries formelles $\mathbf{F} = (f_1, \ldots, f_n) \in \mathbb{K}[[X]]^n$ admet un approximant de Padé-Hermite de type $\mathbf{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$ donné.*

Démonstration. On procède par coefficients indéterminés : en écrivant $P_i = \sum_{j=0}^{d_i} p_{i,j} X^j$, on obtient un système linéaire homogène à $\sigma = \sum_i (d_i + 1) - 1$ équations en les $\sigma + 1$ inconnues $p_{i,j}$. Puisqu'il a moins d'équations que d'inconnues, ce système admet forcément une solution non triviale. □

L'algorithme qui découle de la preuve du Théorème 4 repose sur la recherche d'un élément non trivial dans le noyau d'une matrice à coefficients dans $\mathbb{K}$, de taille $\sigma \times (\sigma+1)$. En utilisant les résultats du Chapitre 3, on aboutit donc à un algorithme de complexité $O(\mathsf{MM}(\sigma)) = O(\sigma^\theta)$ pour $2 \le \theta \le 3$.

Le but de la suite de ce chapitre est de présenter un algorithme plus efficace, dû à Derksen, de complexité seulement quadratique en σ. En anticipant un peu, cet algorithme revient à effectuer une sorte de pivot de Gauss sur une matrice à coefficients dans $\mathbb{K}[X]$, mais de taille bien plus petite que σ (seulement linéaire en $\max(\mathbf{d})$). Dans le cas particulier $n = 2$, l'algorithme de Derksen a essentiellement la même complexité que le calcul d'approximants de Padé via l'algorithme d'Euclide étendu vu en Section 1.

2.1. **Algorithme de Derksen : idées et résultats préliminaires.** Pour simplifier la présentation, on se restreint dans la suite au cas où le type de l'approximant cherché est de la forme $\mathbf{d} = (d, \dots, d) \in \mathbb{N}^n$, pour un certain $d \in \mathbb{N}$.

L'idée de l'algorithme de Derksen est de construire non pas un seul approximant de Padé-Hermite, mais toute une famille de tels approximants, et cela de manière incrémentale. Plus exactement, pour $s = 0, 1, \dots$, il construit une base d'une forme spéciale, appelée « base minimale », du $\mathbb{K}[X]$-module

$$V_s := \{\mathbf{P} \in \mathbb{K}[X]^n \quad | \quad \mathrm{val}(\mathbf{P} \cdot \mathbf{F}) \ge s\}.$$

Comme nous le verrons plus loin, une telle base de V_σ pour $\sigma = nd + n - 1$ contiendra alors nécessairement un approximant de Padé-Hermite de type $\mathbf{d} = (d, \dots, d)$ de $\mathbf{F}$.

Observons que, grâce aux inclusions $X^s\mathbb{K}[X]^n \subseteq V_s \subseteq \mathbb{K}[X]^n$, le module V_s est libre de rang n. Précisons ce que l'on entend par « base minimale ». Pour ce faire, introduisons d'abord une notion de *degré* et de *type* d'un vecteur de polynômes.

Définition 3 (degré et type d'un vecteur de polynômes)**.** Pour un vecteur de polynômes $\mathbf{P} = (P_1, \dots, P_n) \in \mathbb{K}[X]^n$, son *degré* $\deg(\mathbf{P})$ et son *type* $\mathrm{type}(\mathbf{P})$ sont donnés par :

$$\deg(\mathbf{P}) = \max\{\deg(P_1), \dots, \deg(P_n)\} \ \text{ et } \ \mathrm{type}(\mathbf{P}) = \max\{i \mid \deg(\mathbf{P}) = \deg(P_i)\}.$$

Définition 4 (base minimale)**.** Soit $V \subseteq \mathbb{K}[X]^n$ un sous-module libre de rang n. Une suite $\mathbf{Q}_1, \dots, \mathbf{Q}_n$ est appelée « base minimale » de V si, pour tout i, le vecteur $\mathbf{Q}_i$ est non nul, de type i, et de degré minimal parmi les éléments de $V \setminus \{0\}$ de type i.

Le résultat suivant précise le lien entre une base minimale et l'approximation de Padé-Hermite.

Lemme 2. *Soit $d \ge 1$. Supposons que $\mathbf{Q}_1, \dots, \mathbf{Q}_n$ est une base minimale de V_{nd+n-1}. Soit ℓ tel que $\deg(\mathbf{Q}_\ell)$ est minimal. Alors $\mathbf{Q}_\ell$ est un approximant de Padé-Hermite de type $(d, \dots, d)$ pour $\mathbf{F}$.*

Démonstration. Le Théorème 4 montre l'existence d'un vecteur non nul $\mathbf{P}$ de V_{nd+n-1} tel que $\deg(\mathbf{P}) \le d$. Soit i le type de $\mathbf{P}$. La suite d'inégalités

$$\deg(\mathbf{Q}_\ell) \le \deg(\mathbf{Q}_i) \le \deg(\mathbf{P}) \le d,$$

prouve que $\mathbf{Q}_\ell$ est un approximant de Padé-Hermite de type $(d, \dots, d)$ de $\mathbf{F}$. □

Le résultat suivant montre qu'une base minimale est nécessairement une base du $\mathbb{K}[X]$-module V au sens usuel.

Théorème 5. *Soit $V \subseteq \mathbb{K}[X]^n$ un sous-module libre de rang n. Toute base minimale de V est une base du $\mathbb{K}[X]$-module V.*

Démonstration. Montrons d'abord qu'il s'agit d'un système de générateurs. Soit $W := \mathbb{K}[X]\mathbf{Q}_1 + \cdots + \mathbb{K}[X]\mathbf{Q}_n \subseteq V$. On suppose par l'absurde que $V \neq W$. Soit $\mathbf{P} \in V \setminus W$ un élément minimal dans $V \setminus W$ pour l'ordre $\mathbf{P} < \mathbf{Q}$ défini par

$$\begin{aligned} &\deg(\mathbf{P}) < \deg(\mathbf{Q}), \quad \text{ou} \\ &\deg(\mathbf{P}) = \deg(\mathbf{Q}) \quad \text{et} \quad \mathrm{type}(\mathbf{P}) < \mathrm{type}(\mathbf{Q}). \end{aligned} \tag{6}$$

Autrement dit, $\mathbf{P}$ est de type minimal parmi les éléments de degré minimal de $V \setminus W$.

Soit i le type de $\mathbf{P}$. Les relations $\text{type}(\mathbf{P}) = \text{type}(\mathbf{Q}_i)$ et $\deg(\mathbf{P}) \geq \deg(\mathbf{Q}_i)$ entraînent l'existence d'un monôme $q \in \mathbb{K}[X]$ de degré $\deg(q) = \deg(\mathbf{P}) - \deg(\mathbf{Q}_i)$ tel que $\text{type}(\mathbf{P} - q\mathbf{Q}_i) < \text{type}(\mathbf{P})$. Puisque $\deg(\mathbf{P} - q\mathbf{Q}_i) \leq \deg(\mathbf{P})$, on obtient que

$$\mathbf{P} - q\mathbf{Q}_i < \mathbf{P}.$$

Par la minimalité de $\mathbf{P}$, il s'ensuit que $\mathbf{P} - q\mathbf{Q}_i$ appartient à W, donc $\mathbf{P} \in W$, ce qui contredit le choix de $\mathbf{P}$.

Pour conclure la preuve, montrons que les $\mathbf{Q}_i$ forment une famille libre. Si $\sum_i a_i\mathbf{Q}_i = 0$ est une combinaison polynomiale nulle des $\mathbf{Q}_i$, on a que pour tout i, le vecteur $a_i\mathbf{Q}_i$ est de type i. L'assertion découle du lemme suivant. □

Lemme 3. *Si $\mathbf{P}, \mathbf{Q}$ ont types distincts, alors* $\text{type}(\mathbf{P} + \mathbf{Q}) \in \{\text{type}(\mathbf{P}), \text{type}(\mathbf{Q})\}$.

Démonstration. Supposons $j = \text{type}(\mathbf{P}) > i = \text{type}(\mathbf{Q})$. Si $\deg(\mathbf{P}) \geq \deg(\mathbf{Q})$, alors $\text{type}(\mathbf{P}+\mathbf{Q}) = j$ et si $\deg(\mathbf{P}) < \deg(\mathbf{Q})$, alors $\text{type}(\mathbf{P} + \mathbf{Q}) = i$. □

2.2. **Algorithme de Derksen : fonctionnement.** L'idée de l'algorithme est de construire de proche en proche une base minimale de V_s, partant de la base minimale des $\mathbf{Q}_k = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$ (avec 1 en position k) de V_0. D'après le Lemme 2, l'élément de degré minimal dans une base minimale de V_{nd+n-1} fournit un approximant de Padé-Hermite de type $(d, \ldots, d)$ de $\mathbf{F}$.

Le résultat suivant montre comment construire une base minimale de V_{s+1} à partir d'une base minimale de V_s, et il constituera le cœur de l'itération.

Théorème 6. *Soit $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ une base minimale de*

$$V_s = \{\mathbf{P} \in \mathbb{K}[X]^n \mid \text{val}(\mathbf{P} \cdot \mathbf{F}) \geq s\}.$$

1. *Si $\text{val}(\mathbf{Q}_i \cdot \mathbf{F}) \geq s + 1$ quel que soit i, alors V_{s+1} et V_s coïncident, et $\{\mathbf{Q}_1, \ldots, \mathbf{Q}_n\}$ est une base minimale de V_{s+1}.*
2. *Supposons que $1 \leq i \leq n$ est tel que les deux conditions suivantes soient réunies :*
   - *$\text{val}(\mathbf{Q}_i \cdot \mathbf{F}) = s$;*
   - *si $\text{val}(\mathbf{Q}_\ell \cdot \mathbf{F}) = s$ pour un $\ell \neq i$, alors $\mathbf{Q}_i < \mathbf{Q}_\ell$, où $<$ est l'ordre (6).*

   Alors :
   (a) *pour $\ell \neq i$, il existe un scalaire $\lambda_\ell \in \mathbb{K}$ tel que $\tilde{\mathbf{Q}}_\ell := \mathbf{Q}_\ell - \lambda_\ell\mathbf{Q}_i$ vérifie $\text{val}(\tilde{\mathbf{Q}}_\ell \cdot \mathbf{F}) > s$;*
   (b) *en posant $\tilde{\mathbf{Q}}_i = X\mathbf{Q}_i$, la suite $\tilde{\mathbf{Q}}_1, \ldots, \tilde{\mathbf{Q}}_n$ forme une base minimale de V_{s+1}.*

Démonstration. (1) L'inclusion $V_{s+1} \subseteq V_s$ est évidente, et inversement, le Théorème 5 montre que tout $\mathbf{P} \in V_s$ s'écrit comme combinaison linéaire $\sum_i a_i\mathbf{Q}_i$. Ainsi $\mathbf{P} \cdot \mathbf{F} = \sum_i a_i(\mathbf{Q}_i \cdot \mathbf{F})$ est de valuation au moins $s + 1$, donc $\mathbf{P} \in V_{s+1}$.

(2a) Si $\text{val}(\mathbf{Q}_\ell \cdot \mathbf{F}) > s$, on pose $\lambda_\ell = 0$; si $\text{val}(\mathbf{Q}_\ell \cdot \mathbf{F}) = s$, alors $\mathbf{Q}_\ell \cdot \mathbf{F} = c_\ell X^s + \cdots$ et $\mathbf{Q}_i \cdot \mathbf{F} = c_i X^s + \cdots$, avec $c_i \neq 0$, et alors $\lambda_\ell := c_\ell/c_i$ convient.

Pour (2b), commençons par montrer que $\tilde{\mathbf{Q}}_1, \ldots, \tilde{\mathbf{Q}}_{i-1}, \mathbf{Q}_i, \tilde{\mathbf{Q}}_{i+1}, \ldots, \tilde{\mathbf{Q}}_n$ reste une base minimale de V_s. Il suffit pour cela de montrer que pour $\ell \neq i$, le vecteur $\tilde{\mathbf{Q}}_\ell$ a même type et même degré que $\mathbf{Q}_\ell$. Si $\text{val}(\mathbf{Q}_\ell \cdot \mathbf{F}) > s$, cela est évident, car $\lambda_\ell = 0$ et donc $\tilde{\mathbf{Q}}_\ell = \mathbf{Q}_\ell$. Sinon, le choix de i assure que $\mathbf{Q}_\ell > \mathbf{Q}_i$, et donc $\mathbf{Q}_\ell$ et $\mathbf{Q}_\ell - \lambda_\ell\mathbf{Q}_i$ ont même degré et même type.

Montrons maintenant que $(\tilde{\mathbf{Q}}_1, \ldots, \tilde{\mathbf{Q}}_n)$ est une base minimale de V_{s+1}. Comme la multiplication par un polynôme ne change pas le type, celui de $\tilde{\mathbf{Q}}_i = X\mathbf{Q}_i$ est bien i. Il suffit donc de montrer que si $\mathbf{P} \in V_{s+1}$ est de type $\ell \in \{1, 2, \ldots, n\}$, alors $\deg(\mathbf{P}) \geq \deg(\mathbf{Q}_\ell)$. Si $\ell \neq i$, ceci est une évidence : comme $\mathbf{P}$ appartient à $V_{s+1} \subseteq V_s$, et comme la suite $\tilde{\mathbf{Q}}_1, \ldots, \tilde{\mathbf{Q}}_{i-1}, \mathbf{Q}_i, \tilde{\mathbf{Q}}_{i+1}, \ldots, \tilde{\mathbf{Q}}_n$ forme une base minimale de V_s, le degré de $\mathbf{P}$ est nécessairement au moins égal à $\deg(\tilde{\mathbf{Q}}_\ell) = \deg(\mathbf{Q}_\ell)$.

Dans la suite de la preuve, on peut donc supposer que $\mathbf{P} \in V_{s+1}$ est de type i, le but étant de montrer que $\deg(\mathbf{P}) \geq \deg(X\mathbf{Q}_i)$. Le Théorème 5 montre que $\mathbf{P}$ s'écrit $\mathbf{P} = \sum_{j \neq i} a_j\tilde{\mathbf{Q}}_j + a_i\mathbf{Q}_i$. Puisque $\text{type}(\mathbf{P}) = i$, le Lemme 3 entraîne $a_i \neq 0$. De plus, le degré de $\mathbf{P}$ est égal à celui de $a_i\mathbf{Q}_i$, d'après le Lemme 4 ci-dessous.

Comme $\text{val}(\mathbf{P}\cdot\mathbf{F}) > s$ et comme pour $k \neq i$, $\text{val}(\mathbf{Q}_k\cdot\mathbf{F}) > s$, on a nécessairement que $\text{val}(a_i) > 0$. En particulier $\deg(a_i) > 0$ et donc $\deg(\mathbf{P}) \geq 1 + \deg(\mathbf{Q}_i)$. □

Lemme 4. *Si* $\text{type}(\mathbf{P}) = \text{type}(\mathbf{Q}) = i$ *et* $\text{type}(\mathbf{P} + \mathbf{Q}) < i$, *alors* $\deg(\mathbf{P}) = \deg(\mathbf{Q})$.

Derksen

Entrée : $\mathbf{F} = (f_1, \ldots, f_n) \in \mathbb{K}[[X]]^n$ et $d \geq 1$.
Sortie : Un approximant de Padé-Hermite de $\mathbf{F}$, de type $(d, \ldots, d)$.

```
pour k de 1 à n définir
    Q_k := (0, 0, ..., 0, 1, 0, ..., 0), avec 1 en position k.
pour j de 0 à nd + n − 2 faire
    i := 0
    pour k de 1 à n faire
        c_k := coeff(Q_k · F, j)
        si c_k ≠ 0 et (Q_k < Q_i ou i = 0), alors i := k
    si i ≠ 0 alors
        Q_i := c_i^{-1} Q_i
        pour k de 1 à n faire
            si k ≠ i alors
                Q_k := Q_k − c_k Q_i
        Q_i := X Q_i
p := 1
pour k de 2 à n faire
    si deg(Q_k) < deg(Q_p), alors p := k
Renvoyer Q_p.
```

Figure 3: L'algorithme de Derksen pour les approximants de Padé-Hermite.

Démonstration. Soient $\mathbf{P} = (P_1, \ldots, P_n)$ et $\mathbf{Q} = (Q_1, \ldots, Q_n)$. L'hypothèse entraîne les égalités $\deg(P_i) = \deg(\mathbf{P})$ et $\deg(Q_i) = \deg(\mathbf{Q})$. Cela implique que P_i et Q_i ont le même degré ; sinon, le type de $\mathbf{P} + \mathbf{Q}$ serait égal à i. □

L'algorithme qui se déduit de la conjonction du Lemme 2 et du Théorème 6 est donné en Figure 3. Il est de complexité seulement quadratique en σ. En résumé, nous avons le résultat suivant.

Théorème 7. *Soit $\mathbf{F} = (f_1, \ldots, f_n) \in \mathbb{K}[[X]]^n$. Il est possible de calculer un approximant de Padé-Hermite de type $(d, \ldots, d)$ de $\mathbf{F}$ en $O(n\sigma^2) = O(n^3 d^2)$ opérations dans $\mathbb{K}$.*

Exercice 2. Écrire les détails de la preuve du Théorème 7.

Remarque. Dans le cas « générique » (dit *normal*), la sortie $\mathbf{Q} = \mathbf{Q}_1, \ldots, \mathbf{Q}_n$ de l'algorithme de Derksen est de degré

$$\begin{bmatrix} d+1 & d & \cdots & d & d \\ d+1 & d+1 & \cdots & d & d \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ d+1 & d+1 & \cdots & d+1 & d \\ d & d & \cdots & d & d \end{bmatrix}.$$

C'est la dernière ligne $\mathbf{Q}_n$ qui fournit l'approximant désiré.

2.3. **Approximants de Padé-Hermite de type arbitraire.** Pour calculer un approximant de Padé-Hermite de type $\mathbf{d} = (d_1, \ldots, d_n)$, il suffit de remplacer, dans l'algorithme donné en Figure 3, deg par $\deg_{\mathbf{d}}$ et type par $\mathrm{type}_{\mathbf{d}}$, où :

$$\deg_{\mathbf{d}}(\mathbf{P}) = \max_{1 \leq i \leq n} \{\deg(P_i) - d_i\}, \quad \mathrm{type}_{\mathbf{d}}(\mathbf{P}) = \max\{i \mid \deg(P_i) - d_i = \deg_{\mathbf{d}}(\mathbf{P})\}.$$

2.4. **Applications.** Si une série est connue comme rationnelle de degré au plus d, l'approximation de Padé de type (d, d) suffit pour reconstruire la fraction rationnelle. Une question naturelle est comment se généralise cette observation dans le cadre de l'approximation de Padé-Hermite. Les réponses sont partielles, mais entièrement satisfaisantes dans la pratique.

Recherche de relations à coeffcients polynomiaux entre séries formelles. Supposons qu'il existe une combinaison linéaire $\sum_{i=1}^{n} P_i f_i = 0$, à coefficients des polynômes $P_i(X) \in \mathbb{K}[X]$ de degrés

au plus d. Par ailleurs, supposons calculé un approximant de Padé-Hermite $\mathbf{Q} := (Q_1, \ldots, Q_n)$ de $\mathbf{F} = (f_1, \ldots, f_n)$ de type $\mathbf{d} = (d, \ldots, d)$, via l'algorithme de Derksen. La question est donc : quel lien y a-t-il entre $\mathbf{P} = (P_1, \ldots, P_n)$ et $\mathbf{Q}$?

D'abord, dans le cas *générique*, la réponse est très simple : $\mathbf{P}$ et $\mathbf{Q}$ sont identiques, à un coefficient scalaire près. En effet, $\mathbf{Q} = \mathbf{Q}_n$ et $\mathbf{Q}_1, \ldots, \mathbf{Q}_n$ est une base de V_{nd+n-1} dont les degrés sont décrits en page 194. En particulier, $\mathbf{P}$ doit être une combinaison linéaire des $\mathbf{Q}_i$. Des considérations sur les degrés, utilisant la forme bien particulière des degrés des $\mathbf{Q}_i$, mènent à la conclusion désirée.

En effet, si $(c_1(X), \ldots, c_n(X)) \cdot {}^t(\mathbf{Q}_1, \ldots, \mathbf{Q}_n) = \mathbf{P}$, alors $c_1 Q_{11} + \cdots + c_n Q_{n1} = P_1$, et comme $\deg(Q_{11}) = d + 1$ et $\deg(Q_{j1}) = d$ pour $j > 1$ et $\deg(P_1) \leq d$, on obtient que $c_1 = 0$. De même, $c_2 = \cdots = c_{n-1} = 0$ et c_n doit être une constante $c \in \mathbb{K}$ telle que $\mathbf{P} = c\mathbf{Q}$.

Dans le cas général, un argument de nothérianité permet de prouver que pour $D \gg 0$, V_{nD+n-1} contient la relation $\mathbf{P}$, qui sera trouvée par l'algorithme de Derksen. Seulement, on ne dispose pas de borne *a priori* en fonction de d, sur le D minimal avec cette propriété. En effet, si on note

$$W_j := \{\mathbf{Q} \in \mathbb{K}[[X]]^n \mid \operatorname{val}(\mathbf{Q} \cdot \mathbf{F}) \geq j \quad \text{et} \quad \deg(\mathbf{Q}) \leq \deg(\mathbf{P})\},$$

et $W_\infty = \cap_{j \geq 0} W_j$, alors W_∞ contient toutes les relations de F en degré d, et en particulier $\mathbf{P}$. Puisque $W_0 \supseteq W_1 \supseteq W_2 \supseteq \cdots$ est une suite décroissante d'espaces vectoriels de dimension finie, elle est stationnaire, donc il existe un N tel que $W_\infty = W_N = W_{N+1} = \cdots$. Le cas *normal* correspond à la situation où $\dim(W_{k+1}) = \dim(W_k) - 1$ pour chaque k (noter la ressemblance avec la normalité de la suite des restes dans l'algorithme d'Euclide).

Reconstruction d'équations algébriques et différentielles. Deux cas particuliers importants, pour lesquels on peut être encore plus précis, sont les approximants algébriques et différentiels.

Soit $A \in \mathbb{K}[[X]]$ une série formelle algébrique. Le problème d'approximation algébrique consiste à retrouver, à partir des premiers termes de f un polynôme $P(X, Y) \in \mathbb{K}[X, Y]$ tel que $P(X, A(X)) = 0$. Si un tel P, de degré d en X et n en Y, existe, alors les coefficients des puissances de Y formeront un approximant de Padé-Hermite de type $(d, \ldots, d)$ du vecteur de séries $(1, A, \ldots, A^n)$. La difficulté vient de ce que un calcul d'approximation de Padé-Hermite ne trouve, *a priori*, qu'un polynôme $Q \in \mathbb{K}[X, Y]$ tel que $Q(X, A(X)) = 0 \bmod X^\sigma$, où $\sigma = (n+1)d - 1$. On dit alors qu'on a *deviné* un polynôme annulateur Q de A. Pour les approximants différentiels, la problématique est la même, la seule différence étant qu'on calcule un approximant de Padé-Hermite du vecteur des dérivées successives $(A, A', \ldots, A^{(n)})$.

Certification d'identités algébriques. Pour un approximant algébrique, il se pose la question de la *certification a posteriori*. Pour un polynôme annulateur deviné Q, cela veut dire qu'on souhaiterait déduire que non seulement $Q(X, A(X))$ est nul modulo X^σ, mais aussi que $Q(X, A(X)) = 0$.

Le résultat suivant apporte une réponse partielle à la question de la certification. Son avantage est qu'il ne dépend pas de l'algorithme utilisé pour produire l'approximant de Padé-Hermite.

Théorème 8. *Supposons que $A \in \mathbb{K}[[X]]$ est racine d'un polynôme irréductible de $\mathbb{K}[X, Y]$ de degré au plus d en X et au plus n en Y. Soit $\mathbf{Q} = (Q_0, Q_1, \ldots, Q_n)$ un approximant de Padé-Hermite de type $(d, \ldots, d)$ de $\mathbf{F} = (1, A, \ldots, A^n)$.*

Si $\operatorname{val}(\mathbf{Q} \cdot \mathbf{F}) \geq 2dn$, alors $\mathbf{Q} \cdot \mathbf{F} = 0$, c'est-à-dire que A est racine du polynôme $Q = \sum_{i=0}^n Q_i Y^i$.

Démonstration. Soit $P \in \mathbb{K}[X, Y]$ un polynôme irréductible de degré au plus d en X et au plus n en Y tel que $P(X, A) = 0$. Le polynôme $\operatorname{Res}_Y(P, Q) \in \mathbb{K}[X]$ est de degré borné par $\deg_X(P) \deg_Y(Q) + \deg_Y(P) \deg_X(Q) \leq 2dn$. Comme il est de la forme $uP + vQ$ pour u et v deux polynômes, avec $\deg_Y(v) < \deg_Y(P)$, et qu'il ne change pas lorsqu'on remplace la variable Y par la série A, c'est un $O(X^{2dn})$. Par conséquent $uP + vQ = 0$, donc P divise vQ et, étant irréductible, il divise v ou Q. Or $\deg_Y(v) < \deg_Y(P)$, donc finalement $Q \mid P$ et $Q(X, A)$ est la série nulle. □

Reconstruction de récurrences. Soit $(a_n)_n \in \mathbb{K}^{\mathbb{N}}$ une suite vérifiant une récurrence linéaire à coefficients polynomiaux. Comment, à partir des premiers termes de la suite, retrouver les coefficients de cette récurrence ?

Une idée consiste à utiliser la dualité montrée au Théorème 1 en page 177, et à chercher une équation différentielle, à coefficients polynomiaux, portant sur la série génératrice de la suite. Cette

équation peut être devinée comme expliqué précédemment, grâce à une approximation de Padé-Hermite. Il suffit ensuite de passer de l'équation différentielle à l'expression de la récurrence, ce qui n'est pas trivial mais purement formel et a été décrit au Chapitre 8.

2.5. **Calcul quasi-optimal.** Il se trouve que pour le calcul d'approximants de Padé-Hermite, on dispose d'un algorithme quasi-optimal, dont l'efficacité est utilisée en calcul formel comme la base de nombreux autres algorithmes. Cet algorithme peut être vu comme une alternative à l'approche par *matrices structurées*, de même complexité, décrite au Chapitre 11.

Théorème 9. *Soient* $\mathbf{F} = (f_1, \ldots, f_n) \in \mathbb{K}[[X]]^n$, $\mathbf{d} = (d_1, \ldots, d_n) \in \mathbb{N}^n$ *et* $\sigma = d_1 + \cdots + d_n + n - 1$. *Il est possible de calculer un approximant de Padé-Hermite de type* $\mathbf{d}$ *de* $\mathbf{F}$ *en* $O(\mathsf{MM}(n, \sigma) \log \sigma) = O(n^\theta \mathsf{M}(\sigma) \log(\sigma))$ *opérations dans* $\mathbb{K}$.

Démonstration. [Esquisse] Comme pour l'algorithme quadratique de Derksen, une première idée est de calculer non pas un vecteur, mais toute une matrice d'approximants. Une seconde idée est d'utiliser un « diviser pour régner » tout en exploitant la multiplication rapide de matrices polynomiales. Voici l'esquisse de l'algorithme dans le cas où $\mathbf{d} = (d, \ldots, d)$:

1. Si $N = 1$, renvoyer une base de l'orthogonal à $\mathbf{F}(0)$.
2. Sinon :
   (a) calculer récursivement une matrice polynomiale d'approximants $\mathbf{P}_1$ telle que $\mathbf{P}_1 \cdot \mathbf{F} = O(X^{\sigma/2})$ et $\deg(\mathbf{P}_1) \sim d/2$;
   (b) calculer le reste $\mathbf{R}$ tel que $\mathbf{P}_1 \cdot \mathbf{F} = X^{\sigma/2} \cdot \big(\mathbf{R} + O(X^{\sigma/2})\big)$;
   (c) calculer récursivement une matrice polynomiale $\mathbf{P}_2$ d'approximants du reste : $\mathbf{P}_2 \cdot \mathbf{R} = O(X^{\sigma/2})$, avec $\deg(\mathbf{P}_2) \sim d/2$;
   (d) renvoyer $\mathbf{P} := \mathbf{P}_2 \cdot \mathbf{P}_1$.

Le choix précis des degrés des approximants $\mathbf{P}_1$ et $\mathbf{P}_2$ est une question délicate et ne sera pas détaillé. En admettant ces choix, la correction de l'algorithme et son analyse de complexité se déduisent aisément. □

Notes

La présentation de la Section 1 est fortement inspirée par l'article [25], celle de la Section 2 par l'article [12]. Les approximants de Padé sont des objets très classiques en analyse numérique ; on pourra par exemple consulter le livre [1] qui leur est entièrement dédié.

L'algorithme de Berlekamp-Massey a été initialement conçu pour décoder des codes BCH, puis étendu par Zierler [38] et Massey [24], qui ont mis en évidence le lien avec les suites récurrentes linéaires à coefficients constants. Les liens forts entre l'algorithme d'Euclide, l'algorithme de Berlekamp-Massey, l'approximation de Padé et le calcul de fractions continues ont été étudiés dans [26, 37, 9, 14].

Les premiers algorithmes rapides pour la reconstruction rationnelle sont dus à [17]. Ces algorithmes ont été améliorés dans [5, 16, 8]. La problématique de la reconstruction rationnelle pour les entiers est classique en théorie des nombres [18] ; en calcul formel, elle a été très étudiée en lien avec les algorithmes modulaires, par exemple pour la résolution de systèmes linéaires par remontées p-adiques [13]. Des algorithmes rapides ont été récemment proposés dans [36, 29].

Les approximants de Padé et de Padé-Hermite ont été introduits et étudiés par Hermite et Padé dans [19, 21, 27, 28]. Une exposition plus moderne se trouve dans [23]. Le problème d'approximation de Padé-Hermite a été utilisé par Hermite en 1873 dans sa preuve [20] de la transcendance de e, qui utilise le choix très particulier $\mathbf{F} = (1, e^X, e^{2X}, \ldots)$. Deux autres cas particuliers importants sont : les « approximants algébriques » [32, 33] avec $\mathbf{F} = (1, f, f^2, \ldots)$ et les « approximants différentiels » [22] avec $\mathbf{F} = (f, f', f'', \ldots)$, où f est une série de $\mathbb{K}[[X]]$ donnée.

En calcul formel, les approximants de Padé-Hermite ont été introduits par Della Dora et Dicrescenzo [10, 11]. Un premier algorithme de complexité quadratique a été donné par Sergeyev [31]. Cet article est resté méconnu, et d'autres algorithmes quadratiques ont été proposés dans [30, 6, 35, 7]. L'algorithme de complexité quasi-optimale esquissé en Section 2.5 a été proposé dans [2].

Il existe diverses généralisations utiles du problème d'approximation de Padé-Hermite, par exemple les « approximants de Padé-Hermite simultanés et matriciels », ou encore des approximants modulo un polynôme arbitraire de degré $\sigma = \sum_i (d_i + 1) - 1$ au lieu de X^σ. Des algorithmes rapides existent pour toutes ces généralisations [3, 34, 2, 4, 15].

Bibliographie

[1] Baker, Jr. (George A.) and Graves-Morris (Peter). – *Padé approximants*. – Cambridge University Press, Cambridge, 1996, second edition, *Encyclopedia of Mathematics and its Applications*, vol. 59, xiv+746p.

[2] Beckermann (B.) and Labahn (G.). – A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM J. Matrix Anal. Appl.*, vol. 15, n3, 1994, pp. 804–823.

[3] Beckermann (Bernhard). – A reliable method for computing M-Padé approximants on arbitrary staircases. *J. Comput. Appl. Math.*, vol. 40, n1, 1992, pp. 19–42.

[4] Beckermann (Bernhard) and Labahn (George). – Fraction-free computation of matrix rational interpolants and matrix GCDs. *SIAM J. Matrix Anal. Appl.*, vol. 22, n1, 2000, pp. 114–144.

[5] Brent (Richard P.), Gustavson (Fred G.), and Yun (David Y. Y.). – Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, vol. 1, n3, 1980, pp. 259–295.

[6] Cabay (S.) and Labahn (G.). – A fast, reliable algorithm for calculating Padé-Hermite forms. In *ISSAC'89*. pp. 95–100. – ACM, New York, NY, USA, 1989.

[7] Cabay (S.), Labahn (G.), and Beckermann (B.). – On the theory and computation of nonperfect Padé-Hermite approximants. *J. Comput. Appl. Math.*, vol. 39, n3, 1992, pp. 295–313.

[8] Cabay (Stanley) and Choi (Dong Koo). – Algebraic computations of scaled Padé fractions. *SIAM J. Comput.*, vol. 15, n1, 1986, pp. 243–270.

[9] Cheng (Unjeng). – On the continued fraction and Berlekamp's algorithm. *IEEE Trans. Inform. Theory*, vol. 30, n3, 1984, pp. 541–544.

[10] Della Dora (J.) and Dicrescenzo (C.). – Approximants de Padé-Hermite. I. Théorie. *Numer. Math.*, vol. 43, n1, 1984, pp. 23–39.

[11] Della Dora (J.) and Dicrescenzo (C.). – Approximants de Padé-Hermite. II. Programmation. *Numer. Math.*, vol. 43, n1, 1984, pp. 41–57.

[12] Derksen (Harm). – *An algorithm to compute generalized Padé-Hermite forms*. – Report n9403, Dept. of Math., Catholic University Nijmegen, 1994.

[13] Dixon (John D.). – Exact solution of linear equations using p-adic expansions. *Numer. Math.*, vol. 40, n1, 1982, pp. 137–141.

[14] Dornstetter (Jean-Louis). – On the equivalence between Berlekamp's and Euclid's algorithms. *IEEE Trans. Inform. Theory*, vol. 33, n3, 1987, pp. 428–431.

[15] Giorgi (Pascal), Jeannerod (Claude-Pierre), and Villard (Gilles). – On the complexity of polynomial matrix computations. In *ISSAC'03*. pp. 135–142. – ACM, New York, 2003.

[16] Gragg (William B.), Gustavson (Fred G.), Warner (Daniel D.), and Yun (David Y. Y.). – On fast computation of superdiagonal Padé fractions. *Math. Programming Stud.*, n18, 1982, pp. 39–42. – Algorithms and theory in filtering and control (Lexington, Ky., 1980).

[17] Gustavson (Fred G.) and Yun (David Y. Y.). – Fast algorithms for rational Hermite approximation and solution of Toeplitz systems. *IEEE Trans. Circuits and Systems*, vol. 26, n9, 1979, pp. 750–755.

[18] Hardy (G. H.) and Wright (E. M.). – *An introduction to the theory of numbers*. – Oxford University Press, Oxford, 2008, sixth edition, xxii+621p.

[19] Hermite (C.). – Extrait d'une lettre de Monsieur Ch. Hermite à Monsieur Paul Gordan. *J. Reine Angew. Math.*, vol. 78, 1873, pp. 303–311.

[20] Hermite (C.). – Sur la fonction exponentielle. *C. R. Math. Acad. Sci. Paris*, vol. 77, 1873, pp. 18–24.

[21] Hermite (C.). – Sur la généralisation des fractions continues algébriques. *Ann. Math.*, vol. 21, n2, 1893, pp. 289–308.

[22] Khan (M. A. H.). – High-order differential approximants. *J. Comput. Appl. Math.*, vol. 149, n2, 2002, pp. 457–468.

[23] Mahler (K.). – Perfect systems. *Compositio Math.*, vol. 19, 1968, pp. 95–166 (1968).

[24] Massey (James L.). – Shift-register synthesis and BCH decoding. *IEEE Trans. Information Theory*, vol. IT-15, 1969, pp. 122–127.

[25] McEliece (Robert J.) and Shearer (James B.). – A property of Euclid's algorithm and an application to Padé approximation. *SIAM J. Appl. Math.*, vol. 34, n4, 1978, pp. 611–615.

[26] Mills (W. H.). – Continued fractions and linear recurrences. *Math. Comp.*, vol. 29, 1975, pp. 173–180.

[27] Padé (H.). – Sur la représentation approchée d'une fonction par des fractions rationelles. *Ann. Sci. École Norm. Sup.*, vol. 9, 1892, pp. 3–93.

[28] Padé (H.). – Sur la généralisation des fractions continues algébriques. *J. Math. Pures Appl.*, vol. 10, n4, 1894, pp. 291–330.

[29] Pan (Victor Y.) and Wang (Xinmao). – On rational number reconstruction and approximation. *SIAM J. Comput.*, vol. 33, n2, 2004, pp. 502–503.

[30] Paszkowski (Stefan). – Recurrence relations in Padé-Hermite approximation. *J. Comput. Appl. Math.*, vol. 19, n1, 1987, pp. 99–107.

[31] Sergeyev (A. V.). – A recursive algorithm for Padé-Hermite approximations. *USSR Comput. Maths Math. Phys*, vol. 26, n2, 1986, pp. 17–22.

[32] Shafer (R. E.). – On quadratic approximation. *SIAM J. Numer. Anal.*, vol. 11, 1974, pp. 447–460.

[33] Tourigny (Y.) and Drazin (Ph. G.). – The asymptotic behaviour of algebraic approximants. *R. Soc. Lond. Proc. Ser. A Math. Phys. Eng. Sci.*, vol. 456, n1997, 2000, pp. 1117–1137.

[34] Van Barel (M.) and Bultheel (A.). – A general module-theoretic framework for vector M-Padé and matrix rational interpolation. *Numer. Algorithms*, vol. 3, n1-4, 1992, pp. 451–461.

[35] Van Barel (Marc) and Bultheel (Adhemar). – The computation of nonperfect Padé-Hermite approximants. *Numer. Algorithms*, vol. 1, n3, 1991, pp. 285–304.

[36] Wang (Xinmao) and Pan (Victor Y.). – Acceleration of Euclidean algorithm and rational number reconstruction. *SIAM J. Comput.*, vol. 32, n2, 2003, pp. 548–556.

[37] Welch (L. R.) and Scholtz (R. A.). – Continued fractions and Berlekamp's algorithm. *IEEE Trans. Inform. Theory*, vol. 25, n1, 1979, pp. 19–27.

[38] Zierler (N.). – Linear recurring sequences and error-correcting codes. In *Error Correcting Codes (Proc. Sympos. Math. Res. Center, Madison, Wis., 1968)*, pp. 47–59. – Wiley, 1968.

CHAPITRE 10

Algèbre linéaire creuse : algorithme de Wiedemann

Résumé

Les matrices creuses sont les matrices contenant beaucoup d'éléments nuls. L'algorithme de Wiedemann est une méthode itérative pour résoudre des systèmes linéaires représentés par des matrices creuses. Il ramène la résolution au calcul du polynôme minimal, lui-même reposant sur la reconnaissance d'une suite récurrente à coefficients constants.

1. Introduction

Dans ce chapitre, on aborde des questions radicalement différentes de celles du Chapitre 3 : on ne cherche plus à effectuer les produits, inversions, ... de matrices quelconques, mais à manipuler des matrices *creuses*.

On ne commencera pas par définir précisément ce qu'est une matrice creuse. L'approche consiste à donner des algorithmes dont la complexité s'exprime en fonction du nombre d'éléments non nuls des matrices en entrée. 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 $n \times n$ à coefficients dans un corps $\mathbb{K}$ et vérifiant les hypothèses suivantes :

– A est inversible,

– A contient s éléments non nuls.

Le produit de A par un vecteur peut donc s'effectuer en $O(s)$ opérations dans $\mathbb{K}$.

On va décrire 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 denses, 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 : pour simplifier, on retiendra que l'exposant de l'algèbre linéaire creuse est 2 dans les bons cas.

2. Polynôme minimal et résolution de systèmes

L'algorithme de Wiedemann passe par le calcul du *polynôme minimal* de A. Rappelons sa définition.

Il est possible d'associer à tout polynôme en une variable $P = \sum_i p_i X^i$ de $\mathbb{K}[X]$ la matrice $P(A) = \sum_i p_i A^i$. Le théorème de Cayley-Hamilton affirme que le polynôme caractéristique χ_A de A annule la matrice A, c'est-à-dire que $\chi_A(A) = 0$.

Définition 1. Le polynôme minimal de A est le polynôme unitaire de plus petit degré annulant A.

Soit $\mu_A(X)$ le polynôme minimal de A, supposé connu. Puisqu'on a supposé A inversible, le terme constant de $\mu_A(X)$ n'est pas nul : il divise le terme constant du polynôme caractéristique, qui n'est lui-même pas nul.

L'égalité $\mu_A(A) = 0$ se réécrit sous la forme

$$A^{-1} = \frac{-1}{p_0}\left(A^{m-1} + p_{m-1}A^{m-2} + \cdots + p_1 I_n\right).$$

Algorithme de Wiedemann

Entrée : Une matrice creuse $A \in \mathcal{M}_n(\mathbb{K})$.

Sortie : Son polynôme minimal $\mu_A(X)$.

1. Choisir des vecteurs u et v aléatoires dans $\mathbb{K}^n$.
2. Pour $0 \leq i \leq 2n$, calculer les vecteurs $v_i = A^i v$, puis les scalaires $N_i = {}^t u\, v_i$.
3. Renvoyer le polynôme minimal de la suite (N_i).

Figure 1: Algorithme de Wiedemann pour calculer le polynôme minimal d'une matrice creuse.

Pour résoudre le système $Ax = b$, $(b \in \mathbb{K}^n)$, on va calculer $x = A^{-1}b$, c'est-à-dire

$$\frac{-1}{p_0}\left(A^{m-1}b + p_{m-1}A^{m-2}b + \cdots + p_1 b\right).$$

Ainsi, il suffit de calculer les itérés $b, Ab, \ldots, A^{m-1}b$, puis d'additionner les vecteurs $p_1 b, p_2 Ab, \ldots, A^{m-1}b$, pour retrouver x. L'analyse de complexité est immédiate :

- Chacun des $A^i b$ s'obtient à partir de $A^{i-1}b$ 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.

3. Calcul du polynôme minimal

Écrivons le polynôme minimal de A sous la forme

$$\mu_A(X) = X^m + p_{m-1}X^{m-1} + \cdots + p_0.$$

Alors la suite des puissances de A vérifie la récurrence linéaire

$$A^{k+m} + p_{m-1}A^{k+m-1} + \cdots + p_0 A^k = 0,$$

et ne vérifie aucune récurrence d'ordre plus petit.

Pour tous vecteurs u et v, la suite (de nombres) $N_i := {}^t u A^i v$ 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 vérifie 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 $D \in \mathbb{K}[U_1, \ldots, U_n, V_1, \ldots, V_n]$, non nul et 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 $\mu_A(X)$ grâce à l'algorithme de Berlekamp-Massey décrit en page 190 (attention, on ne connaît pas explicitement le polynôme D, mais son existence assure que la plupart des choix de u et v sont « chanceux »).

L'algorithme est donné en Figure 3. Il est probabiliste randomisé. L'analyse de probabilité est facile à faire en utilisant les résultats de l'exercice 1 ci-dessous.

La complexité de l'étape 2 est de $O(n)$ 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 $O(n^2) \subseteq O(sn)$ opérations.

4. Calcul du déterminant

L'algorithme en Figure 3 permet de détecter en cours de route l'inversibilité de la matrice A. En effet, si l'approximant de Padé calculé par l'algorithme de Berlekamp-Massey (page 190) a un dénominateur divisible par X, cela veut dire que $\det(A) = 0$.

Il est possible de calculer en bonne complexité le déterminant de la matrice creuse A, en exploitant le résultat suivant d'algèbre linéaire.

Lemme 1. *Si tous les mineurs principaux de $A \in \mathcal{M}_n(\mathbb{K})$ sont non nuls, alors les polynômes minimal et caractéristique de la matrice $B = A \cdot \mathrm{Diag}(X_1, \ldots, X_n)$ coïncident.*

L'algorithme qui s'en déduit est le suivant :
- Choisir une matrice de permutations aléatoire P, pour assurer que tous les mineurs principaux de la matrice AP sont inversibles ;
- choisir des éléments $x_1, \ldots, x_n$ aléatoirement dans $\mathbb{K}$;
- calculer le polynôme minimal μ_B de $B = A \cdot P \cdot \mathrm{Diag}(x_1, \ldots, x_n)$;
- renvoyer $\mu_B(0)/(x_1 \cdots x_n) \cdot \mathrm{sgn}(P)$.

À nouveau, la complexité de cet algorithme est quadratique en la taille n de A.

5. Calcul du rang

Le rang maximal de la matrice creuse A est détecté par les algorithmes précédents. Quitte à multiplier A par une matrice de permutations aléatoires, on peut donc supposer que son rang r est strictement inférieur à n, et que ses mineurs principaux A_i, pour $1 \leq i \leq r$, sont tous non nuls.

Sous ces hypothèses, il est possible de montrer que, pour un choix aléatoire d'éléments $x_1, \ldots, x_n$ dans $\mathbb{K}$, le rang r est égal à $\deg(\mu_{AD}) - 1$, où D est la matrice diagonale $D = \mathrm{Diag}(x_1, \ldots, x_n)$. L'algorithme qui s'ensuit calcule le polynôme minimal de la matrice creuse AD, et en déduit le rang de A, le tout pour une complexité quadratique en n.

Exercices

Exercice 1. Soit A une matrice dans $\mathcal{M}_n(\mathbb{K})$, soit b un vecteur de $\mathbb{K}^n \setminus \{0\}$ et soit f le polynôme minimal de la suite $(A^i \cdot b)_{i \geq 0}$.
Le but de l'exercice est d'estimer la probabilité $\mathcal{P}$ que f coïncide avec le polynôme minimal de la suite $({}^t u \cdot A^i \cdot b)_{i \geq 0}$ lorsque u est un vecteur de $\mathbb{K}^n$ dont les coordonnées sont choisies aléatoirement au hasard dans un sous-ensemble fini U de $\mathbb{K}$.

1. Montrer qu'il existe une application $\psi : \mathbb{K}^n \to \mathbb{A} = \mathbb{K}[X]/(f)$, $\mathbb{K}$-linéaire et surjective, telle que pour tout $u \in \mathbb{K}^n$ on ait

   $$f \text{ est le polynôme minimal de } ({}^t u \cdot A^i \cdot b)_{i \geq 0} \iff \psi(u) \text{ est inversible dans } \mathbb{A}.$$

   [Indication : l'application $\phi : \mathbb{K}^n \to \mathbb{K}^{\mathbb{N}}$ définie par $\phi(u) = ({}^t u \cdot A^i \cdot b)_{i \geq 0}$ induit une application linéaire surjective $\mathbb{K}^n \to M_f$, où M_f est l'espace vectoriel des suites de $\mathbb{K}^{\mathbb{N}}$ admettant f comme polynôme annulateur. Par ailleurs, $\mathbb{A}$ et M_f sont isomorphes en tant qu'espaces vectoriels.]

2. Soit d le degré de f. Montrer qu'il existe un polynôme non identiquement nul $R \in \mathbb{K}[X_1, \ldots, X_n]$ de degré total au plus d tel que pour tout vecteur $u = (u_1, \ldots, u_n) \in \mathbb{K}^n$ on ait

   $$\psi(u) \text{ est inversible dans } \mathbb{A} \text{ si et seulement si } R(u_1, \ldots, u_n) \neq 0.$$

   [Indication : utiliser un résultant.]

3. Montrer que R admet au plus $d \cdot |U|^{n-1}$ racines dans U^n. En déduire que la probabilité qu'un élément de U^n dont les coordonnées sont choisies aléatoirement au hasard dans U soit racine de R est bornée par $d/|U|$.

4. Conclure que la probabilité $\mathcal{P}$ vérifie

   $$\mathcal{P} \geq 1 - \frac{d}{|U|}.$$

Notes

L'algorithme de Wiedemann a été proposé dans [11]. Cet article contient également l'algorithme esquissé en Section 4. L'algorithme en Section 5 provient de [5]. Des versions « par blocs » de l'algorithme de Wiedemann ont été proposées dans [1, 10, 8, 9]. D'autres généralisations sont traités dans [3, 4, 6].

Les questions (2) et (3) de l'exercice 1 forment le cœur du « lemme de Zippel-Schwartz » [2, 12, 7].

Bibliographie

[1] Coppersmith (Don). – Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.*, vol. 62, n205, 1994, pp. 333–350.

[2] DeMillo (Richard A.) and Lipton (Richard J.). – A probabilistic remark on algebraic program testing. *Inform. Process. Lett.*, vol. 7, n4, 1978, pp. 193–195.

[3] Giesbrecht (M.), Lobo (A.), and Saunders (B. D.). – Certifying inconsistency of sparse linear systems. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation (Rostock)*. pp. 113–119 (electronic). – ACM, New York, 1998.

[4] Giesbrecht (Mark). – Fast computation of the Smith form of a sparse integer matrix. *Comput. Complexity*, vol. 10, n1, 2001, pp. 41–69.

[5] Kaltofen (Erich) and Saunders (B. David). – On Wiedemann's method of solving sparse linear systems. In *Applied algebra, algebraic algorithms and error-correcting codes (New Orleans, LA, 1991)*, pp. 29–38. – Springer, Berlin, 1991.

[6] Mulders (Thom). – Certified sparse linear system solving. *J. Symbolic Comput.*, vol. 38, n5, 2004, pp. 1343–1373.

[7] Schwartz (J. T.). – Fast probabilistic algorithms for verification of polynomial identities. *J. Assoc. Comput. Mach.*, vol. 27, n4, 1980, pp. 701–717.

[8] Thomé (E.). – Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm. *J. Symbolic Comput.*, vol. 33, n5, 2002, pp. 757–775.

[9] Turner (William J.). – A block Wiedemann rank algorithm. In *ISSAC'06*, pp. 332–339. – ACM, New York, 2006.

[10] Villard (G.). – Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems (extended abstract). In *ISSAC'97*. pp. 32–39. – ACM, New York, NY, USA, 1997.

[11] Wiedemann (D.). – Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, vol. IT-32, 1986, pp. 54–62.

[12] Zippel (Richard). – Probabilistic algorithms for sparse polynomials. In *EUROSAM'79*, pp. 216–226. – Springer, Berlin, 1979.

CHAPITRE 11

Algèbre linéaire structurée

Résumé

Les matrices possédant une structure spéciale sont omniprésentes en calcul formel. Ce sont des matrices dont les éléments jouissent d'une certaine répétition, ou satisfont à certaines relations. Plus formellement, une matrice structurée est typiquement définie par $O(n)$ éléments, au lieu de n^2 pour une matrice dense, et peut être multipliée par un vecteur en $\tilde{O}(n)$ opérations arithmétiques, au lieu de $O(n^2)$ opérations pour une matrice dense. Ce chapitre présente une algorithmique unifiée pour manipuler des matrices structurées denses, comme les matrices de Toeplitz, Hankel, Vandermonde et Sylvester. Les calculs avec les matrices de ces classes sont liés aux calculs avec les polynômes, ce qui permet l'emploi des techniques de multiplication polynomiale rapide pour accélérer leur manipulation. Par exemple, on peut résoudre un système linéaire défini par une matrice structurée inversible en $n \times n$ en $\tilde{O}(n)$ opérations.

1. Introduction

Nous avons vu au Chapitre 3 qu'il est possible de manipuler les matrices denses de taille $n \times n$ à coefficients dans un corps $\mathbb{K}$ en $O(n^\theta)$ opérations dans $\mathbb{K}$, où θ est un réel compris entre 2 et 3. Ici, par manipuler, on entend multiplier, inverser, calculer le déterminant, ou encore résoudre un système linéaire.

Dans certaines situations, les matrices qu'on est amené à manipuler présentent une structure que ces algorithmes généraux ne savent pas capturer et exploiter. Voici quelques exemples de matrices qui possèdent une structure.

Définition 1. Une matrice $A \in \mathcal{M}_n(\mathbb{K})$ est dite *de Toeplitz* si elle est invariante le long des diagonales, c'est-à-dire si ses éléments $a_{i,j}$ vérifient $a_{i,j} = a_{i+k,j+k}$ pour tout k.

Une telle matrice est complètement définie par sa première ligne et sa première colonne.

Définition 2. Une matrice $A \in \mathcal{M}_n(\mathbb{K})$ est dite *de Hankel* si elle est invariante le long des anti-diagonales, c'est-à-dire si ses éléments $a_{i,j}$ vérifient $a_{i,j} = a_{i-k,j+k}$ pour tout k.

Une telle matrice est complètement définie par sa première ligne et sa dernière colonne.

Exemple 1. La matrice de la multiplication polynomiale en degré fixé (dans les bases canoniques) par un polynôme fixé est de Toeplitz. Par exemple, la multiplication d'un polynôme de degré au plus 2 par le polynôme fixé $a_0 + a_1X + a_2X^2$ de $\mathbb{K}[X]$ se traduit matriciellement par l'égalité

$$\begin{bmatrix} a_0 & 0 & 0 \\ a_1 & a_0 & 0 \\ a_2 & a_1 & a_0 \\ 0 & a_2 & a_1 \\ 0 & 0 & a_2 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_0b_0 \\ a_0b_1 + a_1b_0 \\ a_0b_2 + a_1b_1 + a_2b_0 \\ a_1b_2 + a_2b_1 \\ a_2b_2 \end{bmatrix}.$$

Dans cet exemple, la matrice Toeplitz est d'une forme particulière, appelée *bande*. En fait, toute matrice de Toeplitz peut être vue comme sous-matrice d'une matrice de Toeplitz bande, et cela entraîne le résultat suivant.

Lemme 1. *Le produit d'une matrice de Toeplitz (ou de Hankel) de $\mathcal{M}_n(\mathbb{K})$ par un vecteur de $\mathbb{K}^n$ peut s'effectuer en $O(\mathsf{M}(n))$ opérations.*

Démonstration. Pour tout $0 \leq i \leq n-1$, l'élément c_i du produit matrice-vecteur

$$\begin{bmatrix} a_{n-1} & \cdots & a_0 \\ & \ddots & \\ a_{2n-2} & \cdots & a_{n-1} \end{bmatrix} \times \begin{bmatrix} b_0 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix}$$

est le coefficient de X^{n-1+i} dans le produit de polynômes

$$(b_0 + \cdots + b_{n-1}X^{n-1})(a_0 + \cdots + a_{2n-2}X^{2n-2}). \tag{1}$$

La preuve est tout à fait similaire pour une matrice de Hankel. □

Remarque. La preuve précédente montre que $2\,\mathsf{M}(n) + O(n)$ opérations suffisent pour multiplier une matrice Toeplitz ou Hankel de taille n par un vecteur. Cette borne peut être améliorée à $\mathsf{M}(n) + O(n)$, en observant que l'extraction de la *partie médiane* du produit (1) est suffisante, et en employant des algorithmes pour le produit médian, évoqués au Chapitre 12.

Exemple 2. Une matrice de Sylvester est la concaténation de deux matrices Toeplitz bande.

Définition 3. Une matrice $A = (a_{i,j})_{i,j=0}^{n-1}$ de $\mathcal{M}_n(\mathbb{K})$ est dite *de Vandermonde* si ses éléments s'écrivent $a_{i,j} = a_i^j$ pour $a_0, \ldots, a_{n-1} \in \mathbb{K}$ avec $a_i \neq a_j$ pour $i \neq j$.

Définition 4. Une matrice $A = (a_{i,j})_{i,j=0}^{n-1}$ de $\mathcal{M}_n(\mathbb{K})$ est dite *de Cauchy* si ses éléments s'écrivent $a_{i,j} = 1/(a_i - b_j)$ pour $a_i, b_j \in \mathbb{K}$ avec $a_i \neq b_j$ pour tous i, j.

Il y a un certain nombre de points en commun entre ces exemples : la matrice est représentable par $O(n)$ éléments ; le produit matrice-vecteur peut s'effectuer plus rapidement que dans le cas générique, en $O(\mathsf{M}(n)\log n)$ opérations au lieu de $O(n^2)$; le produit par une matrice quelconque peut s'effectuer en complexité $O(n\,\mathsf{M}(n)\log n)$ —quasi-optimale en la taille de la sortie pour une multiplication polynomiale à base de FFT. En effet, dans chaque cas, le produit matrice-vecteur Av admet une interprétation analytique :

- multiplication polynomiale dans les cas Toeplitz, Hankel et Sylvester ;
- évaluation polynomiale multipoint dans le cas d'une matrice de Vandermonde (Chapitre 6) ;
- évaluation multipoint de fractions rationnelles de la forme $\sum_{j=1}^{n-1} c_j/(X - b_j)$.

Exercice 1. Montrer que le produit d'une matrice de Cauchy par un vecteur peut s'effectuer en $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$.

Au vu de ces exemples, on pourrait donc être tenté de définir comme *structurée* une matrice telle que son produit par un vecteur peut s'effectuer en $\tilde{O}(n)$ opérations. La question qui se pose naturellement est : peut-on exploiter cette définition de la structure de A pour résoudre le système $Ax = b$? Un premier constat positif est que, dans chacun des exemples précédents, la résolution admet aussi une interprétation analytique :

(i) devinette de récurrences à coefficients constants, si A est de Toeplitz ou de Hankel ;
(ii) interpolation polynomiale, si A est de Vandermonde ;
(iii) interpolation de fractions rationnelles, si A est de Cauchy.

En effet, si la suite (a_n) d'éléments de $\mathbb{K}$ vérifie une récurrence (inconnue) à coefficients constants de la forme

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

alors trouver les coefficients p_i de cette récurrence revient à résoudre le système de Hankel

$$\begin{bmatrix} a_0 & a_1 & \cdots & a_{d-1} \\ a_1 & a_2 & \iddots & a_d \\ \vdots & \iddots & \iddots & \vdots \\ a_{d-1} & a_d & \cdots & a_{2d-2} \end{bmatrix} \times \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{d-1} \end{bmatrix} = \begin{bmatrix} a_d \\ a_{d+1} \\ \vdots \\ a_{2d-1} \end{bmatrix}.$$

Un autre point encourageant est que, pour ces trois opérations, on dispose d'algorithmes de complexité quasi-optimale $O(\mathsf{M}(n)\log n)$, présentés au Chapitre 6 pour (ii), et au Chapitre 9 pour (i) et (iii).

Il y a cependant quelques points négatifs : un premier est que si A est une matrice inversible telle que l'application linéaire $v \mapsto Av$ se calcule en L opérations, cela n'implique pas l'existence d'un

algorithme de complexité $O(L)$ pour l'application $v \mapsto A^{-1}v$. En d'autres termes, on manque d'un *principe d'inversion* analogue au *principe de transposition* présenté au Chapitre 12. Par exemple, pour les matrices creuses, le meilleur algorithme de résolution, dû à Wiedemann, est de complexité quadratique en n (Chapitre 10). Un second point négatif est que les classes vues plus haut ne sont pas stables par inversion : l'inverse d'une matrice de Toeplitz (resp. de Vandermonde) n'est pas de Toeplitz (resp. de Vandermonde).

On a donc besoin de rajouter des hypothèses dans la définition de la *bonne notion* de *matrice structurée*. Cette définition exploite la généralisation du caractère invariant par diagonale d'une matrice de Toeplitz, et vient de l'observation simple suivante : si A est de Toeplitz, alors la matrice

$$\phi(A) = A - (A \text{ décalée de 1 vers le bas et de 1 vers la droite})$$

admet une ligne et une colonne non nulles ; elle est donc de rang borné par 2. On dit que ϕ est un *opérateur de déplacement* et que *le rang de ϕ-déplacement* de A est au plus 2. On peut donc représenter $\phi(A)$ sous forme compacte, comme un produit $G \cdot {}^tH$, avec G et H des matrices rectangulaires de taille $n \times 2$.

Exemple 3. Si A est la matrice 3×3

$$A = \begin{bmatrix} c & d & e \\ b & c & d \\ a & b & c \end{bmatrix},$$

avec $d \neq 0$, alors $\phi(A)$ s'écrit

$$\phi(A) = \begin{bmatrix} c & d & e \\ b & 0 & 0 \\ a & 0 & 0 \end{bmatrix} = \begin{bmatrix} c & d \\ b & 0 \\ a & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & e/d \end{bmatrix}.$$

Définition 5. Les matrices (G, H) de $\mathcal{M}_{n,2}(\mathbb{K})$ telles que $\phi(A) = G \cdot {}^tH$ sont appelées *générateurs de déplacement* pour la matrice de Toeplitz A.

Ces définitions s'étendent, et permettent d'introduire le concept de matrice *quasi-Toeplitz*.

Définition 6. On appelle *opérateur de déplacement* ϕ_+ l'application $A \mapsto A - Z \cdot A \cdot {}^tZ$, où la matrice Z définie par

$$Z = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 \\ \vdots & \ddots & \dots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix}$$

est telle que ZA est la matrice A décalée de 1 ligne vers le bas, et $A \cdot {}^tZ$ est la matrice A décalée de 1 colonne vers la droite. Le *rang de déplacement* de A est l'entier $\alpha_+(A) = \text{rang}(\phi_+(A))$. On appelle *générateurs de déplacement* pour l'opérateur ϕ_+ un couple (G, H) de matrices de taille $n \times \alpha$ vérifiant l'égalité $\phi_+(A) = G \cdot {}^tH$. Si $\alpha_+(A) \ll n$, on dit que A est *quasi-Toeplitz*.

Intuitivement, le rang de déplacement mesure à quel point la matrice A est loin d'être de Toeplitz.

Exemple 4. Les matrices de Toeplitz, Hankel, Sylvester sont toutes quasi-Toeplitz, de rang de déplacement borné par 2.

Ces définitions s'étendent et permettent également de définir des matrices *quasi-Vandermonde* et *quasi-Cauchy*, mais pour des opérateurs de déplacement différents :

- $\mathbf{V_a} \mapsto \mathbf{V_a} - \text{Diag}_\mathbf{a} \cdot \mathbf{V_a} \cdot {}^tZ$ pour le cas Vandermonde ;
- $\mathbf{C_{a,b}} \mapsto \mathbf{C_{a,b}} - \text{Diag}_\mathbf{a}{}^{-1} \cdot \mathbf{C_{a,b}} \cdot \text{Diag}_\mathbf{a}$ pour le cas Cauchy.

Ici, pour deux vecteurs $\mathbf{a} = (a_0, \dots, a_{n-1})$ et $\mathbf{b} = (b_0, \dots, b_{n-1})$, on désigne par $\mathbf{V_a}$ la matrice de Vandermonde $(a_i^j)_{i,i=0}^{n-1}$, par $\text{Diag}_\mathbf{a}$ la matrice diagonale, dont la diagonale porte le vecteur $\mathbf{a}$, et par $\mathbf{C_{a,b}}$ la matrice de Cauchy $(1/(a_i - b_j))_{i,j=0}^{n-1}$.

Dans tous ces cas, $\phi_{M,N}$ est définie par $\phi_{M,N}(A) = A - MAN$, avec M et N bien choisies (en fonction de la structure visée), et le rang de déplacement de A est défini comme étant le rang de la matrice $\phi_{M,N}(A)$.

Exercice 2. Estimer les rangs de déplacement des matrices de Vandermonde et de Cauchy, pour les opérateurs de déplacement définis ci-dessus.

Résultat principal. L'idée-clé derrière les algorithmes rapides pour les matrices structurées de rang de déplacement α est l'utilisation des générateurs de déplacement comme structure de données compacte, de taille $O(\alpha n)$, donc proportionnelle à la taille n de la matrice, dans le cas où $\alpha \ll n$. L'énoncé suivant contient le résultat principal de ce chapitre.

Théorème 1. *Soit $\phi(\cdot)$ l'un des opérateurs Toeplitz, Vandermonde, Cauchy définis ci-dessus. Soit A une matrice de $\mathcal{M}_n(\mathbb{K})$ donnée par des générateurs de déplacement de taille $n \times \alpha$, et soit b un vecteur de $\mathbb{K}^n$. Alors, il est possible de :*

1. *calculer le déterminant de A ;*
2. *calculer le rang de A ;*
3. *calculer une solution du système $Ax = b$, ou prouver qu'il n'y en a aucune*

en $\tilde{O}(\alpha^2 n)$ opérations dans $\mathbb{K}$. Plus exactement, cette complexité s'exprime en termes de la fonction de multiplication polynomiale, et vaut :

1. *$O(\alpha^2\, \mathsf{M}(n) \log n)$ dans le cas quasi-Toeplitz ;*
2. *$O(\alpha^2\, \mathsf{M}(n) \log^2(n))$ dans les cas quasi-Vandermonde et quasi-Cauchy.*

Comme conséquence du Théorème 1 nous obtenons une *algorithmique unifiée* pour résoudre en complexité quasi-optimale différents problèmes sur les polynômes et les séries.

Corollaire 1. *On peut calculer en $O(\mathsf{M}(n)\log(n))$ opérations arithmétiques :*

1. *le pgcd étendu et le résultant de deux polynômes de degré borné par n ;*
2. *un approximant de Padé de type (n, n) d'une série tronquée donnée à précision $2n$;*

Démonstration. [Esquisse] Le résultant de deux polynômes $A, B \in \mathbb{K}[X]$ s'obtient comme le déterminant de leur matrice de Sylvester $\mathsf{Syl}(A, B)$, qui a un rang de déplacement au plus 2. Le degré du pgcd $G = \mathrm{pgcd}(A, B)$ s'obtient par un calcul de rang de la matrice de Sylvester : $\deg(G) = \deg(A) + \deg(B) - \mathrm{rang}(\mathsf{Syl}(A, B))$. Une fois connu le degré du pgcd, la relation de Bézout $UA + VB = G$, avec les contraintes $\deg(U) < \deg(B) - \deg(G)$ et $\deg(V) < \deg(A) - \deg(G)$, se traduit en un système linéaire en les coefficients de U et de V, dont la matrice est quasi-Toeplitz, de rang de déplacement au plus 3. Des considérations similaires s'appliquent pour le calcul d'approximants de Padé. □

Corollaire 2. *Étant données n series $f_1, \ldots, f_n$ de $\mathbb{K}[[X]]$ connues à précision $\sigma = \sum_i (d_i + 1) - 1$, il est possible d'en calculer un approximant de Padé-Hermite $(p_1, \ldots, p_n)$ de type $(d_1, \ldots, d_n)$ en $O(n^2\, \mathsf{M}(\sigma)\log(\sigma))$ opérations dans $\mathbb{K}$.*

Démonstration. [Esquisse] Il s'agit d'un problème linéaire en les coefficients de l'approximant cherché. La matrice du système linéaire est quasi-Toeplitz, de rang de déplacement borné par n. □

Dans la suite, nous allons esquisser les grandes lignes de la preuve du Théorème 1 dans le cas particulier où la matrice A est inversible et « suffisamment générique » (tous les mineurs non nuls), et uniquement dans le cas quasi-Toeplitz.

2. Le cas quasi-Toeplitz

Dans le reste du chapitre, nous allons nous concentrer uniquement sur le cas quasi-Toeplitz, car il contient les principales idées algorithmiques, et il couvre beaucoup d'applications ; les cas quasi-Vandermonde et quasi-Cauchy s'y ramènent et sont plus techniques.

Nous allons montrer que la notion de matrice quasi-Toeplitz est une bonne notion de structure, car elle répond aux propriétés suivantes :

(P1) on peut effectuer quasi-optimalement le produit d'une matrice quasi-Toeplitz par un vecteur ;

(P2) la somme et le produit de deux matrices quasi-Toeplitz restent quasi-Toeplitz ;

(P3) l'inverse aussi.

Or, ces propriétés forment le minimum nécessaire pour concevoir un algorithme de type *inversion de Strassen* (page 117) dans la représentation compacte par générateurs de déplacement.

2.1. **Produit matrice-vecteur, cas quasi-Toeplitz.** Pour montrer la propriété (P1) ci-dessus, le point clé est le résultat suivant.

Proposition 1. (*formule* ΣLU) *L'opérateur* $\phi_+ : A \mapsto A - Z \cdot A \cdot {}^tZ$ *est inversible. Plus exactement, on a la formule suivante, appelée* la formule ΣLU :

$$A - Z \cdot A \cdot {}^tZ = \sum_{i=1}^{\alpha} x_i \cdot {}^t y_i \quad \textit{si et seulement si} \quad A = \sum_{i=1}^{\alpha} L(x_i) \cdot U(y_i),$$

où les x_i *(resp.* y_i*) sont les colonnes du générateur* G *(resp.* H*), et où, pour un vecteur colonne* $v = {}^t[v_0 \cdots v_{n-1}]$*, on note* $L(v)$ *la matrice Toeplitz inférieure*

$$\begin{bmatrix} v_0 & 0 & \dots & 0 \\ v_1 & v_0 & \dots & 0 \\ \vdots & \ddots & \dots & \vdots \\ v_{n-1} & \cdots & v_1 & v_0 \end{bmatrix}$$

et $U(v)$ *la matrice Toeplitz supérieure* ${}^tL(v)$*.*

Démonstration. Par linéarité, il suffit de traiter le cas $\alpha = 1$. Si $C = L(a)U(b)$, alors un calcul immédiat montre que $c_{i+1,j+1} = a_i b_j + a_{i-1} b_{j-1} + \cdots$, et donc $c_{i+1,j+1} - c_{i,j} = a_i b_j$ et $\phi_+(C) = (a_i b_j)_{i,j=0}^{n-1} = a \cdot {}^t b$.

L'implication inverse découle de l'injectivité de ϕ_+ et est laissée en exercice. □

La Proposition 1 permet de donner une définition équivalente pour le rang de déplacement.

Définition 7. $\alpha_+(A)$ est le plus petit entier α tel qu'il existe une décomposition de la forme

$$A = \sum_{i=1}^{\alpha} L_i U_i$$

pour des matrices L_i de Toeplitz inférieures, et U_i de Toeplitz supérieures.

Exemple 5. Si A est de Toeplitz, alors elle admet la décomposition $A = A_{\inf} \cdot I_n + I_n \cdot A_{\sup}$, donc $\alpha_+(A) \leq 2$.

La définition 7 permet de prouver la propriété (P1) en page 206.

Corollaire 3. *Si* A *est donnée en représentation compacte par une paire de générateurs* (G, H) *de taille* $n \times \alpha$*, alors le produit matrice-vecteur* Av *peut s'effectuer en* $O(\alpha \mathsf{M}(n))$ *opérations arithmétiques.*

Démonstration. Le produit Av s'écrit comme la somme des α termes de la forme $L(x_i)(U(y_i)v)$. Or, chacun de ces termes peut être calculé en $O(\mathsf{M}(n))$ opérations, grâce au lemme 1. □

2.2. **Addition et produit en représentation compacte par générateurs.** Pour justifier la propriété (P2) en page 206, on va montrer plus, à savoir que la représentation par générateurs permet un calcul efficace (en temps quasi-linéaire) des opérations d'additions et de multiplication de deux matrices quasi-Toeplitz.

Proposition 2 (opérations matricielles en représentation compacte)**.** *Soient* (T, U) *générateurs de déplacement de taille* $n \times \alpha$ *pour* A*, et* (G, H) *générateurs de déplacement de taille* $n \times \beta$ *pour* B*. Alors*

1. $([T \mid G], [U \mid H])$ *sont des générateurs de déplacement pour* $A + B$ *de longueur* $\alpha + \beta$ *;*
2. $([T \mid W \mid a], [V \mid H \mid -b])$ *sont des générateurs de déplacement pour* AB*, de longueur* $\alpha + \beta + 1$*,*

où $V := {}^tB \cdot U$*,* $W := Z \cdot A \cdot {}^tZ \cdot G$*, et où le vecteur* a *(resp.* b*) est la dernière colonne de* $Z \cdot A$ *(resp. de* $Z \cdot {}^tB$*).*

La preuve, basée sur une vérification immédiate, est laissée en exercice.

Corollaire 4. *En représentation compacte par générateurs de déplacement, de longueurs au plus* α *pour* A *et* B*, on peut calculer*

1. *la somme* $A + B$ *en* $O(\alpha n)$ *opérations dans* $\mathbb{K}$ *;*

2. *le produit AB en $\mathsf{Mul}(n,\alpha) = O(\alpha^2\,\mathsf{M}(n))$ opérations dans $\mathbb{K}$.*

Démonstration. Le seul point non trivial est le calcul des matrices V, W, a et b. Tout repose sur la formule ΣLU. Si B s'écrit $\sum_{i=1}^{\beta} L(x_i)U(y_i)$, alors sa transposée s'écrit ${}^tB = \sum_{i=1}^{\beta} L(y_i)U(x_i)$, et le calcul de $V = {}^tB \cdot U$ se ramène à $\alpha\beta$ multiplications polynomiales, chacune pouvant s'effectuer en $O(\mathsf{M}(n))$. Le même raisonnement s'applique au calcul de W. Enfin, le calcul de a revient à multiplier A par la colonne ${}^t[0,\dots,0,1]$ et une observation similaire pour b permet de conclure. □

2.3. **Inversion en représentation compacte par générateurs.** Pour prouver la propriété (P3) en page 206, il est commode d'introduire un *opérateur dual de déplacement.*

Définition 8. L'opérateur de ϕ_- déplacement d'une matrice A de $\mathcal{M}_n(\mathbb{K})$ est défini par l'égalité

$$\phi_-(A) = A - {}^tZ \cdot A \cdot Z = A - (A \quad \text{décalée de 1 vers le haut et vers la gauche}).$$

Le $(-)$ rang de déplacement α_- est défini par la formule

$$\alpha_-(A) = \operatorname{rang}(A - {}^tZ \cdot A \cdot Z).$$

On peut montrer (de la même façon que pour ϕ_+) qu'on dispose d'une définition équivalente pour ϕ_-, et que ϕ_+ et ϕ_- sont liés.

Lemme 2.

1. *$\alpha_-(A)$ est le plus petit entier α tel que A puisse s'écrire sous la forme $\sum_{=1}^{\alpha} U_iL_i$, avec L_i Toeplitz inférieures, et U_i Toeplitz supérieures.*
2. *On a la* formule ΣUL :

$$A - {}^tZ \cdot A \cdot Z = \sum_{i=1}^{\alpha} x_i \cdot {}^ty_i \quad \textit{si et seulement si} \quad A = \sum_{i=1}^{\alpha} U(\operatorname{rev}(x_i)) \cdot L(\operatorname{rev}(y_i)).$$

   Ici, pour $v = {}^t[v_0,\dots,v_{n-1}]$, on note $\operatorname{rev}(v) = {}^t[v_{n-1},\dots,v_0]$.

3. *Une matrice de Toeplitz pour ϕ_+ est une matrice de Toeplitz pour ϕ_-.*

La preuve de ce résultat découle de la proposition suivante.

Proposition 3 (conversion $\Sigma LU \leftrightarrow \Sigma UL$)**.** *Pour toute matrice A, l'inégalité $|\alpha_+(A) - \alpha_-(A)| \le 2$ est satisfaite. De plus, on peut effectuer les conversions d'une représentation $\sum LU$ à une représentation $\sum UL$, et inversement, en $O(\alpha\,\mathsf{M}(n))$ opérations dans $\mathbb{K}$.*

Démonstration. Il suffit de prouver l'identité :

$$L(x) \cdot U(y) = I_n \cdot L(y') + U(x') \cdot I_n - U(x'') \cdot L(y''),$$

avec : $x'' = Z \cdot \operatorname{rev}(x)$, $y'' = Z \cdot \operatorname{rev}(y)$, $y' = \operatorname{rev}({}^tU(y) \cdot {}^tL(x) \cdot f)$, $x' = \operatorname{rev}(L(x) \cdot U(y) \cdot f)$, où $f = {}^t[0,\dots,0,1]$. □

Théorème 2. *Soit $A \in \mathcal{M}_n(\mathbb{K})$ une matrice quasi-Toeplitz inversible. Alors son inverse est aussi quasi-Toeplitz et $\alpha_+(A^{-1}) = \alpha_-(A)$.*

Démonstration. On a la suite d'égalités

$$\begin{aligned}\alpha_-(A) &= \operatorname{rang}(A - {}^tZ \cdot A \cdot Z) = \operatorname{rang}(I_n - A^{-1} \cdot {}^tZ \cdot A \cdot Z)\\ &= \operatorname{rang}(I_n - Z \cdot A^{-1} \cdot {}^tZ \cdot A) = \operatorname{rang}(A^{-1} - Z \cdot A^{-1} \cdot {}^tZ)\\ &= \alpha_+(A^{-1}),\end{aligned}$$

dans laquelle nous avons utilisé un fait d'algèbre linéaire élémentaire : si A, B sont deux matrices quelconques, alors $\operatorname{rang}(I_n - A \cdot B) = \operatorname{rang}(I_n - B \cdot A)$. □

Inversion rapide de matrices quasi-Toeplitz

Entrée : Une matrice « générique » $A \in \mathcal{M}_n(\mathbb{K})$, de taille $n = 2^k$, représentée par des générateurs (pour l'opérateur de déplacement ϕ_+).

Sortie : Son inverse A^{-1}, représentée par des générateurs (pour l'opérateur de déplacement ϕ_-).

1. Si $n = 1$, renvoyer A^{-1}.
2. Calculer des ϕ_+-générateurs pour $a, b, c, d \in \mathcal{M}_{n/2}(\mathbb{K})$, où $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.
3. Calculer récursivement des ϕ_--générateurs pour $e := a^{-1}$.
4. Calculer des ϕ_+-générateurs pour $Z := d - ceb$.
5. Calculer récursivement des ϕ_--générateurs pour $t := Z^{-1}$.
6. Renvoyer des ϕ_--générateurs de $A^{-1} = \begin{bmatrix} x & y \\ z & t \end{bmatrix}$ via les formules de Strassen $y := -ebt$, $z := -tce$ et $x := e + ebtce$.

FIGURE 1: Algorithme de type Strassen pour inverser une matrice quasi-Toeplitz.

2.4. **Résolution rapide de systèmes quasi-Toeplitz.** Le Théorème 2 ne nous dit pas *comment* calculer des générateurs pour A^{-1}. Cela est fait dans le dernier résultat de ce chapitre.

Théorème 3. *Soit $A \in \mathcal{M}_n(\mathbb{K})$ une matrice quasi-Toeplitz inversible, de rang de déplacement α, donnée par des générateurs G et H de taille $n \times \alpha$. On peut calculer des générateurs de taille $n \times \alpha$ pour A^{-1} en $O(\alpha^2\,\mathsf{M}(n)\log(n))$ opérations dans $\mathbb{K}$.*

À partir de cette représentation de l'inverse, le système $Ax = b$ ($b \in \mathbb{K}^n$) peut se résoudre pour un supplément de $O(\alpha\,\mathsf{M}(n))$ opérations dans $\mathbb{K}$.

Démonstration. L'idée est d'adapter au cas structuré l'algorithme d'inversion de Strassen présenté au Chapitre 3 (page 117). La différence réside dans le choix de la structure de données utilisée pour représenter et calculer avec les matrices quasi-Toeplitz. Pour ce faire, on utilise comme structure de données compacte les générateurs de déplacement pour les opérateurs ϕ_+ et ϕ_- (ou, de façon équivalente, les représentations ΣLU et ΣUL).

L'algorithme, de type « diviser pour régner », est décrit en Figure 1. Sa correction est héritée de celle de l'algorithme d'inversion de Strassen ; l'hypothèse sur la généricité de A est utilisée pour garantir que tous les mineurs qu'on doit inverser au cours de l'algorithme sont bien inversibles.

Sa complexité arithmétique $\mathsf{C}(n)$ obéit à la récurrence

$$\mathsf{C}(n) \leq 2\,\mathsf{C}(n/2) + O(\mathsf{Mul}(n,\alpha)) + O(\alpha^2 n + \alpha\,\mathsf{M}(n)),$$

où le coût en $O(\alpha^2 n + \alpha\,\mathsf{M}(n))$ provient des conversions entre les représentations ϕ_+ et ϕ_- et des calculs de minimisation de longueurs de générateurs, et la notation $\mathsf{Mul}(n,\alpha)$ désigne la complexité du produit matriciel pour des matrices $n \times n$ données par générateurs de taille $n \times \alpha$. La preuve se conclut en utilisant l'estimation

$$\mathsf{Mul}(n,\alpha) = O(\alpha^2\,\mathsf{M}(n)),$$

qui entraîne $\mathsf{C}(n) = O(\alpha^2\,\mathsf{M}(n)\log(n))$. □

EXERCICES

Exercice 3. Soit $A \in \mathcal{M}_n(\mathbb{K})$ une matrice quasi-Toeplitz de rang de déplacement $\alpha \ll n$, représentée de façon compacte par des générateurs (G, H) de taille $n \times \alpha$, *i.e.*, tels que $\phi_+(A) = G \cdot {}^tH$.

1. Soient $v_1, \ldots, v_\alpha$ des vecteurs quelconques de $\mathbb{K}^n$. Montrer que le calcul de tous les produits $A \cdot v_\ell$, $1 \leq \ell \leq \alpha$, se ramène, grâce à la formule ΣLU, au problème suivant :

(P) Étant donnés des polynômes $G_j, H_j, V_j \in \mathbb{K}[X]$ $(j \leq \alpha)$ de degrés au plus $n-1$, calculer

$$A_\ell = \sum_{j=1}^{\alpha} G_j \, (H_j V_\ell \bmod X^n), \qquad 1 \leq \ell \leq \alpha.$$

2. Donner un premier algorithme qui résout le problème **(P)** et estimer sa complexité.
3. Montrer que le problème **(P)** admet la reformulation matricielle suivante :

(MP) Étant données des matrices polynomiales $\mathbf{G}, \mathbf{V}$ dans $\mathcal{M}_{\alpha\times 1}(\mathbb{K}[X])$ et $\mathbf{H}$ dans $\mathcal{M}_{1\times\alpha}(\mathbb{K}[X])$, toutes de degrés au plus $n-1$, calculer $(\mathbf{V}\mathbf{H} \bmod X^n)\,\mathbf{G}$.

4. Soient $\mathbf{A}, \mathbf{B}$ et $\mathbf{C}$ des matrices polynomiales de tailles $(n\times p)$, $(p\times n)$ et $(n\times p)$ et de degré au plus d. Montrer que le produit $\mathbf{ABC}$ peut être calculé en $O(\frac{n}{p}\,\mathsf{MM}(p,d))$ opérations dans $\mathbb{K}$.
5. Proposer un algorithme de type « diviser pour régner » (par rapport à n) résolvant le problème **(MP)** en complexité $O(\frac{1}{\alpha}\,\mathsf{MM}(\alpha,n))$.
6. Conclure qu'on peut multiplier, dans la représentation par générateurs de déplacement, deux matrices quasi-Toeplitz de $\mathcal{M}_n(\mathbb{K})$ de rang de déplacement au plus α en $O(\frac{1}{\alpha}\,\mathsf{MM}(\alpha,n))$ opérations dans $\mathbb{K}$.

Notes

Levinson [13] a donné le premier algorithme de complexité quadratique pour la résolution d'un système linéaire dont la matrice est de Toeplitz, symétrique et définie positive. Trench [18] a par la suite montré que toute l'inverse d'une telle matrice peut se calculer en la même complexité. Ces deux algorithmes sont décrits dans [7, §4.7].

L'algorithme de Trench [18] repose sur une formule, reprise et améliorée par Gohberg et Semencul [6], qui fournit une représentation explicite ΣLU à deux termes de l'inverse d'une matrice de Toeplitz A, les sommands étant construits à partir de la première ligne et de la première colonne de A^{-1}. Brent, Gustavson et Yun [8, 3] montrent comment ramener le calcul des premières ligne et colonne de A^{-1} à un calcul d'approximation de Padé. L'article [3] met en évidence les liens profonds entre la résolution des systèmes de Toeplitz, l'algorithme d'Euclide étendu et l'approximation de Padé, et propose le premier algorithme pour la résolution d'un système défini par une matrice de Toeplitz inversible quelconque de taille n en $O(\mathsf{M}(n)\log n)$ opérations arithmétiques.

Parallèlement, la notion de rang de déplacement a été introduite par Kailath, Kung et Morf dans [10] et utilisée par les mêmes auteurs dans [9] pour donner un algorithme de complexité quadratique pour la résolution d'un système quasi-Toeplitz inversible. Plus exactement, l'algorithme de [9] calcule la solution d'un système défini par une matrice quasi-Toeplitz inversible de rang de déplacement α en $O(\alpha n^2)$ opérations. Une version rapide, de complexité $O(\alpha^d\,\mathsf{M}(n)\log n)$ a été obtenue indépendamment par Bitmead et Anderson [1] (avec $d=4$) et Morf [14] (avec $d=2$). Les algorithmes de [1, 14] fonctionnent seulement sous une hypothèse de forte régularité de l'entrée. Ils ont été adaptés dans [11, 12] au cas où la matrice du système est quelconque. Ces algorithmes ont été étendus au cas quasi-Cauchy dans [4, 17] et au cas quasi-Vandermonde dans [15, 5]. La meilleure complexité asymptotique vis-à-vis simultanément de la taille de la matrice et de son rang de déplacement est due à l'article [2], qui montre comment introduire du produit rapide de matrices denses dans l'algorithmique pour les matrices structurées, et réduit le coût à $\tilde{O}(\alpha^{\theta-1}n)$ pour la résolution des systèmes quasi-Toeplitz, quasi-Vandermonde et quasi-Cauchy.

Une bonne référence sur les matrices structurées est le livre [16].

Bibliographie

[1] Bitmead (Robert R.) and Anderson (Brian D. O.). – Asymptotically fast solution of Toeplitz and related systems of linear equations. *Linear Algebra Appl.*, vol. 34, 1980, pp. 103–116.

[2] Bostan (Alin), Jeannerod (Claude-Pierre), and Schost (Éric). – Solving structured linear systems with large displacement rank. *Theoret. Comput. Sci.*, vol. 407, n1-3, 2008, pp. 155–181.

[3] Brent (Richard P.), Gustavson (Fred G.), and Yun (David Y. Y.). – Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, vol. 1, n3, 1980, pp. 259–295.

[4] Cardinal (Jean-Paul). – On a property of Cauchy-like matrices. *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 328, n11, 1999, pp. 1089–1093.

[5] Gohberg (I.) and Olshevsky (V.). – Complexity of multiplication with vectors for structured matrices. *Linear Algebra Appl.*, vol. 202, 1994, pp. 163–192.

[6] Gohberg (I. C.) and Semencul (A. A.). – On the inversion of finite Toeplitz matrices and their continuous analogues (in russian). *Mat. Issled.*, vol. 7, n2, 1972, pp. 201–223.

[7] Golub (Gene H.) and Van Loan (Charles F.). – *Matrix computations*. – Johns Hopkins University Press, Baltimore, MD, 1996, third edition, *Johns Hopkins Studies in the Mathematical Sciences*, xxx+698p.

[8] Gustavson (Fred G.) and Yun (David Y. Y.). – Fast algorithms for rational Hermite approximation and solution of Toeplitz systems. *IEEE Trans. Circuits and Systems*, vol. 26, n9, 1979, pp. 750–755.

[9] Kailath (T.), Kung (S. Y.), and Morf (M.). – Displacement ranks of a matrix. *Bull. Amer. Math. Soc. (N.S.)*, vol. 1, n5, 1979, pp. 769–773.

[10] Kailath (Thomas), Kung (Sun Yuan), and Morf (Martin). – Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, vol. 68, n2, 1979, pp. 395–407.

[11] Kaltofen (Erich). – Asymptotically fast solution of Toeplitz-like singular linear systems. In *ISSAC'94*. pp. 297–304. – ACM, New York, NY, USA, 1994.

[12] Kaltofen (Erich). – Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. *Math. Comp.*, vol. 64, n210, 1995, pp. 777–806.

[13] Levinson (Norman). – The Wiener RMS (root mean square) error criterion in filter design and prediction. *J. Math. Phys. Mass. Inst. Tech.*, vol. 25, 1947, pp. 261–278.

[14] Morf (M.). – Doubling algorithms for Toeplitz and related equations. In *IEEE Conference on Acoustics, Speech, and Signal Processing*, pp. 954–959. – 1980.

[15] Pan (Victor). – On computations with dense structured matrices. *Math. Comp.*, vol. 55, n191, 1990, pp. 179–190.

[16] Pan (Victor Y.). – *Structured matrices and polynomials*. – Birkhäuser Boston Inc., Boston, MA, 2001, xxvi+278p. Unified superfast algorithms.

[17] Pan (Victor Y.) and Zheng (Ailong). – Superfast algorithms for Cauchy-like matrix computations and extensions. *Linear Algebra Appl.*, vol. 310, n1-3, 2000, pp. 83–108.

[18] Trench (William F.). – An algorithm for the inversion of finite Toeplitz matrices. *J. Soc. Indust. Appl. Math.*, vol. 12, 1964, pp. 515–522.

CHAPITRE 12

Principe de transposition de Tellegen

Résumé

Le principe de transposition est un ensemble de règles de transformation pour les algorithmes calculant des applications linéaires. Pour tout algorithme qui calcule des produits matrice-vecteur par une matrice fixée $\mathbf{M}$, ces règles de transformation permettent d'obtenir un algorithme dual qui calcule des produits matrice-vecteur par la matrice transposée de $\mathbf{M}$. En outre, la complexité arithmétique de l'algorithme dual est essentiellement égale à celle de l'algorithme initial.

1. Introduction

Le *théorème de transposition de Tellegen* affirme que, étant donnée une matrice $\mathbf{M}$ de taille $m \times n$ à coefficients dans un corps $\mathbb{K}$, sans lignes ni colonnes nulles, tout algorithme linéaire $\mathcal{A}$ qui calcule l'application linéaire $\mathbf{v} \mapsto \mathbf{M} \cdot \mathbf{v}$ de $\mathbb{K}^n \to \mathbb{K}^m$ en L opérations dans $\mathbb{K}$ peut être transformé en un *algorithme dual* ${}^t\mathcal{A}$ qui calcule l'application linéaire transposée $\mathbf{w} \mapsto {}^t\mathbf{M} \cdot \mathbf{w}$ de $\mathbb{K}^m \to \mathbb{K}^n$ en $L - n + m$ opérations dans $\mathbb{K}$. Ici, par *algorithme linéaire* on entend un algorithme qui n'utilise que des opérations linéaires en les éléments de l'entrée.

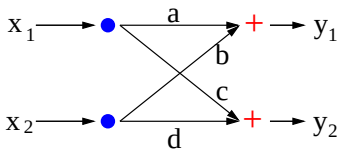
Par extension, on appelle *principe de Tellegen* l'ensemble des règles de transformation réalisant le passage de l'algorithme direct à l'algorithme dual.

Motivation. Si l'algorithme utilisé pour la multiplication matrice-vecteur est l'algorithme naïf, cet énoncé devient trivial. En effet, dans ce cas :

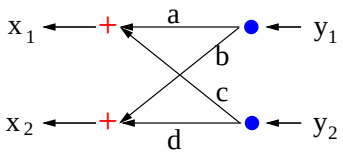
- $\mathbf{M} \cdot \mathbf{v}$ requiert $\sum_i (\alpha_i - 1) = E - m$ opérations $\pm$ et E opérations $\times$
- ${}^t\mathbf{M} \cdot \mathbf{w}$ requiert $\sum_j (\beta_j - 1) = E - n$ opérations $\pm$ et E opérations $\times$

où α_i (resp. β_j) est le nombre d'éléments non-nuls de la i-ième ligne (resp. de la j-ième colonne) de $\mathbf{M}$, et E est le nombre d'éléments non-nuls de $\mathbf{M}$. Par conséquent, pour une matrice complètement générique, le théorème de Tellegen ne présente aucun intérêt. Par contre, si la matrice $\mathbf{M}$ admet une structure, il est concevable que l'on dispose d'un algorithme plus rapide que l'algorithme naïf (quadratique) pour la multiplier par un vecteur de $\mathbb{K}^n$. En utilisant le principe de transposition, on obtient aussitôt un algorithme rapide pour multiplier ${}^t\mathbf{M}$ par un vecteur de $\mathbb{K}^m$.

Exemple 1. Considérons l'exemple suivant, qui illustre ce principe en utilisant la représentation par graphes des algorithmes linéaires. L'algorithme direct prend les valeurs x_1 et x_2 en entrée et renvoie $y_1 = ax_1 + bx_2$ et $y_2 = cx_1 + dx_2$ en sortie. Les arêtes représentent des multiplications par des valeurs scalaires a, b, c, d.



Transposer cet algorithme revient à inverser le flot du calcul, c'est-à-dire, inverser le sens des flèches, permuter les '+' avec les '•' et les entrées avec les sorties. L'algorithme ainsi obtenu prend y_1, y_2 en entrée et renvoie $x_1 = ay_1 + cy_2$ et $x_2 = by_1 + dy_2$. Il calcule donc bien l'application transposée de l'application initiale. De plus, le nombre d'opérations arithmétiques utilisées par les deux algorithmes est le même : 4 multiplications et 2 additions.



Utilité. Comme nous l'avons vu au Chapitre 11, beaucoup de problèmes en calcul formel s'expriment en termes d'algèbre linéaire structurée (multiplication par un vecteur ou résolution de système). Par exemple, les opérations élémentaires sur les polynômes (multiplication, division, évaluation-interpolation, interpolation rationnelle, extrapolation, etc.) sont *linéaires* si l'on fixe l'un des opérandes : ils se codent en termes de produits matrice-vecteur $\mathbf{M} \cdot \mathbf{v}$ ou $\mathbf{M}^{-1} \cdot \mathbf{v}$, où $\mathbf{M}$ est une matrice structurée (de type Toeplitz, Hankel, compagnon, Vandermonde, Cauchy,...)

Grâce au principe de Tellegen, comprendre, analyser et améliorer un algorithme se ramène à comprendre, analyser et améliorer son transposé. Deux sont ses utilités principales : *trouver* des solutions algorithmiques de meilleure complexité, et *clarifier* le statut de certains algorithmes existant dans la littérature, qui parfois sont simplement des transposés d'algorithmes bien connus. Cela permet ainsi le traitement algorithmique unifié des problèmes duaux. On peut résumer en disant que le principe de transposition permet de diviser par deux le nombre d'algorithmes linéaires qu'il reste à découvrir.

Un exemple moins trivial. Considérons le problème de l'évaluation d'un polynôme $P \in \mathbb{K}[X]$ de degré n en une valeur $a \in \mathbb{K}$. C'est une opération linéaire en les coefficients de P, de matrice $\mathbf{M} = [1, a, \ldots, a^n]$ dans les bases canoniques. Le problème transposé est donc le suivant : pour une valeur donnée $x_0 \in \mathbb{K}$, calculer les produits $a^i x_0$, pour $0 \leq i \leq n$. Pour ce problème, un algorithme naturel consiste à multiplier x_0 par a, ensuite multiplier le résultat par a, et ainsi de suite.

Le transposé de cet algorithme s'obtient en parcourant l'algorithme direct en sens inverse, tout en permutant les entrées avec les sorties, et en remplaçant chaque instruction par sa *transposée*, obtenue en appliquant un nombre restreint de règles syntaxiques. Dans ce processus, les boucles `for` montantes deviennent des boucles `for` descendantes.

De cette manière, on obtient *automatiquement* un algorithme pour le problème de départ, à savoir, l'évaluation de P sur a. Dans notre cas, il s'avère que l'algorithme transposé coïncide avec la fameuse *méthode de Horner*, voir la Figure 1. Observons que l'algorithme transposé utilise n opérations de plus que l'algorithme direct. Cette perte s'explique par le théorème de Tellegen : il s'agit tout simplement de la différence entre le nombre de colonnes et le nombre de lignes de $\mathbf{M}$. Cette observation peut être utilisée pour expliquer l'optimalité de la règle de Horner.

$$\mathbf{M} = [1, a, \ldots, a^n]$$

$x_0 \mapsto [x_0, ax_0, \ldots, a^n x_0]$	$[p_0, \ldots, p_n] \mapsto \sum_{i=0}^{n} p_i a^i$
Entrée x_0.	**Entrée** $p = [p_0, \ldots p_n]$.
$p_0 \leftarrow x_0$;	**for** j **from** n **downto** 1 **do**
for j **from** 1 **to** n **do**	$p_j \leftarrow a p_j$;
$p_j \leftarrow p_{j-1}$;	$p_{j-1} \leftarrow p_j + p_{j-1}$;
$p_j \leftarrow a p_j$;	$x_0 \leftarrow p_0$;
Sortie $p = [p_0, \ldots, p_n]$.	**Sortie** x_0.

Figure 1: Le schéma de Horner (à droite) obtenu en transposant l'algorithme qui résout le problème dual.

2. La version en termes de graphes du principe de Tellegen

Dans cette section nous donnons une version du théorème de Tellegen dans un modèle particulier, celui des graphes de calcul (DAG).

Définition 1. Un *graphe acyclique orienté,* ou DAG (de l'anglais *directed acyclic graph*) est un graphe orienté $\mathsf{G} = (V, E)$ qui ne possède pas de cycle. Un DAG $\mathbb{K}$-linéaire est un DAG muni d'une fonction de poids $\lambda : E \to \mathbb{K}$.

Soit $I = \{x_1, \ldots, x_n\}$ l'ensemble des nœuds d'entrée de G. À tout sommet $v \in V$ on associe une forme linéaire dans $\mathbb{K}[X_1, \ldots, X_n]$ de la façon suivante :
- si $v = x_i \in I$, alors $h_v := X_i$,
- si $v \in V \setminus I$, alors $h_v := \displaystyle\sum_{e=(w,v)\in E} \lambda(e)\, h_w$.

Définition 2. On dit que G calcule la matrice $\mathbf{M} = [a_{i,j}]$ de taille $m \times n$ si les formes linéaires associées aux nœuds de sortie sont $F_i = \sum_{j=1}^{n} a_{i,j} X_j$, $i = 1, \ldots, m$. Le coût de G est le nombre d'opérations linéaires $c(\mathsf{G})$ induites par G.

Avec ces notations, le principe de Tellegen s'énonce comme suit.

Théorème 1. *Soit G un $\mathbb{K}$-DAG qui calcule une matrice $\mathbf{M}$ de $\mathcal{M}_{m,n}(\mathbb{K})$. Alors le graphe transposé ${}^t\mathsf{G}$, obtenu en inversant le sens des flèches sans changer leur poids, calcule la matrice ${}^t\mathbf{M}$. De plus,*

$$c({}^t\mathsf{G}) = c(\mathsf{G}) - n + m.$$

Esquisse de preuve. La forme linéaire calculée par un sommet v de G est

$$h_v = \sum_{j=1}^{n} \left(\sum_{p\in \text{ Chemin}(x_j, v)} \lambda(p) \right) X_j, \quad \text{où} \quad \lambda(p) = \prod_{e \text{ arête de } p} \lambda(e).$$

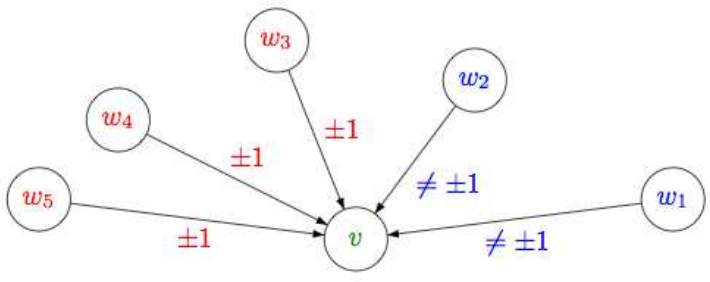
On a donc l'égalité $a_{i,j} = \displaystyle\sum_{p\in \text{ Chemin}(x_j, F_i)} \lambda(p)$, qui montre que ${}^t\mathsf{G}$ calcule bien ${}^t\mathbf{M}$.

L'égalité entre les coûts se déduit du fait que la quantité $c(\mathsf{G}) - |I(\mathsf{G})|$ ne dépend pas de l'orientation de G, comme montré par le lemme suivant. □

Lemme 1 (formule pour le coût d'un graphe de calcul)**.** *Avec les notations précédentes, on a l'égalité*

$$c(\mathsf{G}) = \big|\{e \in E \mid \lambda(e) \neq \pm 1\}\big| + |E| - |V| + |I|.$$

Esquisse de preuve. Tout sommet v qui reçoit $d(v) > 0$ arêtes contribue au coût de G avec $\big|\{e = (w, v) \in E \mid \lambda(e) \neq \pm 1\}\big| + d(v) - 1$ opérations dans $\mathbb{K}$.
Graphiquement, cela se voit sur la représentation suivante :



La conclusion se déduit alors aisément de la suite d'égalités

$$\sum_{v\in V\setminus I} \big(d(v) - 1\big) = \sum_{v\in V\setminus I} d(v) - \sum_{v\in V\setminus I} 1 = |E| - (|V| - |I|).$$

□

3. Principe de Tellegen pour les programmes linéaires

Le but de cette section est de décrire le principe de Tellegen dans le modèle des programmes linéaires sur des machines à allocation de registres (machines RAM).

3.1. **Machines à registres.** Commençons par définir les machines RAM à instructions linéaires.

Définition 3. On fixe un entier M ; il représente la mémoire disponible, de sorte que l'on peut accéder à M registres $R_1, \dots, R_M$. On y stocke des nombres, c'est-à-dire des éléments du corps $\mathbb{K}$.

Un programme linéaire est la donnée des objets suivants :
- un sous-ensemble $R_{i_1}, \dots, R_{i_n}$ des registres que l'on appelle *entrée* ;
- un sous-ensemble $R_{o_1}, \dots, R_{o_m}$ des registres que l'on appelle *sortie* ;
- une suite d'instructions portant sur les registres, choisies parmi :
  - $R_i = \pm R_j \pm R_k$, avec $1 \leq i, j, k \leq M$,
  - $R_i = \lambda R_j$, avec $1 \leq i, j \leq M$ et λ dans $\mathbb{K}$.

Le fonctionnement d'un tel programme est le suivant : à l'initialisation, les registres d'entrée reçoivent des valeurs $x_1, \dots, x_n \in \mathbb{K}$ et les autres sont mis à zéro. On effectue ensuite toutes les instructions, puis le programme renvoie les valeurs des registres de sortie. Noter que l'on calcule bel et bien une fonction linéaire des x_i.

3.2. **Transposition de programme.** Nous décrivons maintenant le principe de transposition des programmes linéaires sur une machine RAM.

Cas d'un jeu d'instructions réduit. Pour commencer, on traite le cas d'une machine avec un jeu d'instructions limité par rapport au cas général. On ne s'autorise que les instructions du type :
- $R_i = R_i \pm R_j$, avec $1 \leq i, j \leq M$;
- $R_i = \lambda R_i$, avec $1 \leq i \leq M, \lambda \in \mathbb{K}$.

Pour « transposer » un programme comportant des instructions de ce type, on interprète chacune de ces intructions comme une application linéaire $\mathbb{K}^M \to \mathbb{K}^M$.

Pour motiver ce procédé, considérons un exemple. Avec $M = 3$, l'instruction $R_1 = R_1 + R_3$ s'interprète comme l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix};$$

R_2 et R_3 n'ont pas changé et R_1 devient $R_1 + R_3$. La transposée de cette application linéaire a pour matrice

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix};$$

on en déduit que l'on peut la traduire par l'instruction $R_3 = R_3 + R_1$.

De manière générale, la transposée de l'instruction $R_i = R_i + R_j$ est l'instruction $R_j = R_j + R_i$. Par écriture matricielle, on voit que l'instruction $R_i = \lambda R_i$ est inchangée par transposition, et que $R_i = R_i - R_j$ se transpose en $R_j = R_j - R_i$. On peut alors définir le programme transposé d'un programme.

Définition 4. Le programme transposé ${}^t\mathcal{P}$ du programme linéaire $\mathcal{P}$ est donné par :
- les entrées de ${}^t\mathcal{P}$ sont les sorties de $\mathcal{P}$;
- les sorties de ${}^t\mathcal{P}$ sont les entrées de $\mathcal{P}$;
- les instructions de ${}^t\mathcal{P}$ sont les transposées des instructions de $\mathcal{P}$, prises en sens inverse.

La dernière condition, le retournement de la liste des instructions, traduit l'égalité matricielle ${}^t(AB) = {}^tB \cdot {}^tA$. La définition montre que si $\mathcal{P}$ calcule une application linéaire $\mathbb{K}^n \to \mathbb{K}^m$, alors ${}^t\mathcal{P}$ calcule bien l'application transposée $\mathbb{K}^m \to \mathbb{K}^n$.

Exemple 2. Considérons le programme

```
R4=R4+R2
R3=3*R3
R4=R4+R3
R2=2*R2
R3=R3+R2
R3=R3+R1
```

dont les entrées sont R_1, R_2, R_3 et les sorties R_3, R_4. Ce programme calcule l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 3 \end{bmatrix}.$$

Le programme transposé s'écrit

```
R1=R1+R3
R2=R2+R3
R2=2*R2
R3=R3+R4
R3=3*R3
R2=R2+R4
```

et on vérifie qu'il calcule l'application linéaire dont la matrice est

$$\begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 3 & 3 \end{bmatrix}.$$

Cas général. Le cas du jeu d'instructions général se ramène au cas du jeu d'instructions réduit de manière immédiate. Ainsi, l'instruction $R_1 = R_2 + R_3$ est équivalente à la suite d'instructions

```
R1=0
R1=R1+R2
R1=R1+R3
```

Il est donc possible de le transposer sous la forme

```
R3=R3+R1
R2=R2+R1
R1=0
```

De même, on réécrit aisément l'instruction $R_i = \lambda R_j$ en utilisant le jeu d'instructions réduit, ce qui permet de la transposer, ...

3.3. **Optimisations.** Notre principe de transposition n'est pas complètement satisfaisant : au vu des règles de réécriture ci-dessus, il semble que l'on puisse perdre jusqu'à un facteur 3 (en termes de nombre de lignes). On peut optimiser le code produit, afin de regagner, autant que possible, les lignes perdues. On utilise pour ce faire les règles suivantes :

Suppression des zéros. La transformation pour passer du cas général au jeu d'instructions réduit introduit des lignes du type $R_i = 0$. On peut supprimer une telle ligne, à condition de réécrire les instructions suivantes en conséquence.

Suppression des recopies. Après avoir supprimé les zéros, il se peut qu'il reste des lignes du type $R_i = \pm R_j$. On peut également supprimer ce type de ligne, à condition de réécrire de manière judicieuse les instructions qui suivent.

Muni de ces règles supplémentaires, on peut montrer qu'il est possible d'obtenir un code transposé qui fait exactement le même nombre de lignes que l'original mais cela dépasse le cadre de ce cours.

4. Applications

Dans cette section, nous illustrons l'utilité pratique du principe de Tellegen à travers quelques exemples. Les techniques de transposition permettront d'engendrer des programmes linéaires pour effectuer les opérations suivantes sur les polynômes à une variable : multiplication, division euclidienne, évaluation et interpolation multipoint.

La figure 2 contient une liste (non-exhaustive) d'opérations linéaires de base sur les polynômes à une variable et leurs problèmes transposés. Tout algorithme résolvant un problème de la colonne de gauche peut être transposé en un algorithme de même complexité pour le problème correspondant de la colonne de droite.

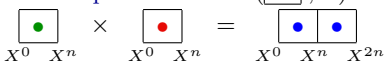
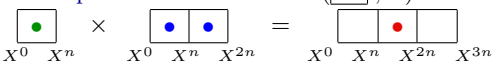
problème direct	problème transposé
multiplication mul $(\bullet, \ )$	produit médian $\mathrm{mul}^{\mathbf{t}}$ $(\bullet, \ )$
	
division euclidienne	extension de récurrences
$A \mapsto A \mod P$	$(a_0, \dots, a_{n-1}) \mapsto (a_0, \dots, a_{2n-1})$
évaluation multipoint	sommes de Newton pondérées
$P \mapsto (P(a_0), \dots, P(a_{n-1}))$	$(p_0, \dots, p_{n-1}) \mapsto (\sum p_i, \dots, \sum p_i a_i^{n-1})$
interpolation	décomposition en éléments simples
(systèmes de Vandermonde)	(systèmes de Vandermonde transposés)
décalage de polynômes	évaluation dans les factorielles descendantes
$P(X) \mapsto P(X+1)$	$P = \sum a_i X^{\underline{i}} \mapsto (P(0), \dots, P(n-1))$
extrapolation sur $0, 1, 2, \dots$	division modulo $(X-1)^n$
q-décalage	évaluation dans les q-factorielles descendantes
composition modulaire	projection des puissances
$\dots$	$\dots$

Figure 2: « Dictionnaire de Tellegen » pour les polynômes univariés.

4.1. **Produit médian des polynômes.** Commençons par transposer la multiplication des polynômes. Cette opération n'est pas linéaire, mais elle le devient lorsqu'on fixe l'un des deux opérandes. Fixons une fois pour toutes un polynôme f dans $\mathbb{K}[X]$ de degré m. Pour $n \geq 0$ quelconque, considérons l'application de multiplication $m_{f,n} : \mathbb{K}[X]_{\leq n} \to \mathbb{K}[X]_{\leq m+n}$ qui à un polynôme g associe le produit fg. Il est aisé de voir que la transposée de cette opération ${}^t m_{f,n} : \mathbb{K}[X]_{\leq m+n} \to \mathbb{K}[X]_{\leq n}$ consiste à extraire les coefficients de $X^m, X^{m+1}, \dots, X^{m+n}$ du produit d'un polynôme de degré au plus $m+n$ par le polynôme réciproque $\tilde{f}$ de f.

Exemple 3. Par exemple, soit $f = 2 + X + 3X^2$; la matrice de $m_{f,2}$ dans les bases canoniques de $\mathbb{K}[X]_{\leq 2}$ et $\mathbb{K}[X]_{\leq 4}$ est donnée par la matrice de type Toeplitz

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 3 & 1 & 2 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix}.$$

Sa transposée est

$$\begin{bmatrix} 2 & 1 & 3 & 0 & 0 \\ 0 & 2 & 1 & 3 & 0 \\ 0 & 0 & 2 & 1 & 3 \end{bmatrix}.$$

Cette dernière matrice est la partie médiane de la matrice de Toeplitz suivante

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ \mathbf{2} & \mathbf{1} & \mathbf{3} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{3} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{3} \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix},$$

qui représente le produit du polynôme $\tilde{f} = 3 + X + 2X^2$ par un polynôme h de degré 4. Cette partie médiane donne les coefficients de degré $2, 3, 4$ du produit $\tilde{f}h$. Cela explique pourquoi le produit transposé est appelé *produit médian*.

Le produit médian est une opération importante, qui intervient comme brique de base dans nombre d'algorithmes sur les polynômes et les séries. Son calcul efficace est un problème important à résoudre avant de passer à la transposition d'autres algorithmes plus sophistiqués.

Par exemple, l'utilisation du produit médian permet d'améliorer (d'un facteur constant) l'efficacité de l'opérateur de Newton pour l'inverse de séries formelles, et mène aux optimisations mentionnées en page 131. Cette remarque s'applique également aux autres opérations rapides sur les séries décrites au Chapitre 4. L'accélération se répercute aussi sur la division euclidienne rapide, qui repose sur une inversion de série (Théorème 1 du Chapitre 5).

En vertu du principe de Tellegen (Théorème 1) appliqué aux matrices de Toeplitz, le coût du produit médian est celui du produit fg, à m opérations près. En particulier, si $m = n$, cela permet d'effectuer le calcul de la partie médiane pour le coût d'une multiplication en degré n, au lieu des deux qu'on attendrait naïvement. Il y a plusieurs algorithmes pour multiplier des polynômes, à chacun correspond donc un algorithme de même complexité pour calculer le produit transposé ; cet algorithme transposé peut être obtenu en appliquant le principe de Tellegen, soit dans sa version graphes, soit par transformation de programmes linéaires. Plus exactement, on a le résultat général suivant.

Théorème 2. *Tout algorithme de complexité arithmétique* $\mathsf{M}(n)$ *pour la multiplication polynomiale en degré* n *peut être transformé en un algorithme de complexité arithmétique* $\mathsf{M}(n) + O(n)$ *pour le produit médian en degrés* $(n, 2n)$.

Exercice 1. Vérifier le théorème dans le cas de la multiplication naïve des polynômes. Expliciter l'algorithme transposé dans ce cas.

Exemple 4. En Figure 3, à gauche, est représenté un DAG qui calcule le produit de deux polynômes de degré 1 *à la Karatsuba* (en utilisant 3 multiplications au lieu de 4). Par transposition, on déduit un algorithme *à la Karatsuba* pour le produit transposé de deux polynômes de degré 1 et 2. L'algorithme direct utilise 6 opérations arithmétiques (3 additions et 3 multiplications) ; l'algorithme transposé en utilise 7 (4 additions et 3 multiplications). La différence de 1 opération est bien sûr prédite par le Théorème 1.

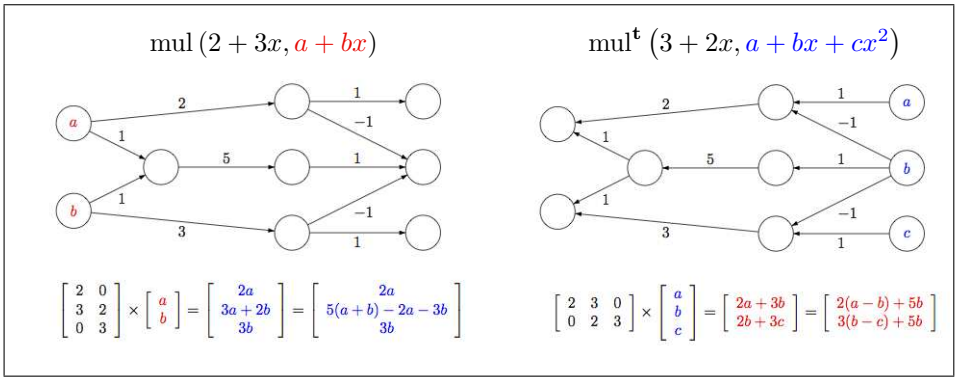


FIGURE 3: Produit direct et transposé *à la Karatsuba*, version DAG.

L'algorithme de Karatsuba et son code transposé sont donnés en Figure 4. Tous les deux ont la même complexité $O(n^{\log(3)})$.

Exemple 5. La matrice de la DFT étant symétrique, à tout algorithme (linéaire) qui la calcule est associé un algorithme transposé de même coût qui la calcule également. Par exemple, à la variante de la DFT décrite au Chapitre 2 (due à Gentleman et Sande) correspond un algorithme de Cooley et Tukey. En figure 5 sont représentés les DAG de ces algorithmes pour une DFT sur 4 points.

4.2. **Division avec reste et extension des récurrences à coefficients constants.** On peut montrer que le dual du problème de la division avec reste d'un polynôme de degré N par un polynôme fixé de degré n est l'extension des récurrences linéaires à coefficients constants. Étant donnés les n premiers termes d'une suite qui vérifie une récurrence linéaire à coefficients constants d'ordre n, il s'agit de calculer la tranche des N termes suivants.

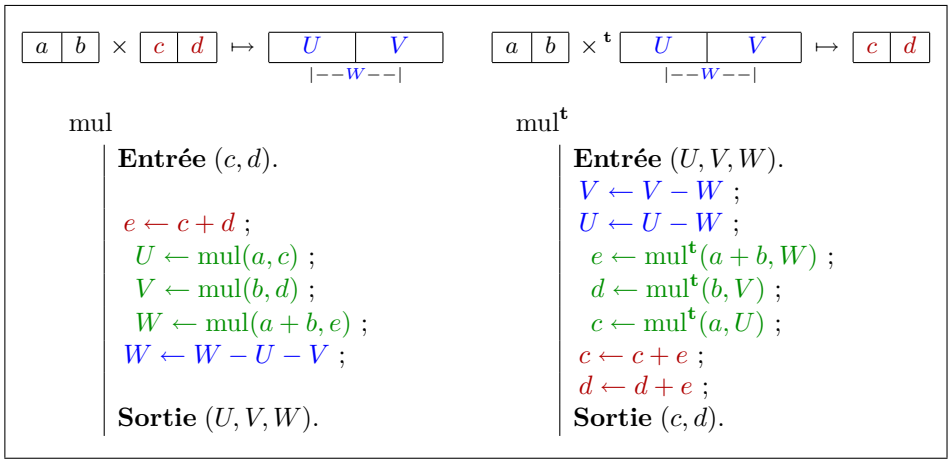


Figure 4: Algorithme de Karatsuba et son transposé.

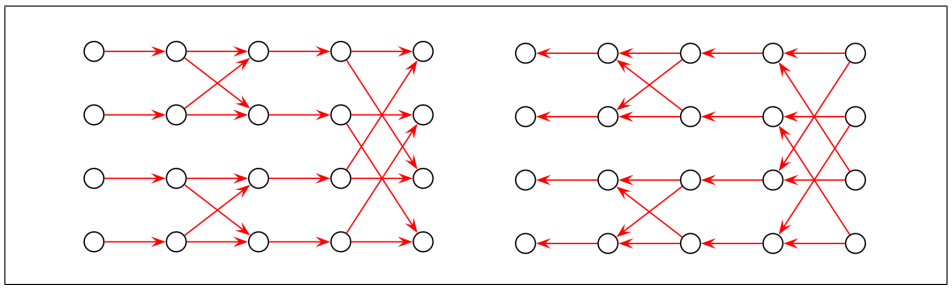


Figure 5: Dualité entre deux classes d'algorithmes pour la DFT : à gauche, l'algorithme *decimation-in-time* de Cooley–Tukey, à droite l'algorithme *decimation-in-frequency* de Gentleman–Sande.

Pour une récurrence d'ordre $n = 1$, la preuve est immédiate : étendre une récurrence $u_{n+1} = au_n$, de condition initiale $u_0 = x_0$ revient à calculer la suite des valeurs $x_0, ax_0, \dots, a^N x_0$ à partir de x_0. Or, on a vu en page 214 que le dual de ce problème est l'évaluation polynomiale en a, autrement dit, la division avec reste modulo $X - a$.

Exercice 2. Finir la preuve dans le cas général (n quelconque).

Exercice 3. Transposer l'algorithme rapide de division euclidienne donné au Chapitre 5 dans la preuve du Théorème 1 (page 141). En déduire un algorithme de complexité $O(\mathsf{M}(n))$ permettant de calculer les $2n$ premiers termes d'une suite donnée par une récurrence linéaire d'ordre n à coefficients constants et n conditions initiales.

4.3. **Évaluation multipoint et interpolation.** Comme vu au Chapitre 6, l'évaluation multipoint se traduit en termes matriciels par un produit entre la matrice de Vandermonde associée aux points a_i et le vecteur des coefficients du polynôme P, voir la Figure 6. Ainsi, le problème transposé est la multiplication d'une matrice de Vandermonde transposée par un vecteur, c'est-à-dire, le calcul de sommes de Newton pondérées (on les appelle ainsi car si tous les p_i valent 1, on retrouve les sommes des puissances des a_i).

Il est possible d'exploiter algorithmiquement cette dualité, à l'aide du principe de transposition. L'idée est de proposer d'abord un algorithme rapide pour le calcul de ces sommes de Newton, et ensuite de le transposer.

Pour ce faire, le point de départ est l'observation que la série génératrice des sommes de Newton pondérées

$$\sum_{s\geq 0}\Big(\sum_{i=0}^{n-1} p_i a_i^s\Big)X^{-s-1}$$

$$\begin{bmatrix} 1 & a_0 & \cdots & a_0^{n-1} \\ 1 & a_1 & \cdots & a_1^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & a_{n-1} & \cdots & a_{n-1}^{n-1} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} P(a_0) \\ P(a_1) \\ \vdots \\ P(a_{n-1}) \end{bmatrix} \quad \text{et} \quad \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a_0 & a_1 & \cdots & a_{n-1} \\ \vdots & \vdots & & \vdots \\ a_0^{n-1} & a_1^{n-1} & \cdots & a_{n-1}^{n-1} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} \sum p_i \\ \sum a_i p_i \\ \vdots \\ \sum a_i^{n-1} p_i \end{bmatrix}.$$

FIGURE 6: L'évaluation multipoint et sa transposée, les sommes de Newton pondérées.

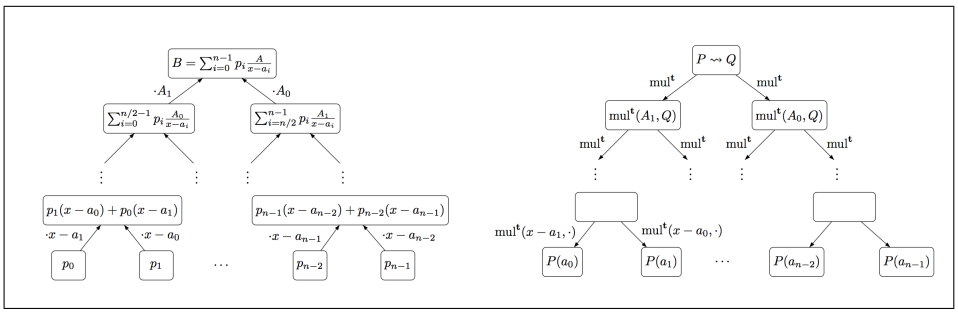


FIGURE 7: Algorithme rapide d'évaluation multipoint par produits médians répétés (à droite), obtenu par transposition d'un algorithme pour les sommes de Newton pondérées (à gauche).

est *rationelle* et vaut $Q = B/A$, où

$$A = (X - a_0) \cdots (X - a_{n-1}) \quad \text{et} \quad B = \sum_{i=0}^{n-1} p_i \frac{A}{X - a_i}.$$

D'où l'algorithme suivant : on calcule A et B et on retourne les n premiers coefficients du développement en série de Taylor à l'infini de $Q = B/A$. Le calcul de A se fait en utilisant l'algorithme ArbreSousProduits donné en Figure 1 (page 153), de complexité $\frac{1}{2}\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$. Celui de B peut se faire par l'algorithme de type « diviser pour régner » représenté en Figure 7 à gauche, de complexité $\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$. (Une solution équivalente serait de calculer simultanément A et B par l'algorithme SommesFractions en page 155). Enfin, le développement en série de Q s'effectue en $O(\mathsf{M}(n))$ opérations arithmétiques grâce à des itérations de Newton, comme décrit au Chapitre 4.

En résumé, on obtient un algorithme pour le calcul des sommes de Newton pondérées, de complexité

$$\frac{3}{2}\mathsf{M}(n)\log(n) + O\big(\mathsf{M}(n)\big).$$

Par transposition, on construit un algorithme *de même complexité* pour l'évaluation multipoint. Cet algorithme est représenté graphiquement en Figure 7, à droite. Tout comme l'algorithme EvaluationRapide représenté en Figure 2 (page 153), le nouvel algorithme repose sur une stratégie « diviser pour régner », mais remplace, à chaque niveau de récursion, les divisions avec reste par des produits médians, moins coûteux, d'où le gain d'un facteur constant dans la complexité.

Exercice 4. Écrire les détails des deux algorithmes en Figure 7.

Exercice 5. Soit $\mathbb{K}$ un corps et soit n un entier. On considère le problème suivant :

> Étant données les valeurs en $0, 1, \ldots, n-1$ d'un polynôme (inconnu) de $\mathbb{K}[X]$ de degré au plus n, calculer les valeurs prises par ce polynôme en $n, n+1, \ldots, 2n-1$.

Déterminer le problème dual, donner un algorithme de complexité $O(\mathsf{M}(n))$ pour ce dernier, et en déduire un algorithme explicite de complexité $O(\mathsf{M}(n))$ pour le problème de départ.

4.4. **Bonus : l'évaluation et l'interpolation ont des coûts équivalents.** Un dernier exemple d'application du théorème de Tellegen, de nature plus théorique, est la preuve du fait que les problèmes d'évaluation multipoint et d'interpolation polynomiale sont équivalents du point de vue de la complexité.

Théorème 3. *Tout algorithme qui effectue l'évaluation multipoint (resp. l'interpolation) sur une famille fixée de points peut être transformé en un algorithme d'interpolation (resp. d'évaluation multipoint) sur la même famille de points, de même complexité à un nombre constant de multiplications polynomiales près.*

Autrement dit, si $\mathsf{E}(n)$ *et* $\mathsf{I}(n)$ *représentent les complexités de l'évaluation multipoint et de l'interpolation en n points, alors :*

$$\mathsf{I}(n) \in O\big(\mathsf{E}(n) + \mathsf{M}(n)\big) \quad \textit{et} \quad \mathsf{E}(n) \in O\big(\mathsf{I}(n) + \mathsf{M}(n)\big).$$

Esquisse de preuve. Supposons qu'on dispose d'un algorithme $\mathcal{I}$ de complexité $\mathsf{I}(n)$ pour l'interpolation sur les points $\mathbf{a} = (a_0, \ldots, a_{n-1})$. Le théorème de Tellegen montre qu'on peut construire un algorithme ${}^t\mathcal{I}$ de complexité $\mathsf{I}(n)$ pour *l'interpolation transposée* sur $\mathbf{a}$. On va montrer comment en déduire un algorithme $\mathcal{E}$ pour l'évaluation multipoint sur $\mathbf{a}$. L'idée est d'utiliser la factorisation matricielle

$$\mathbf{V_a} = ({}^{\mathbf{t}}\mathbf{V_a})^{-1} \cdot \mathbf{H_a}, \quad \text{où} \quad \mathbf{H_a} = \begin{bmatrix} n & \sum_i a_i & \ldots & \sum_i a_i^{n-1} \\ \sum_i a_i & \sum_i a_i^2 & \ldots & \sum_i a_i^n \\ \vdots & \vdots & & \vdots \\ \sum_i a_i^{n-1} & \sum_i a_i^{n-1} & \ldots & \sum_i a_i^{2(n-1)} \end{bmatrix},$$

qui suggère l'algorithme $\mathcal{E}$ suivant :

- Appliquer $\mathcal{I}$ pour calculer $A(X) = \prod_i (X - a_i)$, en complexité $\mathsf{I}(n)$;
- calculer $\mathbf{H_a}$ à partir des $2n-1$ premières sommes de Newton de $A(X)$, par l'algorithme de complexité $O(\mathsf{M}(n))$ donné en Proposition 4, page 133 ;
- calculer $\mathbf{v} = \mathbf{H_a} \cdot \mathbf{p}$ en $O(\mathsf{M}(n))$ opérations, comme au Chapitre 11 ;
- retourner $\mathbf{V_a} \cdot \mathbf{p} = {}^t(\mathbf{V_a^{-1}}) \cdot \mathbf{v}$ en $\mathsf{I}(n)$ opérations en utilisant l'algorithme ${}^t\mathcal{I}$.

La preuve du sens inverse est laissée en exercice. □

Exercice 6. Finir la preuve du Théorème 3.

Notes

La présentation de la Section 2 est inspirée de [24]. L'algorithme rapide de l'exercice 3 pour l'extension de récurrences a été donné par Shoup dans [29, 31]. La dualité *division polynomiale* ↔ *extension de récurrences* présentée en Section 4.2 est implicite dans [12, §IV.3]. Elle permet de clarifier le statut de l'algorithme de Shoup : c'est la transposée de l'algorithme de Strassen [33] pour la division euclidienne des polynômes. Ce fait a été mis en évidence dans [9].

L'algorithme présenté en Section 4.3, améliorant (d'un facteur constant) les algorithmes classiques d'évaluation, est dû à [9]. Le principe de transposition est également employé dans [9] pour accélérer (toujours par des facteurs constants) les algorithmes classiques pour l'interpolation et pour son problème dual (la résolution de systèmes de Vandermonde transposés). Les mêmes méthodes s'étendent au cas plus général du théorème des restes chinois [7].

Le Théorème 3 est dû à [8]. La factorisation matricielle utilisée dans sa preuve est tirée de [11].

Une autre application récente du principe de transposition est la suivante [10] : si $(P_\ell)_{\ell \geq 0}$ est une famille de polynômes de $\mathbb{K}[X]$, avec $\deg(P_\ell) = \ell$, dont la série génératrice exponentielle $\sum_{\ell \geq 0} P_\ell(X)/\ell! \cdot Y^\ell$ est de la forme $v(Y) \cdot \exp(X \cdot h(Y))$ [1], alors les conversions entre la base canonique $(X^\ell)_{\ell \geq 0}$ de $\mathbb{K}[X]$ et la base $(P_\ell)_{\ell \geq 0}$ peuvent s'effectuer en $O(\mathsf{M}(n))$ opérations dès lors que $v \bmod Y^n$ peut se calculer en $O(\mathsf{M}(n))$ opérations, et pour une large classe de séries h.

Historique. Le principe de transposition a une histoire longue et tortueuse. Cette technique de transformation des algorithmes linéaires est issue du domaine des circuits électroniques [6, 27, 1] et de la théorie du contrôle [20]. Le principe même remonte aux années 1950 ; on en trouve les traces dans un article de Tellegen [34] sur les réseaux électriques. Il a été généralisé aux filtres digitaux par Fettweis [14], où l'on trouve une version embryonnaire de la version par graphes. Le théorème de Tellegen a été redémontré plusieurs fois, dans divers contextes et degrés de généralité, par Fiduccia [15, 16], Hopcroft et Musinski [19], Knuth et Papadimitriou [25], et par Kaminski, Kirkpatrick et Bshouty [24]. En calcul formel, il a été popularisé dans les travaux de Ben-Or, Tiwari [3], Kaltofen, Canny, Lakshman [11, 21], Shoup [29, 30, 31], Lecerf, Schost [26], Hanrot,

1. Une telle famille de polynômes est appelée *suite de Sheffer* [28].

Quercia, Zimmermann [18], Zippel [37], von zur Gathen et Gerhard [35], ... Il est resté longtemps méconnu, comme le montre cet extrait de l'article [36] de Wiedemann : "I was not able to find an example of a linear operator that is easy to apply but whose transpose is difficult to apply". Le nom *principe de transposition* semble avoir été introduit par Kaltofen et Shoup dans [23, 32]. Un point de vue légèrement différent sur les aspects historiques liés au principe de transposition peut être trouvé dans [4].

Dans la littérature du calcul formel, c'est surtout son caractère de théorème d'existence qui est exploité : connaissant un algorithme pour une certaine opération linéaire, on en déduit l'existence d'un algorithme de même complexité pour l'opération duale. Un fait mis en évidence plus récemment [9] est que la transposition des programmes peut se faire de manière systématique et (quasi-)automatique. Les algorithmes sont transposés directement, d'une manière similaire à celle utilisée en différentiation automatique [17], mais en tirant profit des spécificités linéaires. Par ailleurs, lorsqu'un problème linéaire doit être résolu efficacement, une démarche naturelle consiste à essayer de trouver un algorithme rapide résolvant le problème dual. Le cas échéant, une solution rapide pour le problème de départ est obtenue en re-transposant cet algorithme. C'est le point de vue adopté en Section 4.3.

Produit médian. Le concept de *produit médian* a été introduit dans l'article [18], qui souligne l'importance de cette *nouvelle opération* sur les polynômes.

En traitement du signal [5, 13], il était déjà connu que, si la multiplication est effectuée à base de DFT, le produit médian en degrés $(n, 2n)$ a la même complexité que le produit en degré n. En effet, il se code matriciellement en un produit matrice–vecteur par une matrice de Hankel de taille n ; or, celle-ci peut être plongée dans une matrice circulante de taille $2n$. Cela implique que dans le modèle de multiplication polynomiale par FFT, un produit médian $(n, 2n)$ peut être effectué en utilisant 3 DFT (deux directes et une inverse) de taille uniquement $2n$ (au lieu de $3n$).

Cette interprétation n'est plus utile dans le modèle de multiplication par l'algorithme de Karatsuba. L'article [18] est le premier à avoir proposé un algorithme de type Karatsuba pour le produit médian.

Lien avec la différentiation automatique. Il a été remarqué dans [11, 22] que le principe de transposition est lié au *mode inverse* en *différentiation automatique* pour le calcul du gradient d'une fonction. En effet, les éléments de ${}^t\mathbf{M}\cdot\mathbf{w}$ sont les dérivées partielles de ${}^t\mathbf{v}\cdot{}^t\mathbf{M}\cdot\mathbf{w}$ par rapport aux coordonnées de $\mathbf{v}$. Ce produit n'est autre que ${}^t\mathbf{w}\cdot\mathbf{M}\cdot\mathbf{v}$, et le théorème de Baur-Strassen [2] montre que l'on peut calculer ces dérivées partielles au prix d'un surcoût linéaire.

Lien avec la théorie de la complexité bilinéaire. Hopcroft et Musinski ont donné dans [19] une *version bilinéaire* de l'algorithme de Tellegen. Étant donné un algorithme bilinéaire qui utilise N multiplications pour le problème, noté (m, n, p), de la multiplication de deux matrices de tailles $m \times n$ et $n \times p$, il est possible d'engendrer cinq algorithmes duaux, chacun utilisant exactement N multiplications, pour les problèmes (n, m, p), (p, m, n), (m, p, n), (n, p, m) et (p, n, m). Ce résultat admet la conséquence utile suivante (mentionnée en page 123) : si l'on peut calculer en N multiplications dans $\mathbb{K}$ le produit de deux matrices quelconques sur $\mathbb{K}$ de tailles $m \times n$ et $n \times p$, alors $\omega \leq 3\ \log_{mnp}(N)$.

Dualité pour l'évaluation des monômes à plusieurs variables. Knuth et Papadimitriou [25] ont montré que si $\mathbf{M} = (a_{i,j})$ est une matrice carrée inversible dont les éléments sont des entiers strictement positifs, alors, partant de $\{X_1, \ldots, X_n\}$, le nombre minimum de multiplications pour évaluer les monômes

$$\left\{X_1^{a_{11}} X_2^{a_{12}} \cdots X_n^{a_{1n}},\ X_1^{a_{21}} X_2^{a_{22}} \cdots X_n^{a_{2n}}, \ldots,\ X_1^{a_{n1}} X_2^{a_{n2}} \cdots X_n^{a_{nn}}\right\}$$

est le même que le nombre minimum de multiplications pour évaluer les monômes

$$\left\{X_1^{a_{11}} X_2^{a_{21}} \cdots X_n^{a_{n1}},\ X_1^{a_{12}} X_2^{a_{22}} \cdots X_n^{a_{n2}}, \ldots,\ X_1^{a_{1n}} X_2^{a_{2n}} \cdots X_n^{a_{nn}}\right\}.$$

Cet énoncé peut être vu comme cas particulier du théorème de Tellegen.

Lien entre les algorithmes de Keller-Gehrig et de Storjohann. L'itération de Keller-Gehrig (Chapitre 3, page 118) permettant le calcul de la suite $\mathsf{s} = [x, ax, a^2x, \ldots, a^Nx]$ $(N = 2^k - 1)$ est la transposée de l'algorithme « diviser pour régner » pour d'évaluation d'un polynôme $P(x) = a_0 + \cdots + a_N x^N$ en a via la décomposition de type *decimation-in-time* $P(x) = P_0(x^2) + x \cdot P_1(x^2)$. L'algorithme « diviser pour régner » pour l'évaluation de P en a via la décomposition de type

decimation-in-frequency $P(x) = P_0(x) + x^{\frac{N+1}{2}} P_1(x)$ admet comme dual l'algorithme de Storjohann (Chapitre 14, page 237) pour calculer `s`.

Bibliographie

[1] Antoniou (A.). – *Digital Filters : Analysis and Design*. – McGraw-Hill Book Co., 1979.

[2] Baur (W.) and Strassen (V.). – The complexity of partial derivatives. *Theoretical Computer Science*, vol. 22, 1983, pp. 317–330.

[3] Ben-Or (M.) and Tiwari (P.). – A deterministic algorithm for sparse multivariate polynomial interpolation. In *STOC'88*. pp. 301–309. – ACM Press, 1988.

[4] Bernstein (D. J.). – The transposition principle. – `http://cr.yp.to/transposition.html`.

[5] Bluestein (Leo I.). – A linear filtering approach to the computation of the discrete Fourier transform. *IEEE Northeast Electronics Research and Engineering Meeting*, vol. 10, 1968, pp. 218–219.

[6] Bordewijk (J. L.). – Inter-reciprocity applied to electrical networks. *Appl. Sci. Res. B.*, vol. 6, 1956, pp. 1–74.

[7] Bostan (A.), Lecerf (G.), Salvy (B.), Schost (É.), and Wiebelt (B.). – Complexity issues in bivariate polynomial factorization. In *ISSAC'04*, pp. 42–49. – ACM, New York, 2004.

[8] Bostan (A.) and Schost (É.). – On the complexities of multipoint evaluation and interpolation. *Theoret. Comput. Sci.*, vol. 329, n1–3, 2004, pp. 223–235.

[9] Bostan (Alin), Lecerf (Grégoire), and Schost (Éric). – Tellegen's principle into practice. In Sendra (J. R.) (editor), *ISSAC'03*. pp. 37–44. – ACM Press, 2003.

[10] Bostan (Alin), Salvy (Bruno), and Schost (Éric). – Power series composition and change of basis. In *ISSAC'08*, pp. 269–276. – ACM, New York, 2008.

[11] Canny (J.), Kaltofen (E.), and Yagati (L.). – Solving systems of non-linear polynomial equations faster. In *ISSAC'89*. pp. 121–128. – ACM Press, 1989.

[12] Cerlienco (L.), Mignotte (M.), and Piras (F.). – Suites récurrentes linéaires. Propriétés algébriques et arithmétiques. *L'Enseignement Mathématique*, vol. 33, 1987, pp. 67–108.

[13] Chu (Eleanor) and George (Alan). – *Inside the FFT black box*. – CRC Press, 2000, xxii+312p. Serial and parallel fast Fourier transform algorithms.

[14] Fettweis (A.). – A general theorem for signal-flow networks, with applications. *Archiv für Elektronik und Übertragungstechnik*, vol. 25, n12, 1971, pp. 557–561.

[15] Fiduccia (C. M.). – On obtaining upper bounds on the complexity of matrix multiplication. In *Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)*, pp. 31–40. – Plenum, New York, 1972.

[16] Fiduccia (C. M.). – *On the algebraic complexity of matrix multiplication*. – PhD thesis, Brown Univ., Providence, RI, Center Comput. Inform. Sci., Div. Engin., 1973.

[17] Gilbert (J.-C.), Le Vey (G.), and Masse (J.). – *La différentiation automatique de fonctions représentées par des programmes*. – Technical report, RR INRIA 1557, 1991.

[18] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, vol. 14, n6, March 2004, pp. 415–438.

[19] Hopcroft (J.) and Musinski (J.). – Duality applied to the complexity of matrix multiplication and other bilinear forms. *SIAM Journal on Computing*, vol. 2, 1973, pp. 159–173.

[20] Kalman (R. E.). – On the general theory of control systems. *IRE Transactions on Automatic Control*, vol. 4, n3, 1959, pp. 481–491.

[21] Kaltofen (E.), Corless (R. M.), and Jeffrey (D. J.). – Challenges of symbolic computation : my favorite open problems. *Journal of Symbolic Computation*, vol. 29, n6, 2000, pp. 891–919.

[22] Kaltofen (Erich). – Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. In *Applied algebra, algebraic algorithms and error-correcting codes*, pp. 195–212. – Springer, Berlin, 1993.

[23] Kaltofen (Erich). – Computational differentiation and algebraic complexity theory. In *Workshop Report on First Theory Institute on Computational Differentiation*, pp. 28–30. – 1993.

[24] Kaminski (M.), Kirkpatrick (D. G.), and Bshouty (N. H.). – Addition requirements for matrix and transposed matrix products. *Journal of Algorithms*, vol. 9, n3, 1988, pp. 354–364.

[25] Knuth (Donald E.) and Papadimitriou (Christos H.). – Duality in addition chains. *Bulletin of the European Association for Theoretical Computer Science*, vol. 13, 1981, pp. 2–4.

[26] Lecerf (G.) and Schost (É.). – Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, vol. 5, n1, 2003, pp. 1–10.

[27] Penfield, Jr. (P.), Spence (R.), and Duinker (S.). – *Tellegen's theorem and electrical networks*. – The M.I.T. Press, Cambridge, Mass.-London, 1970, xv+143p.

[28] Roman (Steven). – *The umbral calculus*. – Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1984, *Pure and Applied Mathematics*, vol. 111, x+193p.

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

[30] Shoup (V.). – A new polynomial factorization algorithm and its implementation. *Journal of Symbolic Computation*, vol. 20, n4, 1995, pp. 363–397.

[31] Shoup (V.). – Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *ISSAC'99*. pp. 53–58. – ACM Press, New York, 1999.

[32] Shoup (Victor). – Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, vol. 17, n5, 1994, pp. 371–391.

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

[34] Tellegen (B.). – A general network theorem, with applications. *Philips Research Reports*, vol. 7, 1952, pp. 259–269.

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

[36] Wiedemann (D.). – Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, vol. IT-32, 1986, pp. 54–62.

[37] Zippel (R.). – Interpolating polynomials from their values. *Journal of Symbolic Computation*, vol. 9, n3, 1990, pp. 375–403.

CHAPITRE 13

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

Résumé

Pour calculer tous les N premiers termes d'une suite P-récursive, l'algorithme direct par déroulement de la récurrence est quasi-optimal vis-à-vis de N : sa complexité arithmétique (resp. binaire) est linéaire en N (resp. quasi-quadratique en N). Le N-ième terme d'une telle suite peut être calculé plus rapidement que l'ensemble des N premiers termes, en essentiellement $\sqrt{N}$ opérations arithmétiques et N opérations binaires.

Le Chapitre 5 a montré comment calculer le N-ième terme d'une suite donnée par une récurrence à coefficients constants en complexité arithmétique $O(\log N)$. Dans ce chapitre, nous nous attaquons au cadre plus général des récurrences à coefficients polynomiaux.

Le calcul du N-ième terme ou des N premiers termes [1] d'une suite P-récursive, donnée par la récurrence à coefficients polynomiaux

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

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, dans un algorithme de factorisation déterministe d'entiers, ou encore en cryptologie, dans des questions de comptage de points sur des courbes.

Si les coefficients $p_i(n)$ de la récurrence (1) ont degré au plus d en n, l'algorithme naïf par déroulement de la récurrence a une complexité arithmétique en $O(rdN)$ et une complexité binaire en $O\big(rN^2\mathsf{M}_{\mathbb{Z}}(d\log N)\big)$. Ces complexités sont quasi-optimales vis-à-vis de N pour le calcul de tous les N premiers termes.

Pour le calcul du N-ième terme seul, les algorithmes présentés dans ce chapitre n'ont pas une aussi bonne complexité que pour des coefficients constants, mais cette fois encore, ils sont 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. Quant à l'algorithme par *scindage binaire* de la section 3, il n'abaisse en rien la complexité arithmétique, et améliore pourtant la complexité binaire jusqu'à la rendre quasi-linéaire en la taille de sa sortie.

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

1.1. Cas de la factorielle.

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 $k+1$, soit donc un entier de taille $\log k! \sim k\log k$ avec un entier de taille $\log(k+1) \sim \log k$. Ce produit déséquilibré coûte donc moins de $ck\mathsf{M}_{\mathbb{Z}}(\log k)$ opérations

1. Dans la suite on fait l'hypothèse que le coefficient de tête p_r de la récurrence (1) ne s'annule sur aucun des entiers $0, \ldots, N-r$, où N est l'indice du terme maximal que l'on souhaite calculer.

binaires pour une certaine constante c. Le calcul complet de $N!$ par la méthode naïve effectue donc moins de

$$\sum_{k=1}^{N} ck\mathsf{M}_{\mathbb{Z}}(\log k) = O\big(N^2\mathsf{M}_{\mathbb{Z}}(\log N)\big)$$

opérations binaires.

La formule de Stirling montre que ces complexités arithmétique et binaire sont quasi-optimales pour le calcul conjoint des nombres 1!, 2!, …, $N!$.

1.2. **Cas général.** Pour une suite P-récursive donnée par une récurrence de la forme (1), le calcul par l'algorithme direct de u_{n+r} à partir des r valeurs précédentes 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 naïf de $u_0, \ldots, u_N$ à partir des valeurs initiales u_0, …, u_{r-1} demande donc $O(rdN)$ opérations arithmétiques.

Une légère amélioration de cette borne vient de l'observation que les coefficients $p_i(n)$ peuvent être évalués sur les entiers $n = 0, 1, \ldots, N$ en utilisant des techniques d'évaluation multipoint rapide, avant de procéder au déroulement de (1). Cela permet d'abaisser la complexité arithmétique du calcul de $O(rdN)$ à $O(r\mathsf{M}(N+d)\log(N+d))$, ce qui devient $\tilde{O}(r(N+d))$ si la multiplication polynomiale est à base de FFT.

Lorsque les coefficients des polynômes p_i sont entiers, la complexité binaire de l'algorithme naïf découle de nouveau essentiellement de la croissance de u_N. Pour estimer celle-ci, il est agréable d'adopter une vision matricielle et de faire apparaître le rationnel u_N comme un quotient d'entiers. Introduisons la suite vectorielle des $U_n = {}^t(u_n, \ldots, u_{n+r-1})$, qui vérifie la récurrence du premier ordre

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

Considérons aussi la suite hypergéométrique (w_n) donnée par la condition initiale $w_0 = 1$ et la récurrence

$$w_{n+1} = p_r(n) w_n, \qquad (n \in \mathbb{N}),$$

qui permet de récrire U_n sous la forme

$$U_n = \frac{1}{w_n}\big(A(n-1)\cdots A(0)\big)U_0.$$

Le rationnel u_N s'écrit alors $u_N = v_N/w_N$, où v_N est le premier coefficient du vecteur $\big(A(N-1)\cdots A(0)\big)U_0$.

Pour estimer la croissance de v_N, introduisons pour un vecteur U de taille r et une matrice M de taille $r \times r$ la norme d'indice 1 et sa norme subordonnée

$$\|U\| = \sum_{j=1}^{r} |v_j| \qquad \text{et} \qquad \|M\| = \max_{1\le j\le r} \sum_{i=1}^{r} |m_{i,j}|.$$

Soit ℓ un majorant commun pour la taille binaire des coefficients des p_i, $0 \le i \le r$, et des conditions initiales u_j, $0 \le j < r$. Il s'ensuit que la norme de $A(n)$ est bornée par $r\,2^{\ell}(d+1)(n+1)^d$ et celle de U_0 par $r2^{\ell}$. Les majorations

$$|v_N| \le \|A(N-1)\| \cdots \|A(0)\|\,\|U_0\| \le r^{N+1}\,2^{\ell(N+1)}(d+1)^N (N!)^d$$

fournissent la borne $O\big(dN\log N + N\ell + N\log r\big)$ sur la taille $\log|v_N|$. Par le même raisonnement, la taille du dénominateur w_N est du même ordre (mêmes formules pour $r = 1$), si bien que par le même type d'argument que pour la factorielle, la complexité binaire du calcul naïf de u_N est $O\big(rN^2\mathsf{M}_{\mathbb{Z}}(d\log N + \ell + \log r)\big)$.

2. Pas de bébés et pas de géants

On a vu au Chapitre 5 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 $\log N$ (il est fort possible qu'un tel algorithme n'existe pas). 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 $n \geq 0$. Une solution efficace exploite le théorème « évaluation interpolation » (théorème 1 du Chapitre 6). 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

$$P(X) = (X+1)(X+2)\cdots(X+N^{1/2}),$$

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 utilisant $O\big(\mathsf{M}(\sqrt{N})\log N\big)$ opérations arithmétiques (en construisant un arbre binaire de feuilles $X+i$ comme en Section 4.2 du Chapitre 6).
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 les techniques d'évaluation multipoint rapide (Chapitre 6, Section 4.3), ceci peut se faire également en $O\big(\mathsf{M}(\sqrt{N})\log N\big)$ opérations arithmétiques.

Le coût total de cet algorithme est $O\big(\mathsf{M}(\sqrt{N})\log N\big)$ 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. La technique gagne encore par rapport à l'algorithme naïf si l'on emploie une multiplication par Karatsuba, mais ne présente aucun intérêt avec la multiplication naïve.

Cette technique se généralise aux suites hypergéométriques quelconques ainsi qu'au calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux. À cette fin, le polynôme à considérer est maintenant le polynôme matriciel

$$P(X) = A(X+m-1)\cdots A(X+1)A(X),$$

pour $m = (N/d)^{1/2}$, où A est la matrice définie en page 228.

Le produit $A(N-1)\cdots A(0)$ 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 séparément les évaluations multipoints des r^2 coordonnées de la matrice P.

La complexité arithmétique de cet algorithme est en $O\big(r^2\mathsf{M}(md)\log(md) + \mathsf{MM}(r,m)\log m\big) = O\big(r^2\mathsf{M}(\sqrt{dN})\log(dN) + r^\theta\mathsf{M}(\sqrt{dN})\log(N/d)\big)$ opérations arithmétiques, soit $O\big(\mathsf{M}(\sqrt{dN})\log(dN)\big)$ si on ne tient pas compte de la dépendance en l'ordre r de la récurrence.

Exercice 1. Écrire les détails de l'algorithme et de l'analyse de complexité.

Exercice 2. Étant donné un polynôme $f \in \mathbb{K}[X]$ de degré 2 et un entier positif N, borner le nombre d'opérations dans $\mathbb{K}$ suffisantes pour déterminer le coefficient de X^N du polynôme f^N. (Indication : La solution consiste à calculer par exponentiation binaire tous les coefficients u_0, …, u_N de $f^N \bmod X^{N+1}$. Une approche encore plus rapide repose sur le fait que les u_n satisfont à une récurrence à coefficients polynomiaux d'ordre 2.)

2.1. **Factorisation déterministe des entiers.** Supposons que nous devons factoriser un entier N. Tester tous les diviseurs plus petits que $\sqrt{N}$ a une complexité binaire linéaire en $\sqrt{N}$. Afin d'accélérer ce calcul, on peut 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 $f_0, \dots, f_{c-1}$ avec N : la complexité binaire associée est $O\big(\sqrt[4]{N}\,\mathsf{M}_{\mathbb{Z}}(\log N)\log\log N\big)$.

Ainsi, la principale difficulté est de calculer les valeurs f_i. Pour ce faire, on prend $\mathbb{A} = \mathbb{Z}/N\mathbb{Z}$ et F le polynôme $(X+1)\cdots(X+c) \in \mathbb{A}[X]$, qu'on évalue en les points $0, c, 2c, \dots, c(c-1)$ en $O(\mathsf{M}(c)\log c)$ opérations de $\mathbb{A}$. Par le théorème « évaluation interpolation », puisque c est d'ordre $\sqrt[4]{N}$, la complexité pour calculer les f_i est de $O(\mathsf{M}(\sqrt[4]{N})\log N)$ opérations modulo N, soit $O(\mathsf{M}(\sqrt[4]{N})\,\mathsf{M}_{\mathbb{Z}}(\log N)\log N)$ opérations bits. Ce terme est dominant et est donc aussi la complexité totale du calcul.

3. Scindage binaire

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

$$C(a,b) \le C(a,m) + C(m,b) + \mathsf{M}_{\mathbb{Z}}\big(\log P(a,m), \log P(m,b)\big).$$

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

$$C(a,b) \le 2C(m,b) + \mathsf{M}_{\mathbb{Z}}\big(P(m,b)\big).$$

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

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

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 $\mathsf{M}_{\mathbb{Z}}$, la dernière inégalité devient

$$C(0,n) \le 2^k C(n/2^k,n) + k\mathsf{M}_{\mathbb{Z}}(n\log n).$$

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

$$C(0,n) \le O(\log n) + \mathsf{M}_{\mathbb{Z}}(n\log n)\log n$$

donc à une complexité binaire pour le calcul de $N!$ dans $O\big(\mathsf{M}_{\mathbb{Z}}(N\log N)\log N\big)$.

Si la multiplication entière utilisée repose sur la FFT, cette complexité s'écrit $O(N\log^3 N\log\log N)$; si la multiplication entière utilisée est d'exposant α strictement plus grand que 1, elle s'écrit $O\big((N\log N)^\alpha\big)$.

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 le terme 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 $O\big(\mathsf{M}_{\mathbb{Z}}(dN\log N)\log N\big)$.

Ensuite, si le dénominateur est non nul, la méthode de Newton (Chapitre 5, théorème 2) permet d'effectuer la division finale en $O\big(\mathsf{M}_{\mathbb{Z}}(dN\log N)\big)$ 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 $(e_n)_{n\ge0}$ 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 $O(\mathsf{M}_{\mathbb{Z}}(N\log N)\log N)$ 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 $O(\mathsf{M}_{\mathbb{Z}}(n)\log n)$ opérations binaires. Il faut ensuite effectuer une division qui ne demande que $O(\mathsf{M}_{\mathbb{Z}}(n))$ 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.** Considérons le cas général d'une suite P-récursive vérifiant la récurrence (1) avec $p_i\in\mathbb{Z}[X]$, $i=0,\dots,r$. On garde les notations d pour une borne sur les degrés des polynômes p_i, et ℓ pour une borne sur la taille de leurs coefficients. La même technique du scindage binaire permet d'obtenir le résultat suivant.

Théorème 1. *La complexité binaire de la méthode du scindage binaire pour calculer u_N est en $O(r^\theta\mathsf{M}_{\mathbb{Z}}(dN\log N+\ell N+N\log r)\log N)$.*

Comme précédemment, cette borne de complexité peut être améliorée à $O(r^\theta\mathsf{M}_{\mathbb{Z}}(dN\log N+\ell N+N\log r))$ si l'exposant α de la complexité $\mathsf{M}_{\mathbb{Z}}(m)=O(m^\alpha)$ est supérieur à 1.

Exercice 3. Vérifier les formules de ce théorème.

Exercices

Exercice 4 (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 $Q=\sum_{i=0}^{2N}q_iX^i\in\mathbb{Z}[X]$ 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 :

$$\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k},\qquad k\ge1,n\ge1.$$

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

$$\mathrm{ind}(p, N) = \sum_{i=1}^{\infty} \left\lfloor \frac{N}{p^i} \right\rfloor,$$

où la notation $\lfloor x \rfloor$ représente la partie entière de x.

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

Exercice 5. 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 $O(\mathsf{M}(N))$ 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 $\mathbb{K} = \mathbb{Z}/2\mathbb{Z}$.
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 $O(\mathsf{M}(N\log N)\log N)$ opérations binaires.

Notes

La méthode du scindage binaire est un grand classique en calcul formel. Elle a été redécouverte à plusieurs reprises dans la littérature, dans différents contextes : conversion d'une base à l'autre, calcul de décimales de constantes comme π ou e [7], calcul avec des récurrences linéaires [10], [8, §6].

L'une des premières références mentionnant le scindage binaire est [1, §178], qui suggère (sans preuve) qu'il permet l'évaluation numérique rapide des fonctions D-finies. L'article de synthèse [2, §12] est une bonne référence sur l'historique de la méthode.

L'algorithme par pas de bébés et pas de géants a été introduit par Strassen [11] et généralisé dans [8] au problème du calcul d'un terme d'une suite récurrente linéaire à coefficients polynomiaux.

L'algorithme de factorisation déterministe en Section 2.1 est basé également sur la référence [11]. Une amélioration de cet algorithme se trouve dans [6]. Notons qu'on ne connaît pas de meilleur algorithme déterministe ; l'existence d'un tel algorithme est un grand problème ouvert.

On trouve également dans [6] une application cryptographique du calcul du N-ième terme d'une suite P-récursive au problème du comptage de points sur une courbe hyperelliptique sur un corps fini. Le point de départ de cette application est l'exercice 2, inspiré de [9, Pb. 4].

Les algorithmes de recherche de solutions polynomiales d'équations fonctionnelles linéaires évoqués dans l'introduction sont donnés dans [5, 4].

Le résultat de la Section 3.3 est dû à [7] ; 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 4 consacrée au calcul de $N!$ est due à P. Borwein [3].

Bibliographie

[1] Beeler (Michael), Gosper (R.), and Schroeppel (Richard). – *HAKMEM.* – MIT, 1972, *Artificial Intelligence Memo No. 239.*

[2] Bernstein (Daniel J.). – Fast multiplication and its applications. In *Algorithmic number theory : lattices, number fields, curves and cryptography*, pp. 325–384. – Cambridge Univ. Press, 2008.

[3] Borwein (Peter B.). – On the complexity of calculating factorials. *Journal of Algorithms*, vol. 6, n3, 1985, pp. 376–380.

[4] Bostan (A.), Chyzak (F.), Cluzeau (T.), and Salvy (B.). – Low complexity algorithms for linear recurrences. In *ISSAC'06*, pp. 31–38. – ACM, New York, 2006.

[5] Bostan (Alin), Cluzeau (Thomas), and Salvy (Bruno). – Fast algorithms for polynomial solutions of linear differential equations. In *ISSAC'05*. pp. 45–52. – ACM Press, NY, 2005.

[6] Bostan (Alin), Gaudry (Pierrick), and Schost (Éric). – Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM J. Comput.*, vol. 36, n6, 2007, pp. 1777–1806.

[7] Brent (R. P.). – Multiple-precision zero-finding methods and the complexity of elementary function evaluation. In *Analytic computational complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1975)*, pp. 151–176. – Academic Press, New York, 1976.

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

[9] Flajolet (Philippe) and Salvy (Bruno). – The Sigsam challenges : Symbolic asymptotics in practice. *SIGSAM Bulletin*, vol. 31, n4, December 1997, pp. 36–47.

[10] Kogge (P.) and Stone (H.). – A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comp.*, vol. C-22, 1973, pp. 786–793.

[11] Strassen (V.). – Einige Resultate über Berechnungskomplexität. *Jber. Deutsch. Math.-Verein.*, vol. 78, n1, 1976/77, pp. 1–8.

CHAPITRE 14

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 de taille $m \times k$ à coefficients 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 $\mathsf{M}(d)$ opérations dans $\mathbb{K}$. L'exposant θ est tel que la multiplication de deux matrices $n \times n$ à coefficients dans $\mathbb{K}$ coûte $O(n^\theta)$ opérations dans $\mathbb{K}$.

Nous aurons aussi besoin de produits de matrices de $\mathcal{M}_n(\mathbb{K}[X]_d)$. Dans le cas le plus général, ce produit est connu pour pouvoir être exécuté en $O(n^\theta\, \mathsf{M}(d))$ 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 $O(n^\theta d + n^2\, \mathsf{M}(d) \log d)$; dans les mêmes conditions, en choisissant des points en progression géométrique, cette complexité peut être encore abaissée à $O(n^\theta d + n^2\, \mathsf{M}(d))$.

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 appartenant à $\mathcal{M}_n(\mathbb{K}[X]_d)$. 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 (Chapitre 9).

Résolution de $A(X)Y(X) = B(X)$ [Moenk-Carter 1979]
Entrée : $A \in \mathcal{M}_n(\mathbb{K}[X]_d)$, $B \in \mathcal{M}_{n,1}(\mathbb{K}[X]_{d-1})$
Sortie : le vecteur de fractions 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. Développement comme une fraction rationnelle

La Proposition 3 du Chapitre 4 (page 131) montre que la méthode de Newton fonctionne pour toute matrice inversible de séries. Avec cet algorithme, le calcul de la fraction rationnelle solution de (1) demande $O(n^\theta \mathsf{M}(nd))$ opérations dans $\mathbb{K}$.

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 2. *Soit $A(X) \in \mathcal{M}_n(\mathbb{K}[X]_d)$ 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*

$$A^{-1}B = a_0 + a_1X + \cdots + a_{k-1}X^{k-1} + X^kA^{-1}B_k, \tag{2}$$

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 $a_0, \ldots, a_{k-1}, B_k$.

La proposition suivante estime le coût de cet algorithme. Elle peut être vue comme l'analogue matriciel du Théorème 3 au Chapitre 5 (page 143).

Proposition 1. *Soit $A \in \mathcal{M}_n(\mathbb{K}[X]_d)$ avec $A(0)$ inversible, alors on peut calculer les $N \geq d$ premiers coefficients de A^{-1} en $O(n^\theta N\, \mathsf{M}(d)/d)$ 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 $O(n^2 N\, \mathsf{M}(d)/d)$ opérations dans $\mathbb{K}$.*

Démonstration. L'algorithme calcule d'abord l'inverse $S = A^{-1} \bmod X^d$ par l'algorithme de Newton, en $O(n^\theta \mathsf{M}(d))$ opérations dans $\mathbb{K}$.

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 $O(n^\theta \mathsf{M}(d))$ opérations dans $\mathbb{K}$ si B a n colonnes, $O(n^2 \mathsf{M}(d))$ s'il n'en a qu'une. □

Avec cet algorithme, le calcul de la fraction rationnelle solution de (1) demande $O(n^3 \mathsf{M}(d))$ opérations dans $\mathbb{K}$.

3. L'algorithme de Storjohann

L'algorithme de Storjohann permet de calculer les N premiers coefficients du développement en série de $A^{-1}B$ en $O(n^{\theta-1}N\log N\,\mathsf{M}(d))$ opérations dans $\mathbb{K}$. Si $\theta < 3$, 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 $O(n^\theta\,\mathsf{M}(d)\log(nd))$ 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 type « 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 $a_0,\dots,a_{k-1}$. L'entrée nécessaire est moins grosse, et la complexité plus faible lorsque k est grand devant d. Pour une série $V = v_0+v_1X+\cdots$, on note

$$[V]_a^b := v_aX^a+\cdots+v_{a+b}X^{a+b},$$

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 $i = k-d,\dots,k-1$.

Ensuite, il faut calculer $[X^kB_k]_k^{d-1}$. L'extraction des coefficients de l'équation (2) donne

$$\begin{aligned}[X^kB_k]_k^{d-1} &= [B-A(a_0+\cdots+a_{k-1}X^{k-1})]_k^{d-1}\\ &= [B-A(a_{k-d}X^{k-d}+\cdots+a_{k-1}X^{k-1})]_k^{d-1},\end{aligned}$$

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. Noter que la ressemblance entre cette idée et celle de l'algorithme de Keller-Gehrig en page 119.

Ainsi, si l'on connaît $B_0, B_{2m},\dots,B_{2sm}$ et $[A^{-1}]_{m-2d}^{2d}$, alors le calcul de la suite $B_m, B_{3m},\dots,B_{(2s+1)m}$ ne requiert que $O(n^{\theta-1}s\,\mathsf{M}(d))$ opérations dans $\mathbb{K}$.

En itérant cette idée, en partant de B_0 et en supposant connus $[A^{-1}]_{2^k-2d+1}^{2d-2}$ pour $k = 0,\dots,\lceil\log(N/d)\rceil =: k_{\max}$, 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 calculer toute la suite $B_0, B_d, B_{2d},\dots,B_{d\lceil N/d\rceil}$ en $O(k_{\max}\,n^{\theta-1}N\,\mathsf{M}(d)/d)$ opérations dans $\mathbb{K}$. 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 $k = p$:

$$\begin{aligned}A^{-1} &= a_0+\cdots+a_{m-1}X^{m-1}+X^mA^{-1}B_m\\ &= a_0+\cdots+a_{m-1}X^{m-1}+X^m(a_0+\cdots+a_{p-1}X^{p-1}+X^pA^{-1}B_p)B_m.\end{aligned}$$

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)**.** *Soient 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 $O(n^\theta \, \mathsf{M}(d) \log(nd))$ opérations dans $\mathbb{K}$.*

Notes

L'idée de résoudre des systèmes linéaires à coefficients polynomiaux (resp. entiers) par développement de Taylor (resp. p-adique) et reconstruction rationnelle provient de [4, 1]. L'algorithme en page 237 et la Proposition 1 sont tirés de [4].

Dans tout ce texte, nous avons supposé que $A(0)$ est inversible. En fait, les 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 où effectuer ces calculs. Ces considérations sont prises en compte dans [5, 6].

L'article [6] montre que le déterminant d'une matrice polynomiale de taille n et degré au plus d peut se calculer en $O(n^\theta \, \mathsf{M}(d) \log^2(n))$ opérations arithmétiques. Storjohann et Villard [8] montrent comment calculer le rang et une base du noyau d'une telle matrice en $\tilde{O}(n^\theta d)$ opérations arithmétiques.

Pour n une puissance de 2, Jeannerod et Villard [3] ont donné un algorithme qui calcule l'inverse d'une matrice polynomiale inversible de taille n et degré au plus d en $\tilde{O}(n^3 d)$ opérations arithmétiques. Des réductions entre diverses opérations sur les matrices polynomiales ont été étudiées dans [2].

L'algorithme de ce chapitre et tous les autres résultats de [5, 6] ont été étendus au cas entier dans [7]. Il n'est toujours pas connu si l'on peut calculer le polynôme caractéristique d'une matrice polynomiale de taille n et degré au plus d en $\tilde{O}(n^\theta d)$ opérations arithmétiques.

Bibliographie

[1] Dixon (John D.). – Exact solution of linear equations using p-adic expansions. *Numer. Math.*, vol. 40, n1, 1982, pp. 137–141.

[2] Giorgi (Pascal), Jeannerod (Claude-Pierre), and Villard (Gilles). – On the complexity of polynomial matrix computations. In *ISSAC'03*. pp. 135–142. – ACM, New York, 2003.

[3] Jeannerod (Claude-Pierre) and Villard (Gilles). – Essentially optimal computation of the inverse of generic polynomial matrices. *J. Complexity*, vol. 21, n1, 2005, pp. 72–86.

[4] 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.

[5] Storjohann (Arne). – High-order lifting. In Mora (Teo) (editor), *ISSAC'02*. 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.

[6] Storjohann (Arne). – High-order lifting and integrality certification. *J. Symbolic Comput.*, vol. 36, n3-4, 2003, pp. 613–648.

[7] Storjohann (Arne). – The shifted number system for fast linear algebra on integer matrices. *J. Complexity*, vol. 21, n4, 2005, pp. 609–650.

[8] Storjohann (Arne) and Villard (Gilles). – Computing the rank and a small nullspace basis of a polynomial matrix. In *ISSAC'05*, pp. 309–316. – ACM, New York, 2005.

CHAPITRE 15

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 Chapitre 4 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 chapitre sont présentés au tableau 1, dans 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 tout ce chapitre nous ferons *l'hypothèse supplémentaire que* $2, 3, \ldots, N$ *sont inversibles dans* $\mathbb{A}$.

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

$$y(X) = y(0) \exp\left(\int A(X)\right). \tag{1}$$

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, 1 solution)	$O(r\,\mathsf{M}(N)\log N)$	$O(drN)$	$O\left(\mathsf{M}(r)\frac{N}{r}\right)$	N
(système, base)	$O(r^2\mathsf{M}(N))$	$O(dr^\theta N)$	$O(r\mathsf{M}(r)N)$	r^2N
(système, 1 solution)	$O(r^2\,\mathsf{M}(N)\log N)$	$O(dr^2N)$	$O(\mathsf{M}(r)N)$	rN

Table 1: Complexité de la résolution d'équations et de systèmes différentiels linéaires pour $N \gg r$.

Le calcul utilisant cette formule est dominé par celui de l'exponentielle, donc en $O(\mathsf{M}(N))$ par les algorithmes du Chapitre 4.

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 $F(X,Y) \in \mathbb{A}[[X,Y]]$ une série bivariée telle que, pour toute série $s(X) \in \mathbb{A}[[X]]$, 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 termes 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 Chapitre 4 pour avoir $\mathsf{L}(N) = O(N\sqrt{N \log N}\,\mathsf{M}(N))$ qui domine alors la complexité de la résolution. Cependant, pour des équations plus simples, la composition avec F et sa dérivée pourra s'effectuer 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

$$\begin{aligned} y_{k+1} &:= y_k + h_k, \\ h_k &:= \exp\left(\int \frac{\partial F}{\partial y}(X,y_k)\right) \int \big(F(X,y_k) - y_k'\big) \exp\left(-\int \frac{\partial F}{\partial y}(X,y_k)\right) \bmod X^{2^{k+1}}. \end{aligned}$$

Il reste à prouver la convergence quadratique. Comme pour le théorème sur l'itération de Newton sur les séries du Chapitre 4 (p. 132), 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 théorème « 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, avec $a_r(0) \neq 0$, 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 l'injectant dans l'équation, ce qui mène à

$$XU' + (I - XA(X))U = C, \quad C = B + A(X)v.$$

Soient maintenant $d < N$ et $U = U_0 + X^d U_1$ où U_0 est un 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 $p \in \{1, \ldots, N\}$.

L'algorithme récursif est donc le suivant :

DiviserPourRégner(A, p, s)

Entrée : $A_0, \dots, A_{N-p}$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $s_0, \dots, s_{N-p}$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $s = \sum s_i X^i$, $p \in \{1, \dots, N\}$.
Sortie : y dans $\mathcal{M}_{r\times \ell}(\mathbb{A})[X]$ tel que $\deg_X y \leq N - p$ et $Xy' + (pI - XA)y = s + O(X^{N-p+1})$.

$m \leftarrow N - p$
Si $m = 1$ alors renvoyer $p^{-1}s(0)$
sinon
  $d \leftarrow \lfloor m/2 \rfloor$
  $y_0 \leftarrow$ DiviserPourRégner$(A, p + d, s \bmod X^d)$
  $R \leftarrow [s - Xy_0' - (pI - XA)y_0]_d^m$
  $y_1 \leftarrow$ DiviserPourRégner$(A, p + d, R)$
  renvoyer $y_0 + X^d y_1$

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.

SolDPR(A, N, B, v)

Entrée : $A_0, \dots, A_N$ dans $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$, $B_0, \dots, B_N$ et v dans $\mathcal{M}_{r\times \ell}(\mathbb{A})$, $B = \sum B_i X^i$.
Sortie : $y = \sum_{i=0}^{N} y_i X^i$ dans $\mathcal{M}_{r\times \ell}(\mathbb{A})[X]$ tel que $y' - Ay = B$ et $y(0) = v$.

  renvoyer $v + X$DiviserPourRégner$(A, 1, B + Av)$

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* SolDPR *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 $\ell = r$ (calcul de toute une base des solutions), mais dans ce cas, 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 théorème « 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

$$Y \mapsto Y' - A(X)Y$$

SolSysDiffHom(A, N, Y_0)

Entrée : $Y_0, A_0, \ldots, A_{N-2}$ in $\mathcal{M}_{r\times r}(\mathbb{A})$, $A = \sum A_i X^i$.

Sortie : $Y = \sum_{i=0}^{N-1} Y_i X^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$, et $Z = Y^{-1} + O(X^{N/2})$.

$Y \leftarrow (I + XA_0)Y_0$
$Z \leftarrow Y_0^{-1}$
$m \leftarrow 2$
tant que $m \leq N/2$ faire
    $Z \leftarrow Z + Z(I_r - YZ) \bmod X^m$
    $Y \leftarrow Y - Y\left(\int Z(Y' - A \bmod X^{2m-1}Y)\right) \bmod X^{2m}$
    $m \leftarrow 2m$
renvoyer Y, Z

FIGURE 1: Résolution de $Y' = A(X)Y$, $Y(0) = Y_0$ par itération de Newton.

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 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 troncature, 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

$$Z := z(2I - yz) \bmod X^m, \quad Y := y\left(I - \int Z(y' - Ay)\right) \bmod X^{2m}.$$

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

$$\begin{aligned} I - YZ &= I - y\big(z + z(I - yz)\big) + O(X^m) \\ &= (I - yz) - yz(I - yz) + O(X^m) \\ &= (I - yz)^2 + O(X^m) = O(X^m). \end{aligned}$$

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 théorème « diviser pour régner » mène au résultat de complexité.

Théorème 2 (Newton pour les systèmes différentiels linéaires)**.** *L'algorithme* SolSysDiffHom *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 sont données au Chapitre 6 :

$$\mathsf{MM}(r, N) = O(r^\theta N + r^2 \mathsf{M}(N)).$$

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 Chapitre 4.

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 vue au Chapitre 8 : les solutions séries de ces équations ont des coefficients qui vérifient des récurrences linéaires.

Les résultats du Chapitre 13 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 $S = s_0 + s_1 X + s_2 X^2 + \dots$ est une série, cette transformée est définie par

$$\mathcal{L}S = s_0 + 1!s_1 X + 2!s_2 X^2 + \dots.$$

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 :

CoefficientsConstants(A, v, N)

Entrée : A in $\mathcal{M}_{r\times r}(\mathbb{A})$, $v \in \mathcal{M}_{r\times 1}(\mathbb{A})$.

Sortie : $Y = \sum_{i=0}^{N-1} Y_i X^i$ dans $\mathcal{M}_{r\times r}(\mathbb{A})[X]$ tel que $Y' = AY + O(X^{N-1})$.

1. Calculer les vecteurs $v, Av, A^2v, A^3v, \ldots, A^{2r}v$;
2. Pour tout $j = 1, \ldots, r$:
   (a) reconstruire la fraction rationnelle ayant pour développement en série $\sum (A^i v)_j X^i$;
   (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 $O(r^\theta \log r)$ opérations, est à la base de l'algorithme de Keller-Gehrig pour le calcul du polynôme caractéristique d'une matrice présenté au Chapitre 3. 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 (Chapitre 9), mais cette partie du calcul ne dépend pas de N. L'étape 2 (b) utilise l'algorithme de développement de fractions rationnelles du Chapitre 5 (Théorème 3, page 143), et c'est là que se concentre le gain en complexité.

4. Extensions

4.1. **Composer se ramène à résoudre.** Le seul algorithme du Chapitre 4 dont la complexité n'est pas quasi-optimale est l'algorithme de composition.

Dans les applications où l'on connaît une équation $\Phi(X, f, f', \ldots, f^{(m)}) = 0$ 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 fait 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 chapitre.

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 $O(\cdot)$ (ou deux implantations !) pour déterminer laquelle des deux méthodes est préférable.

Exercice 3. Les polynômes de Legendre sont définis par

$$P_n(X) = \frac{1}{2^n \cdot n!} \left((X^2 - 1)^n\right)^{(n)}.$$

Montrer que $P_n(X)$ vérifie l'équation différentielle

$$\frac{d}{dX}\left[(1 - X^2)\frac{d}{dX}P_n(X)\right] + n(n+1)P_n(X) = 0.$$

En déduire que pour toute série de terme constant nul $F \in \mathbb{Q}[[X]]$, et pour tout $N \geq 0$, la série composée $P_N \circ F \bmod X^N$ peut se calculer en $O(\mathsf{M}(N))$ opérations.

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*

$$\begin{cases} y_1'(t) = \varphi_1(t, y_1(t), \dots, y_r(t)), \\ \qquad \vdots \\ y_r'(t) = \varphi_r(t, y_1(t), \dots, y_r(t)), \end{cases}$$

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 chapitre 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 [5]. Des algorithmes plus récents se trouvent dans [4].

Bibliographie

[1] Bostan (A.), Chyzak (F.), Ollivier (F.), Salvy (B.), Schost (É.), and Sedoglavic (A.). – Fast computation of power series solutions of systems of differential equations. In *SODA'07*. pp. 1012–1021. – SIAM, January 2007. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana.

[2] Brent (R. P.) and Kung (H. T.). – Fast algorithms for manipulating formal power series. *Journal of the ACM*, vol. 25, n4, 1978, pp. 581–595.

[3] Hanrot (Guillaume), Quercia (Michel), and Zimmermann (Paul). – The middle product algorithm I. *Appl. Algebra Engrg. Comm. Comput.*, vol. 14, n6, March 2004, pp. 415–438.

[4] van der Hoeven (Joris). – Newton's method and FFT trading. *Journal of Symbolic Computation*. – To appear. Available at `http://hal.archives-ouvertes.fr/hal-00434307/fr/`.

[5] van der Hoeven (Joris). – Relax, but don't be too lazy. *Journal of Symbolic Computation*, vol. 34, n6, 2002, pp. 479–542.

CHAPITRE 16

Exercices récapitulatifs

Sauf mention contraire, dans toute la suite, $\mathbb{K}$ désigne un corps effectif de caractéristique nulle.

Exercice 1 (Multiplication de polynômes à deux variables)**.** Soient f et g dans $\mathbb{K}[X,Y]$ de degrés au plus d_X en X et au plus d_Y en Y.

1. Montrer qu'il est possible de calculer tous les coefficients de $h = fg$ en $O(\mathsf{M}(d_X d_Y))$ opérations dans $\mathbb{K}$. Utiliser la substitution $X \leftarrow Y^{2d_Y+1}$ pour se ramener à une multiplication de polynômes à une variable.
2. Améliorer ce résultat en donnant un schéma d'évaluation-interpolation qui permette le calcul des coefficients de h en $O(d_X\mathsf{M}(d_Y) + d_Y\mathsf{M}(d_X))$ opérations de $\mathbb{K}$.

Exercice 2 (Équivalence des complexités de diverses opérations polynomiales)**.** Soit $\mathbb{K}$ un corps de caractéristique différente de 2 et soient $\mathsf{M}(n), \mathsf{I}(n), \mathsf{D}(n)$ et $\mathsf{S}(n)$ les complexités (mesurées en nombre d'opérations arithmétiques dans $\mathbb{K}$) pour calculer respectivement : le produit de deux polynômes de $\mathbb{K}[X]$ de degré $< n$, l'inverse modulo X^n d'un polynôme de degré $< n$, le reste de la division d'un polynôme de degré $< 2n$ par un polynôme de degré n et le carré d'un polynôme de degré $< n$. Le but de l'exercice est de prouver que les fonctions $\mathsf{M}, \mathsf{I}, \mathsf{D}$ et S ont le même ordre de grandeur.

1. Par quelle technique peut-on montrer que $\mathsf{I} \in O(\mathsf{M})$ et $\mathsf{D} \in O(\mathsf{M})$?
2. Prouver l'identité $y^2 = \left(y^{-1} - (y+1)^{-1}\right)^{-1} - y$ et conclure que $\mathsf{S} \in O(\mathsf{I})$.
3. Montrer que $\mathsf{M} \in O(\mathsf{S})$, en utilisant l'égalité $2fg = (f+g)^2 - f^2 - g^2$.
4. Le polynôme réciproque d'un polynôme $b \in \mathbb{K}[X]$ de degré n est noté $\text{réc}_n(b)$. Établir une relation entre $\text{réc}_n(b)^{-1} \bmod X^n$ et le quotient de la division de X^{2n} par b. En déduire que $\mathsf{I} \in O(\mathsf{D})$.
5. Conclure que $O(\mathsf{M}) = O(\mathsf{I}) = O(\mathsf{S}) = O(\mathsf{S})$.

Exercice 3 (Décalage rapide de polynômes)**.**

1. Montrer que si P est un polynôme dans $\mathbb{Q}[X]$ de degré au plus n, alors, quel que soit $a \in \mathbb{Q}$, on peut calculer les coefficients du « polynôme décalé » $Q(X) = P(X+a)$ en $O(\mathsf{M}(n))$ opérations dans $\mathbb{Q}$.

   Indication. Commencer par prouver l'identité suivante dans $\mathbb{Q}[X]$:

   $$\sum_{i=0}^{n} P^{(i)}(a) X^{n-i} = \left(\sum_{i=0}^{n} P^{(i)}(0) X^{n-i}\right) \times \left(\sum_{i=0}^{n} \frac{(aX)^i}{i!}\right) \mod X^{n+1}.$$

2. Montrer que les coefficients des numérateurs et des dénominateurs des fractions rationnelles

   $$R_0 = \frac{1}{X-1} + \frac{1}{X-2} + \cdots + \frac{1}{X-n} \quad \text{et} \quad R_1 = \frac{1}{X-1} + \frac{2}{X-2} + \cdots + \frac{n}{X-n}$$

   peuvent être calculés en $O(\mathsf{M}(n))$ opérations dans $\mathbb{Q}$.

   Indication. Pour R_0, donner un algorithme de type « diviser pour régner » basé sur l'algorithme de la question (1). Pour R_1, en ramener le calcul à celui de R_0.

La formule de Lagrange montre qu'il existe un unique polynôme $P(X) \in \mathbb{Q}[X]$ de degré strictement inférieur à n tel que

$$P(i) = (-1)^i \binom{n}{i-1}^{-1}, \quad \text{pour tout} \quad i = 1, 2, \ldots, n.$$

3. Exprimer P en termes de R_0 et R_1 et donner un algorithme qui calcule les coefficients de P en $O(\mathsf{M}(n))$ opérations dans $\mathbb{Q}$. Ces coefficients sont supposés connus dans la suite de l'exercice.
4. Montrer que la suite d'éléments de $\mathbb{Q}$ de terme général $a_i = P(i)$ vérifie une récurrence linéaire d'ordre n à coefficients constants.
5. En déduire un algorithme pour calculer les valeurs $P(n+1), \ldots, P(2n)$ en $O(\mathsf{M}(n))$ opérations dans $\mathbb{Q}$.
6. Montrer qu'on peut calculer $P(n+1), \ldots, P(2n)$ en $O(n)$ opérations dans $\mathbb{Q}$.

Exercice 4 (Extrapolation sur une suite arithmétique)**.** Soit P un polynôme inconnu de $\mathbb{K}[X]$. Le but de l'exercice est de montrer que, à partir de la donnée d'un élément $a \in \mathbb{K}$, d'une borne d sur le degré de P, et des valeurs $P(0), P(1), \ldots, P(d)$, il est possible de déterminer l'ensemble des valeurs $P(a), P(a+1), \ldots, P(a+d)$ en $O(\mathsf{M}(d))$ opérations dans $\mathbb{K}$.

1. Traiter d'abord le cas où a appartient à l'ensemble $\{-d, -d+1, \ldots, d-1, d\}$.

Supposons dans la suite que $a \in \mathbb{K}$ est tel que tous les éléments $a-d, \ldots, a+d$ sont non nuls. On rappelle la formule d'interpolation de Lagrange

$$P(X) = \sum_{i=0}^{d} p_i \prod_{j=0,\, j\neq i}^{d} (X-j), \quad \text{où} \quad p_i = \frac{P(i)}{\prod_{j=0,\, j\neq i}^{d} (i-j)}. \tag{L}$$

2. Montrer que les éléments $p_0, p_1, \ldots, p_d$ peuvent être tous calculés en $O(d)$ opérations dans $\mathbb{K}$.

En substituant $X = a + k$ dans l'équation (L), on obtient l'égalité

$$P(a+k) = \Delta_k \cdot \sum_{i=0}^{d} \frac{p_i}{a+k-i}, \quad \text{où} \quad \Delta_k = \prod_{j=0}^{d} (a+k-j). \tag{$\tilde{\mathsf{L}}$}$$

3. Montrer que les éléments Δ_k pour $k = 0, \ldots, d$, peuvent être tous calculés en $O(d)$ opérations dans $\mathbb{K}$.
4. Utiliser l'égalité ($\tilde{\mathsf{L}}$) pour conclure que l'ensemble des valeurs $P(a), P(a+1), \ldots, P(a+d)$ peut se calculer en $O(\mathsf{M}(d))$ opérations dans $\mathbb{K}$.

Exercice 5 (Décomposition en éléments simples)**.** On considère une fraction rationnelle

$$F(X) = \frac{P(X)}{Q_1^{\ell_1}(X) \cdots Q_k^{\ell_k}(X)},$$

où $P, Q_1, \ldots, Q_k$ sont des polynômes de $\mathbb{Q}[X]$ et les ℓ_i sont des entiers strictement positifs tels que

(H1) $\deg(P) < \sum_{i=1}^{k} \ell_i \cdot \deg(Q_i) = n$, et

(H2) $Q_1, \ldots, Q_k$ sont deux à deux premiers entre eux.

La décomposition en éléments simples de F est une écriture

$$\frac{P(X)}{Q_1^{\ell_1}(X) \cdots Q_k^{\ell_k}(X)} = \sum_{i=1}^{k} \sum_{j=1}^{\ell_i} \frac{C_{i,j}(X)}{Q_i^j(X)},$$

où les $C_{i,j}$ sont des polynômes de $\mathbb{Q}[X]$ tels que $\deg(C_{i,j}) < \deg(Q_i)$ pour tous i, j. L'existence et l'unicité des polynômes $C_{i,j}$ est admise. On appelle DES_n le problème du calcul des polynômes $C_{i,j}$ à partir des polynômes $P, Q_1, \ldots, Q_k$ et des entiers $n, \ell_1, \ldots, \ell_k$ satisfaisant aux hypothèses (H1) et (H2).

L'objectif de cet exercice est de montrer qu'il est possible de résoudre le problème DES_n en $O(\mathsf{M}(n)\log(n))$ opérations. Ici, et dans toute la suite de l'exercice, par *opération* on entend *opération arithmétique dans le corps* $\mathbb{Q}$ et $\mathsf{M}(n)$ représente une borne sur le nombre d'opérations suffisant à multiplier deux polynômes de $\mathbb{Q}[X]$ de degré au plus n.

1. Donner un algorithme simple, reposant sur de l'algèbre linéaire, qui résout le problème DES_n et estimer le nombre d'opérations que cet algorithme utilise.
2. Dans le cas $k = 1$, le problème DES_n se spécialise au problème du calcul, à partir de deux polynômes $P, Q \in \mathbb{Q}[X]$ tels que $\deg(P) < \ell \cdot \deg(Q) = n$, des polynômes $C_1, \ldots, C_\ell$ vérifiant

$$\frac{P(X)}{Q^\ell(X)} = \sum_{j=1}^{\ell} \frac{C_j(X)}{Q^j(X)}, \quad \text{et } \deg(C_j) < \deg(Q) \text{ pour tout } j \geq 1.$$

   Montrer que dans ce cas, le problème peut se résoudre en $O(\mathsf{M}(n)\log(n))$ opérations.

Le but des questions 3–9 ci-dessous est de montrer que le problème DES_n avec $\ell_i = 1$ pour tout i peut se résoudre en $O(\mathsf{M}(n)\log(n))$ opérations. Autrement dit, étant donnés les polynômes $P, R_1, \ldots, R_k$ dans $\mathbb{Q}[X]$, avec $\deg(P) < \sum_{i=1}^{k} \deg(R_i) = n$ et $R_1, \ldots, R_k$ deux à deux premiers entre eux, on veut calculer en $O(\mathsf{M}(n)\log(n))$ opérations les polynômes $C_1, \ldots, C_k \in \mathbb{Q}[X]$ tels que

$$\frac{P(X)}{R_1(X)\cdots R_k(X)} = \sum_{i=1}^{k} \frac{C_i(X)}{R_i(X)}, \quad \text{et } \deg(C_i) < \deg(R_i) \text{ pour tout } i \geq 1.$$

Pour ce faire, on introduit

$$R(X) = \sum_{i=1}^{k} \left(\prod_{\substack{j=1 \\ j \neq i}}^{k} R_j(X) \right)$$

et pour $1 \leq i \leq k$, on considère le reste $D_i(X)$ de la division euclidienne de $R(X)$ par $R_i(X)$.

3. Donner un algorithme qui calcule le polynôme $R(X)$ en $O(\mathsf{M}(n)\log(k))$ opérations.
4. Montrer que les polynômes $D_1, \ldots, D_k$ peuvent être calculés eux aussi en $O(\mathsf{M}(n)\log(k))$ opérations.
5. Montrer que pour tout $1 \leq i \leq k$, les polynômes D_i et R_i sont premiers entre eux.
6. En notant E_i l'inverse de D_i modulo R_i, montrer que C_i est égal au reste de la division euclidienne de $P \cdot E_i$ par R_i.
7. Donner un algorithme pour le calcul des E_i en $O(\mathsf{M}(n)\log(n))$ opérations.
8. Donner un algorithme pour le calcul des C_i, en $O(\mathsf{M}(n)\log(k))$ opérations.
9. Conclure qu'on peut résoudre le problème DES_n avec $\ell_i = 1$ pour tout i en $O(\mathsf{M}(n)\log(n))$ opérations.
10. Conclure qu'il est possible de résoudre le problème DES_n en $O(\mathsf{M}(n)\log(n))$ opérations.

Exercice 6 (Polynômes de Fibonacci)**.** Soit $F_n(X) \in \mathbb{K}[X]$ la suite des polynômes de Fibonacci, définie par les conditions initiales $F_0 = 1$, $F_1 = X$ et la récurrence $F_{n+1} = XF_n + F_{n-1}$ pour tout $n \geq 1$. Soit $N \geq 1$.

1. Estimer le coût d'un calcul direct de tous les coefficients de $F_N(X)$.
2. Montrer qu'il est possible de calculer tous les coefficients de $F_N(X)$ en utilisant $O\big(\mathsf{M}(N)\big)$ opérations dans $\mathbb{K}$.
3. Montrer qu'il est possible de calculer la somme des coefficients de F_N en seulement $O(\log N)$ opérations dans $\mathbb{K}$.
4. Donner un algorithme qui permet le calcul des coefficients de F_N en $O(N)$ opérations dans $\mathbb{K}$.

Exercice 7 (Fonctions algébriques et résultants)**.** Soit $y(X)$ une fonction algébrique annulée par $P(X,Y) \in \mathbb{Q}[X,Y]$.

1. Montrer que la dérivée $y'(X)$ est aussi algébrique, en exhibant un polynôme résultant $R(X,Y)$ qui l'annule.
2. Déterminer un tel polynôme R dans le cas où $P(X,Y) = XY^2 - Y + 1$.
3. Supposant que P est de degré au plus d en X et en Y, estimer les degrés en X et en Y du polynôme R, puis proposer un algorithme pour son calcul.
4. Analyser la complexité de votre algorithme, mesurée en nombre d'opérations arithmétiques dans $\mathbb{Q}$. En utilisant les meilleurs algorithmes vus en cours pour les opérations fondamentales sur les polynômes et les matrices, à quel exposant aboutissez-vous ? Ici, par *exposant* on entend un réel α tel que votre algorithme a une complexité en $\tilde{O}(n^\alpha)$, c'est-à-dire linéaire en n^α à des facteurs logarithmiques près.

Exercice 8 (Produit diamant)**.** On considère deux polynômes unitaires $f, g \in \mathbb{Q}[T]$, de degrés strictement plus petits que n. Étant donné $H \in \mathbb{Q}[X,Y]$, dont les degrés partiels par rapport à chacune des variables X et Y sont strictement inférieurs à n, le *produit diamant* $f \diamond_H g$ est le polynôme de degré $D = n^2$ défini par

$$f \diamond_H g = \prod_{f(\alpha)=0,\, g(\beta)=0} \big(T - H(\alpha,\beta)\big),$$

le produit étant pris sur les racines complexes de f et de g comptées avec multiplicités. On s'intéresse dans cet exercice au calcul efficace du produit diamant.

1. Montrer que le produit diamant peut s'exprimer à l'aide de résultants et que ses coefficients sont rationnels.

On note A l'algèbre quotient $\mathbb{Q}[X,Y]/(f(X), g(Y))$. Un élément $g \in A$ admet une écriture unique

$$g = \sum_{0 \le i,j < n} g_{i,j} x^i y^j$$

avec $g_{ij} \in \mathbb{Q}$, où x et y sont les images canoniques de X et Y dans A. On admet que si P_1, P_2 sont deux polynômes de $\mathbb{Q}[X,Y]$ de degrés au plus d_X en X et au plus d_Y en Y, on peut calculer leur produit $P_1 P_2$ en $O\big(\mathsf{M}(d_X d_Y)\big)$ opérations dans $\mathbb{Q}$.

2. Montrer que $O\big(\mathsf{M}(D)\big)$ opérations suffisent pour effectuer le produit de deux éléments de A.

À tout polynôme G de $\mathbb{Q}[X,Y]$ on associe l'application $\mathbb{Q}$-linéaire de multiplication par $G(x,y)$ dans A, définie par $v \in A \mapsto G(x,y)v \in A$. On admet que la trace, notée $\mathrm{Tr}(G)$, de cette application linéaire est égale à $\sum_{\alpha,\beta} G(\alpha,\beta)$, la somme étant prise sur les racines complexes α de f et β de g comptées avec multiplicités.

3. Montrer que, pour tout $s \in \mathbb{N}$, la trace $\mathrm{Tr}(H^s)$ s'exprime comme la somme des puissances s-ièmes des racines de $f \diamond_H g$.
4. Expliciter pour $0 \le i < n$ ce que valent $\mathrm{Tr}(X^i)$ et $\mathrm{Tr}(Y^i)$.

On rappelle que tout polynôme unitaire $P \in \mathbb{Q}[T]$ de degré au plus D peut être représenté soit par la suite de ces coefficients, soit par la suite des sommes des puissances s-ièmes de ses racines, pour $s = 0, \ldots, D$, et qu'il est possible de passer d'une représentation à l'autre en $O\big(\mathsf{M}(D)\big)$ opérations.

5. Donner un algorithme pour le calcul des valeurs $\mathrm{Tr}(X^i), 0 \le i < n$, en $O\big(\mathsf{M}(n)\big) \subseteq O(D)$ opérations arithmétiques. Même question pour les valeurs $\mathrm{Tr}(Y^i)$, $0 \le i < n$.
6. En déduire que l'ensemble des valeurs $\mathrm{Tr}(X^i Y^j)$, pour $0 \le i, j < n$, peut être calculé en $O(D)$ opérations dans $\mathbb{Q}$.
7. Donner un algorithme qui, pour tout $N \ge 1$, calcule la suite
$$\mathrm{Tr}(1), \mathrm{Tr}(H), \mathrm{Tr}(H^2), \ldots, \mathrm{Tr}(H^{N-1})$$
en $O\big(N\,\mathsf{M}(D)\big)$ opérations dans $\mathbb{Q}$.

8. En déduire un algorithme pour le calcul de $f \diamond_H g$ en $O\big(D\,\mathsf{M}(D)\big)$ opérations dans $\mathbb{Q}$.

Exercice 9 (Composition itérée de séries formelles)**.** Soit F une série de la forme $F(X) = f_1X + f_2X^2 + \cdots$ à coefficients dans $\mathbb{K}$. En notant $F^{[0]}(X) = X$, on définit la q-ième itérée de F par

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

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 $\mathsf{M}(N)$ 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. On note également $C(N)$ une borne sur le nombre d'opérations dans $\mathbb{K}$ nécessaires pour calculer la composition $f(g(X))$ de deux séries tronquées à l'ordre N telles que $g(0) = 0$. On rappelle qu'aucun algorithme quasi-optimal n'est connu pour la composition et que la meilleure complexité connue est $C(N) = O(\sqrt{N \log N}\mathsf{M}(N))$.

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

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

la constante K ne dépendant ni de N, ni de q.

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 $C_q(N) = O(q\sqrt{N \log N}\mathsf{M}(N))$.
3. Décrire une idée simple permettant de réduire la dépendance en q pour aboutir à $C_q(N) = O(\log q\sqrt{N \log N}\mathsf{M}(N))$.

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 $S(X) = s_1X + s_2X^2 + \cdots$ par

$$S(F(X)) = f_1S(X), \qquad s_1 = 1. \tag{S}$$

(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 (S), on va s'intéresser à la résolution d'une équation plus générale :

$$A(X)Y(F(X)) - B(X)Y(X) - C(X) = 0 \bmod X^k, \tag{E}$$

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

$$Y(X) = U(X) + X^nV(X),$$

où U est un polynôme de degré inférieur à n, montrer que U et V sont solutions d'équations du même type que (E). Expliciter les coefficients de ces équations.

(7) Décrire un algorithme de calcul des N premiers coefficients de Y en nombre d'opérations $O(\sqrt{N \log N}\mathsf{M}(N))$. En notant a_0, b_0, c_0 les termes constants des séries A, B, C, on supposera que $a_0f_1^m \neq b_0$, 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.

Exercice 10 (Multiplication polynomiale de complexité quasi-linéaire)**.** Le but de cet exercice est de montrer que la multiplication de polynômes de degré au plus N à coefficients dans un corps $\mathbb{K}$ contenant N éléments distincts en progression géométrique peut s'effectuer en complexité arithmétique quasi-linéaire par rapport à N.

Soit A et B deux polynômes de $\mathbb{K}[X]$, de degré strictement inférieurs à N. Soient n et d deux entiers tels que $N \leq nd$ et écrivons

$$A = A_0(X) + A_1(X)X^d + \cdots + A_{n-1}(X)(X^d)^{n-1},$$
$$B = B_0(X) + B_1(X)X^d + \cdots + B_{n-1}(X)(X^d)^{n-1},$$

où les polynômes $A_i(X), B_i(X) \in \mathbb{K}[X]$ ont degré au plus $d-1$. Soit $\mathcal{G}$ un sous-ensemble de $\mathbb{K}$ contenant $2d$ éléments distincts en progression géométrique.

1. Si T opérations de $\mathbb{K}$ suffisent pour multiplier dans $\mathbb{K}[X,Y]$ les polynômes
$$\mathsf{A}(X,Y) = A_0(X) + A_1(X)Y + \cdots + A_{n-1}(X)Y^{n-1} \quad \text{et}$$
$$\mathsf{B}(X,Y) = B_0(X) + B_1(X)Y + \cdots + B_{n-1}(X)Y^{n-1},$$
montrer qu'on peut multiplier $A(X)$ et $B(X)$ en $\mathsf{T} + O(nd)$ opérations de $\mathbb{K}$.

Pour calculer AB, il suffit donc d'effectuer le produit $\mathsf{C}(X,Y) = \mathsf{A}(X,Y)\mathsf{B}(X,Y)$, c'est-à-dire de calculer les coefficients $C_i(X) \in \mathbb{K}[X]$, avec $\deg(C_i) < 2d$, de

$$\mathsf{C}(X,Y) = C_0(X) + C_1(X)Y + \cdots + C_{2n-2}(X)Y^{2n-2}.$$

2. Montrer que le calcul du produit AB se ramène à $O(d)$ multiplications de $\mathbb{K}[Y]$ en degré $< n$ et $O(n)$ évaluations/interpolations sur les points de $\mathcal{G}$.
Indication : utiliser l'égalité $\mathsf{A}(g,Y)\mathsf{B}(g,Y) = \mathsf{C}(g,Y)$, valable pour $g \in \mathcal{G}$.
3. En déduire qu'il existe trois constantes $\alpha, \beta, \gamma \geq 1$, universelles (c'est-à-dire indépendantes de N, n, d), telles que la complexité arithmétique M de l'algorithme de (2) vérifie $\mathsf{M}(nd) \leq \alpha n\, \mathsf{M}(d) + \beta d\, \mathsf{M}(n) + \gamma nd$.
4. Écrire un algorithme récursif de multiplication dans $\mathbb{K}[X]$ de complexité arithmétique $O\left(N(\log N)^{\log(\alpha\beta)}\right)$ en degré N.

Exercice 11 (Multiplication d'opérateurs différentiels)**.** Dans l'algèbre de polynômes non-commutatifs $\mathbb{Q}[X]\langle\theta\rangle$ en les indéterminées X et $\theta = \frac{d}{dX}$ satisfaisant à la règle de commutation $\theta X = X\theta + X$, on considère deux éléments A et B donnés sous la forme

$$A = \sum_{i=0}^{n}\sum_{j=0}^{n} a_{i,j}X^i\theta^j \qquad \text{et} \qquad B = \sum_{i=0}^{n}\sum_{j=0}^{n} b_{i,j}X^i\theta^j$$

avec $a_{i,j} \in \mathbb{Q}$ et $b_{i,j} \in \mathbb{Q}$. Le but de cet exercice est d'étudier la complexité arithmétique du calcul du produit $C = BA$, c'est-à-dire du calcul, à partir des constantes $a_{i,j}, b_{i,j} \in \mathbb{Q}$, des constantes $c_{i,j} \in \mathbb{Q}$ telles que

$$C = \sum_{i=0}^{n}\sum_{j=0}^{n} c_{i,j}X^i\theta^j.$$

Les questions 1–3 qui suivent sont largement indépendantes.

1. (a) Montrer que $\theta^i X^\ell = X^\ell(\theta+\ell)^i$ pour $i \geq 0$ et $\ell \geq 0$, et en déduire que le calcul du produit $\theta^i X^\ell$ peut s'effectuer en $O(i)$ opérations arithmétiques dans $\mathbb{Q}$.
   (b) Donner des bornes, en fonction de n, sur les degrés de C en X et en θ.
   (c) Proposer un algorithme naïf pour le calcul de C et estimer son coût arithmétique.
2. Montrer qu'il est possible de calculer C en $O\big(n^2\,\mathsf{M}(n)\big)$ opérations dans $\mathbb{Q}$, à l'aide d'un schéma itératif. On pensera à écrire B sous la forme
$$B = \sum_{i=0}^{n} b_i(X)\theta^i, \qquad \text{avec } b_i(X) \in \mathbb{Q}[X].$$

Dans la suite, on montre comment améliorer les résultats des questions (1) et (2), grâce à un algorithme de type évaluation-interpolation.

3. (a) Calculer $\theta^j(X^k)$, pour $j \geq 0$ et $k \geq 0$. En déduire $P(\theta)(X^k)$, pour $k \geq 0$ et $P \in \mathbb{Q}[X]$.
   (b) Montrer que les coefficients de C sont uniquement déterminés par la suite des polynômes $C(X^0), C(X^1), \ldots, C(X^{2n})$.
   (c) Donner un algorithme de complexité arithmétique $O\big(n\,\mathsf{M}(n)\log n\big)$ pour le calcul de tous les polynômes $p_k := A(X^k)$, pour $0 \leq k \leq 2n$, et $q_i := B(X^i)$, pour $0 \leq i \leq 3n$.
   (d) Montrer que le calcul des polynômes $B(p_0), \ldots, B(p_{2n})$ se ramène à une multiplication de matrices de tailles $(4n+1) \times (3n+1)$ et $(3n+1) \times (2n+1)$.
   (e) Montrer que, à partir de $C(X^0), \ldots, C(X^{2n})$, il est possible de calculer les coefficients de C en $O\big(n\,\mathsf{M}(n)\log n\big)$ opérations arithmétiques.
   (f) Conclure en écrivant un algorithme complet pour le calcul de C basé sur (c), (d) et (e). Estimer sa complexité.

Exercice 12 (Une famille d'intégrales)**.** L'intégrale

$$c_{n,k} = \int_0^{+\infty} t^k K_0(t)^n \, dt,$$

où K_0 est une fonction de Bessel (voir ci-dessous), est intervenue récemment dans des études de physique liées au modèle d'Ising. Empiriquement, il est apparu que ces intégrales vérifient une récurrence de la forme

$$(k+1)^{n+1} c_{n,k} + \sum_{\substack{2 \leq j < n \\ j \text{ pair}}} P_{n,j}(k) c_{n,k+j} = 0, \tag{R}$$

où les $P_{n,j}$ sont des polynômes de degré au plus $n+1-j$.

Le but de cet exercice est de prouver l'existence d'une telle récurrence et de développer un algorithme permettant de la calculer. Le point de départ est l'équation différentielle

$$\theta^2(y) - t^2 y = 0$$

vérifiée par $y = K_0(t)$. Dans cette équation, θ représente l'opérateur $t\frac{d}{dt}$. Ainsi, $\theta^2(y) = t(ty')'$. Il sera plus commode de travailler avec θ qu'avec d/dt, et on rappelle $\theta(uv) = u\theta(v) + \theta(u)v$.

On admettra que les intégrales $c_{n,k}$ convergent pour tous n et k entiers positifs, et que pour tout i, j entiers positifs on a en outre

$$\lim_{t \to 0} t^j (K_0(t)^n)^{(i)} = \lim_{t \to \infty} t^j (K_0(t)^n)^{(i)} = 0.$$

1. Montrer que K_0^2 vérifie l'équation différentielle
$$\theta^3(y) - 4t^2\theta(y) - 4t^2 y = 0.$$
2. Plus généralement, montrer que pour tout $n \in \mathbb{N}$, K_0^n vérifie une équation différentielle linéaire. Donner une borne supérieure sur son ordre. Vérifier que cette borne donne la bonne valeur pour $n = 2$.
3. En déduire l'existence d'une récurrence linéaire vérifiée par les $c_{n,k}$, pour chaque n. Donner un algorithme calculant cette récurrence.
4. Montrer par exemple que la suite $c_{2,k}$ vérifie
$$k^3 c_{2,k} = 4(k+1)c_{2,k-2}.$$
5. Borner le degré des coefficients dans l'équation calculée en question (2).
6. En déduire une borne de complexité sur l'algorithme de (3) en nombre d'opérations dans $\mathbb{Q}$.

La suite de l'exercice montre comment accélérer ce calcul en tirant parti du fait que l'équation différentielle vérifiée par K_0 est d'ordre 2 seulement.

7. Soit $\phi_k := n(n-1)\cdots(n-k+1)K_0^{n-k}\theta(K_0)^k$. Montrer que
$$\phi_{k+1}(t) = \theta(\phi_k)(t) - k(n-k+1)t^2\phi_{k-1}(t).$$
8. Utiliser cette récurrence pour $n=2$ pour retrouver l'équation de la question (1), en récrivant $\phi_1, \phi_2, \ldots$ au moyen de ϕ_0.
9. Plus généralement, décrire un algorithme de calcul d'une équation différentielle linéaire vérifiée par K_0^n à partir du résultat de la question (7).
10. Montrer que la complexité du calcul de l'équation différentielle par cette méthode est bornée par $O(n^3)$.

Exercice 13 (Calcul rapide du polynôme caractéristique)**.** On considère un réel $\theta > 2$ tel que la multiplication de deux matrices de $\mathcal{M}_n(\mathbb{K})$ peut se faire en $\mathsf{MM}(n) = n^\theta$ opérations dans $\mathbb{K}$. L'objectif de cet exercice est de prouver que le polynôme caractéristique d'une matrice « générique » de $\mathcal{M}_n(\mathbb{K})$ peut être calculé en $O(\mathsf{MM}(n))$ opérations de $\mathbb{K}$.

Une matrice $F = (f_{ij})_{1\le i,j\le n}$ de $\mathcal{M}_n(\mathbb{K})$ sera appelée « de type m-Frobenius » si ses $n-m$ premières colonnes ne contiennent que des éléments nuls, à l'exception des éléments $f_{m+1,1}, f_{m+2,2}, \ldots, f_{n,n-m}$ qui valent tous 1. Par exemple, les matrices de type 1-Frobenius sont les matrices compagnon.

Dans la suite de l'exercice, on suppose que la taille n est une puissance de 2 et on note $n = 2^r$. On associe à A une suite de matrices $A_r = A, A_{r-1}, \ldots, A_1, A_0$ qui lui sont toutes semblables (c'est-à-dire de la forme $P^{-1}AP$, avec P inversible). Ces matrices sont définies de manière inductive. Pour passer de A_{i+1} à A_i, on calcule $s_i = n/2^i$ matrices, notées $A_{i,1} = A_{i+1}, A_{i,2}, \ldots, A_{i,s_i} = A_i$ et définies comme suit :

- On considère la matrice U_{ij} de type 2^i-Frobenius dont les 2^i dernières colonnes coïncident avec les 2^i dernières colonnes de A_{ij}.
  On suppose que A est « suffisamment générique » pour que U_{ij} soit inversible.
- On pose $A_{i,j+1} := U_{ij}^{-1}A_{i,j}U_{ij}$.

On admettra sans preuve que la matrice A_i ainsi construite est de type 2^i-Frobenius.

1. Montrer que tous les éléments de l'inverse $D = (d_{ij})$ d'une matrice compagnon $C = (c_{ij})$ avec $c_{1n} \neq 0$ sont nuls, à l'exception des éléments suivants
$$d_{1,2} = \cdots = d_{n-1,n} = 1,\ d_{n,1} = {c_{1,n}}^{-1}, d_{n-j+1,1} = -c_{j,n}\, d_{n,1},\ \text{pour } 2 \le j \le n.$$
   En déduire que les éléments de D peuvent se calculer en $O(n)$ opérations dans $\mathbb{K}$.
2. Étendre le résultat de la question (1), en montrant que l'inverse d'une matrice de type m-Frobenius peut se calculer en $O(n\,\mathsf{MM}(m)/m)$ opérations de $\mathbb{K}$.
3. En déduire que le passage de A_{i+1} à A_i se fait en $O(n^2\mathsf{MM}(2^i)/4^i)$ opérations de $\mathbb{K}$.
4. Conclure que le polynôme caractéristique de A peut se calculer en $O(\mathsf{MM}(n))$ opérations de $\mathbb{K}$.

Exercice 14 (Résolution de systèmes de Vandermonde transposés)**.** Soient $a_0, a_1, \ldots, a_n$ des éléments distincts de $\mathbb{K}$ et soit V la matrice de Vandermonde $V = (a_i^j)_{0\le i,j\le n}$. Le but de cet exercice est de montrer que, pour tout vecteur $\mathsf{b} \in \mathbb{K}^{n+1}$, on peut calculer l'unique solution $\mathsf{x} \in \mathbb{K}^{n+1}$ du système linéaire ${}^tV \cdot \mathsf{x} = \mathsf{b}$ en $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$.

1. Donner un algorithme pour le calcul des coefficients du numérateur et du dénominateur de la fraction rationnelle $R = \sum_{i=0}^n \frac{1}{1-a_iX}$ en $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$.
2. Montrer qu'à partir de R, on peut calculer en $O(\mathsf{M}(n))$ opérations dans $\mathbb{K}$ les $2n$ premiers éléments de la suite $(z_i)_{i\ge 0}$ définie par $z_i = a_0^i + a_1^i + \cdots + a_n^i$.

On rappelle que, par définition, une matrice de type Hankel est constante le long de toutes ses anti-diagonales. On rappelle le théorème suivant vu au Chapitre 11 :

Si H est une matrice de Hankel inversible de taille $n+1$, alors on peut résoudre le système linéaire $H\mathsf{u} = \mathsf{b}$ en $O(\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$.

3. Montrer que la matrice $H := {}^tV \cdot V$ vérifie l'hypothèse de ce théorème.

4. Utiliser (1) et (2) pour estimer la complexité du calcul des éléments de H.
5. Conclure.
6. (Bonus) Écrire un algorithme résolvant ${}^tV \cdot \mathsf{x} = \mathsf{b}$, obtenu par transposition (au sens du Chapitre 12) de l'algorithme d'interpolation rapide vu au Chapitre 6.

Exercice 15 (Évaluation rapide de polynôme en une matrice)**.** On rappelle que le polynôme caractéristique d'une matrice arbitraire de $\mathcal{M}_n(\mathbb{K})$ peut se calculer en $O(\mathsf{MM}(n)\log n)$ opérations de $\mathbb{K}$. Le but de cet exercice est d'étudier la complexité arithmétique du problème suivant :

> $\mathcal{E}(P, A)$: étant donnés un polynôme $P = p_0 + p_1 X + \cdots + p_D X^D$ de $\mathbb{K}[X]$ et une matrice $A \in \mathcal{M}_n(\mathbb{K})$, calculer la matrice $P(A) = p_0 I_n + p_1 A + \cdots + p_D A^D$.

1. Montrer que $O(\mathsf{MM}(n)\sqrt{D})$ opérations de $\mathbb{K}$ suffisent pour résoudre $\mathcal{E}(P, A)$.
   Indication : Écrire $P(X)$ sous la forme $P_0(X) + P_1(X)X^d + P_2(X)(X^d)^2 + \cdots$, avec d bien choisi et $P_i(X)$ de degrés au plus $d-1$.
2. Montrer qu'on peut améliorer ce résultat à $O(\mathsf{MM}(n)\sqrt{n} + \mathsf{M}(D))$ opérations.
   Indication : commencer par calculer le polynôme caractéristique de A.
3. Montrer qu'on peut résoudre $\mathcal{E}(X^D, A)$ en $O(\mathsf{MM}(n)\sqrt{n} + \mathsf{M}(n)\log D)$ opérations de $\mathbb{K}$.
4. Montrer que si C est une matrice compagnon, alors il est possible de résoudre $\mathcal{E}(P, C)$ en $O(n^2 + \mathsf{M}(D))$ opérations de $\mathbb{K}$.
5. 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 $\mathbb{K}$.
6. Montrer que si $A \in \mathcal{M}_n(\mathbb{K})$ et $\deg(P) = n$, la première colonne de $P(A)$ peut être calculée en $O(\mathsf{MM}(n)\log n)$ opérations de $\mathbb{K}$.

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-ième colonne coïncide avec la première colonne de A^ℓ, est une matrice inversible.

7. Montrer que la matrice $C = UAU^{-1}$ est une matrice de type compagnon et qu'elle peut être calculée en $O(\mathsf{MM}(n)\log n)$ 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 $\mathcal{V}$.
8. En déduire que, pour une matrice A « suffisamment générique », le problème $\mathcal{E}(P, A)$ peut se résoudre en $O(\mathsf{MM}(n)\log n + \mathsf{M}(D))$ opérations de $\mathbb{K}$.
9. Montrer que, pour une matrice A « suffisamment générique », on peut résoudre $\mathcal{E}(X^D, A)$ en $O(\mathsf{MM}(n)\log n + \mathsf{M}(n)\log D)$ opérations de $\mathbb{K}$.

Exercice 16 (Somme de solutions d'équations différentielles linéaires)**.** Dans tout cet exercice, on considère deux équations différentielles

$$a_r(X)y^{(r)}(X) + \cdots + a_0(X)y(X) = 0 \tag{E1}$$

et

$$b_s(X)y^{(s)}(X) + \cdots + b_0(X)y(X) = 0 \tag{E2}$$

et on recherche une équation différentielle vérifiée par toutes les sommes $h = f + g$ où f est solution de (E1) et g une solution de (E2). On suppose que les a_i et les b_i sont des polynômes de $\mathbb{Q}[X]$ tels que $\deg a_i \le m$ et $\deg b_i \le n$, et qu'aucun de r, s, a_r et b_s n'est nul. On note $R = r + s$ et $D = \max\{m, n\}$.

Préambule

1. Montrer que h est solution d'une équation $\mathcal{R}(y) = 0$ d'ordre au plus R.

Les équations (E1) et (E2) permettent de récrire toute dérivée d'ordre arbitraire de f ou de g comme combinaison linéaire des $r + s$ fonctions f, …, $f^{(r-1)}$, g, …, et $g^{(s-1)}$. L'espace vectoriel sur $\mathbb{Q}(X)$ engendré par les dérivées d'ordre arbitraire f, f', g'', …, g, g', g'', …, a donc dimension au plus R.

Dans cet espace vectoriel, h, …, $h^{(R)}$ sont liées, et une relation de liaison non triviale fournit $\mathcal{R}$ après avoir chassé les dénominateurs.

Première méthode : recombinaison des équations

Pour $i \in \mathbb{N}$, $j \in \mathbb{N}$, $k \in \{1, 2\}$, on note $\mathcal{M}_{i,j}^{(k)}(y) = 0$ l'équation obtenue en dérivant j fois l'équation (k) puis en multipliant le résultat par X^i. On voit $\mathcal{M}_{i,j}^{(k)}$ comme une application de la variable y. Pour un entier $\delta \geq m + n$, on considère :

$$S^{(\mathrm{E1})} = \left\{ \mathcal{M}_{i,j}^{(\mathrm{E1})} : i \leq \delta - m,\ j \leq s \right\} \quad \text{et} \quad S^{(\mathrm{E2})} = \left\{ \mathcal{M}_{i,j}^{(\mathrm{E2})} : i \leq \delta - n,\ j \leq r \right\}.$$

2. Déterminer des bornes supérieures $d_{\max}$ et $r_{\max}$ telles que la famille des $X^i y^{(j)}$ pour $0 \leq i \leq d_{\max}$ et $0 \leq j \leq r_{\max}$ permette d'exprimer tous les $\mathcal{R}(y)$ quand $\mathcal{R}$ décrit $S^{(\mathrm{E1})} \cup S^{(\mathrm{E2})}$.
3. On veut résoudre

$$\text{(R)} \qquad \sum_{\mathcal{R} \in S^{(\mathrm{E1})}} \lambda_{\mathcal{R}} \mathcal{R}(y) = \sum_{\mathcal{R} \in S^{(\mathrm{E2})}} \mu_{\mathcal{R}} \mathcal{R}(y)$$

en des coefficients λ et μ dans $\mathbb{Q}$. En égalant les coefficients sur la base des $X^i y^{(j)}$, compter le nombre d'équations et le nombre d'inconnues. Choisir δ pour assurer l'existence de solutions non triviales.

4. Donner un algorithme pour le calcul d'une équation pour les sommes $h = f + g$ et justifier sa correction.
5. Évaluer en termes de R, D et avec des $O(\cdot)$ le coût arithmétique du calcul des $\mathcal{M}_{i,j}^{(k)}(y)$ de proche en proche et celui de la résolution du système. Conclure sur la complexité arithmétique de la méthode.

Deuxième méthode : réductions par les équations

6. Pour une solution f de (E1) et une combinaison linéaire $u = p_0 f + \cdots + p_{r-1} f^{(r-1)}$ à coefficients des polynômes de degré maximal δ, montrer que $a_r u'$ s'exprime comme une combinaison linéaire de f, …, $f^{(r-1)}$ pour des coefficients polynomiaux dont on bornera le degré en fonction de δ.
7. En déduire que pour tout entier $i \geq r - 1$, le produit $a_r^{i+1-r} f^{(i)}$ s'exprime comme une combinaison linéaire $p_{i,0} f + \cdots + p_{i,r-1} f^{(r-1)}$ pour des polynômes $p_{i,j}$ dont on bornera le degré.
8. Énoncer un résultat analogue pour $b_s^{i+1-s} g^{(j)}$ et pour des polynômes $q_{i,j}$.
9. On veut résoudre

$$\text{(P)} \qquad \sum_{i=0}^{R} c_i(X) f^{(i)} = 0 = \sum_{i=0}^{R} c_i(X) g^{(i)}$$

pour des polynômes c_i dans $\mathbb{Q}[X]$ communs aux deux équations. Montrer qu'en multipliant le membre de gauche de (P) par une puissance convenable de a_r et le membre de droite par une puissance convenable de b_s, et en réduisant par les équations (E1) et (E2), on obtient un système d'équations linéaires sur les c_i à coefficients polynomiaux. Indiquer les dimensions de ce système et borner le degré de ses coefficients en termes de R et D.

10. Donner un algorithme pour le calcul d'une équation pour les sommes $h = f + g$ et justifier sa correction.
11. Évaluer en termes de R, D et avec des $\tilde{O}(\cdot)$ (c'est-à-dire à des facteurs logarithmiques près) le coût arithmétique de cet algorithme.
12. Comparer avec la première méthode.

Notes

L'exercice 11 vient de [12]. Il montre que le produit d'opérateurs différentiels de bidegrés (n, n) en (X, θ) se ramène à un nombre constant de multiplications de matrices carées de taille n. L'article plus récent [5] montre que ces deux problèmes sont équivalents du point de vue de la complexité. L'exercice 2 est inspiré par l'article [2], qui montre que la prise de racine carrée $\mathsf{R}(n)$ d'une série tronquée à précision n vérifie aussi $O(\mathsf{R}) = O(\mathsf{M})$. L'exercice 12 est tiré de [3]. L'exercice 5 provient de [11]. L'exercice 9 est une adaptation de l'article [9]. L'exercice 16 est extrait de [4], l'exercice 8 de [6]. La première question de l'exercice 3 est tirée de [1], la seconde est implicite dans [8]. L'exerice 13 est inspiré par [10], l'exercice 4 est tiré de [7].

Bibliographie

[1] Aho (A. V.), Steiglitz (K.), and Ullman (J. D.). – Evaluating polynomials at fixed sets of points. *SIAM Journal on Computing*, vol. 4, n4, 1975, pp. 533–539.

[2] Alt (H.). – Square rooting is as difficult as multiplication. *Computing*, vol. 21, n3, 1978/79, pp. 221–232.

[3] Borwein (Jonathan M.) and Salvy (Bruno). – A proof of a recurrence for Bessel moments. *Experiment. Math.*, vol. 17, n2, 2008, pp. 223–230.

[4] Bostan (A.), Chyzak (F.), Li (Z.), and Salvy (B.). – Common multiples of linear ordinary differential and difference operators. – In preparation.

[5] Bostan (Alin), Chyzak (Frédéric), and Le Roux (Nicolas). – Products of ordinary differential operators by evaluation and interpolation. In *ISSAC'08*, pp. 23–30. – ACM, New York, 2008.

[6] Bostan (Alin), Flajolet (Philippe), Salvy (Bruno), and Schost (Éric). – Fast computation of special resultants. *Journal of Symbolic Computation*, vol. 41, n1, January 2006, pp. 1–29.

[7] Bostan (Alin), Gaudry (Pierrick), and Schost (Éric). – Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator. *SIAM J. Comput.*, vol. 36, n6, 2007, pp. 1777–1806.

[8] Bostan (Alin) and Schost (Éric). – Polynomial evaluation and interpolation on special sets of points. *Journal of Complexity*, vol. 21, n4, 2005, pp. 420–446.

[9] Brent (R. P.) and Traub (J. F.). – On the complexity of composition and generalized composition of power series. *SIAM Journal on Computing*, vol. 9, n1, 1980, pp. 54–66.

[10] Keller-Gehrig (Walter). – Fast algorithms for the characteristic polynomial. *Theoret. Comput. Sci.*, vol. 36, n2-3, 1985, pp. 309–317.

[11] Kung (H. T.) and Tong (D. M.). – Fast algorithms for partial fraction decomposition. *SIAM J. Comput.*, vol. 6, n3, 1977, pp. 582–593.

[12] van der Hoeven (Joris). – FFT-like multiplication of linear differential operators. *J. Symbolic Comput.*, vol. 33, n1, 2002, pp. 123–127.

Annexe : liste des algorithmes

La liste suivante contient les algorithmes décrits ou évoqués dans ce cours.

– Multiplication de polynômes par l'algorithme de Karatsuba ..... p. 96
– Transformée de Fourier rapide (FFT) ..... p. 99
– Multiplication de polynômes par transformée de Fourier discrète ..... p. 98
– Multiplication de polynômes par l'algorithme de Schönhage-Strassen ..... p. 101
– Multiplication de matrices par l'algorithme de Strassen ..... p. 112
– Inversion de matrices par l'algorithme de Strassen ..... p. 117
– Calcul de polynôme caractéristique par l'algorithme de Keller-Gehrig ..... p. 119
– Inverse de séries par l'algorithme de Sieveking-Kung ..... p. 130
– Inverse de matrices de séries par l'algorithme de Schulz ..... p. 131
– Logarithme de séries par l'algorithme de Brent ..... p. 131
– Exponentielle de séries par l'algorithme de Brent ..... p. 133
– Itération de Newton générale sur les séries ..... p. 132
– Conversion rapide *sommes de Newton* ↔ *coefficients* par l'algorithme de Schönhage . p. 133
– Somme et produit composés de polynômes par les algorithmes de Bostan-Flajolet-Salvy-Schost ..... p. 134
– Composition de séries par l'algorithme de Brent-Kung ..... p. 137
– Division de polynômes par l'algorithme de Strassen ..... p. 141
– Développement rapide en série de Taylor de fraction rationnelle : N-ième terme et N premiers termes ..... p. 143
– N-ième terme d'une suite récurrente linéaire à coefficients constants par l'algorithme de Fiduccia ..... p. 145
– N premiers termes d'une suite récurrente linéaire à coefficients constants par l'algorithme de Shoup ..... p. 145
– Évaluation multipoint par l'algorithme de Borodin-Moenck ..... p. 154
– Interpolation par l'algorithme de Borodin-Moenck ..... p. 155
– Évaluation multipoint sur des points en progression géométrique par l'algorithme de Bluestein p. 156
– Interpolation sur des points en progression géométrique par l'algorithme de Bostan-Schost p. 156
– Produit de matrices polynomiales par l'algorithme de Bostan-Schost ..... p. 157
– Algorithme d'Euclide ..... p. 160
– Algorithme d'Euclide étendu ..... p. 162
– Algorithme des sous-résultants ..... p. 169
– Algorithme de pgcd rapide de Lehmer-Knuth-Schönhage ..... p. 171
– Algorithme du demi-pgcd rapide de Lehmer-Knuth-Schönhage ..... p. 172
– Conversion rapide *équation différentielle* ↔ *récurrence* ..... p. 179
– Multiplication de séries D-finies ..... p. 180
– Calcul de résolvante différentielle par l'algorithme d'Abel-Harley ..... p. 181
– Algorithme de reconstruction rationnelle ..... p. 189
– Reconstruction de récurrences linéaires à coefficients constants par l'algorithme de Berlekamp-Massey ..... p. 190
– Interpolation de Cauchy des fractions rationnelles ..... p. 190
– Algorithme de reconstruction de récurrences hypergéométriques ..... p. 190
– Algorithmes rapides pour la reconstruction rationnelle, l'approximation de Padé et l'interpolation de Cauchy ..... p. 191

– Algorithmes rapides de reconstruction de récurrences hypergéométriques et de récurrences linéaires à coefficients constants ..... p. 191
– Calcul d'approximants de Padé-Hermite par l'algorithme de Derksen ..... p. 194
– Reconstruction d'équations algébriques et différentielles ..... p. 195
– Reconstruction de récurrences linéaires à coefficients polynomiaux ..... p. 195
– Calcul d'approximants de Padé-Hermite par l'algorithme de Beckermann-Labahn .... p. 196
– Résolution d'un système linéaire creux par l'algorithme de Wiedemann ..... p. 199
– Calcul du polynôme minimal d'une matrice creuse par l'algorithme de Wiedemann .. p. 200
– Résolution de systèmes linéaires quasi-Toeplitz par l'algorithme de Bitmead-Anderson-Morf p. 209
– Évaluation multipoint rapide par l'algorithme de Bostan-Lecerf-Schost ..... p. 221
– Calcul du N-ième terme d'une suite P-récursive par l'algorithme de Chudnovsky .... p. 229
– Factorisation déterministe d'entiers par l'algorithme de Strassen ..... p. 229
– Calcul du N-ième terme d'une suite P-récursive par scindage binaire ..... p. 230
– Calcul de e par l'algorithme de Brent ..... p. 231
– Résolution de systèmes linéaires à coefficients polynomiaux par l'algorithme de Storjohann p. 237
– Calcul de solutions séries d'équations différentielles d'ordre 1 par l'algorithme de Brent-Kung p. 241
– Calcul de solutions séries d'équations différentielles d'ordre arbitraire par les algorithmes de Bostan-Chyzak-Ollivier-Salvy-Schost-Sedoglavic ..... p. 243