Université Paris-Sud
École doctorale de Mathématiques de la Région Paris-Sud – ED 142

Algorithmique des polynômes à plusieurs variables : opérations élémentaires, factorisation, élimination

par Grégoire Lecerf

Laboratoire d'informatique de l'École polytechnique
LIX, UMR 7161 CNRS
Campus de l'École polytechnique
91128 Palaiseau Cedex

Courriel : gregoire.lecerf@math.cnrs.fr

Mémoire d'habilitation à diriger des recherches
Spécialité : Mathématiques

Date de soutenance : le 26 novembre 2013

Composition du jury

M. Antoine Chambert-Loir, Prof., Univ. Paris-Sud, directeur
M. Marc Giusti, DR CNRS, École polytechnique, examinateur
M. Dmitry Grigoryev, DR CNRS, Univ. Lille I, rapporteur
M. Bernard Mourrain, DR INRIA, Sophia Antipolis, rapporteur
Mme Bernadette Perrin-Riou, Prof., Univ. Paris-Sud, présidente du jury
Mme Marie-Françoise Roy, Prof. émérite, Univ. Rennes I, rapporteur

Table des matières

Introduction

Thématiques de recherche

Dans les paragraphes suivants sont présentés les thèmes de recherche sur lesquels j'ai travaillé ainsi que mes principaux résultats.

Opérations élémentaires sur les polynômes

Au fur et à mesure de l'évolution de mes travaux sur la résolution des systèmes algébriques et la factorisation des polynômes à plusieurs variables décrits ci-dessous, je me suis aperçu que les opérations arithmétiques de base sur les polynômes et séries devenaient critiques pour la complexité asymptotique et les performances pratiques. J'ai donc été naturellement conduit à étudier la complexité de la multiplication, la division, l'interpolation, l'évaluation, les restes chinois et le lemme de Hensel. Tout d'abord, en collaboration avec A. Bostan et É. Schost, nous avons répondu à une question ouverte depuis plusieurs années sur la complexité en espace mémoire du principe de transposition des circuits linéaires [35]. Nous avons illustré notre transformation de programme sur le produit et la division des polynômes, et avons validé notre approche en C++, en utilisant la bibliothèque logicielle NTL [263]. Ceci nous a aussi conduit à de nouveaux algorithmes plus rapides pour l'interpolation et l'évaluation des polynômes à une variable.

En collaboration avec É. Schost nous avons exhibé un algorithme de coût quasi-linéaire pour multiplier en caractéristique zéro des séries en représentation dense, tronquées en degré total [195]. Ensuite, avec J. van der Hoeven, nous avons analysé la complexité des méthodes par blocs pour le produit des polynômes et séries à plusieurs variables [143]. Enfin nous avons conçu de nouveaux algorithmes, plus efficaces en pratique, avec des coûts quasi-linéaires pour les corps de coefficients usuels [146].

Factorisation des polynômes

Le calcul de la décomposition en irréductibles des fermés algébriques aboutit naturellement à la factorisation irréductible des polynômes à plusieurs variables. Lorsque je me suis penché sur cette question, les algorithmes de factorisation existants avaient une complexité théorique bien plus élevée que celle que j'avais avec mon processus de décomposition équidimensionnelle [189]. J'ai d'abord étudié la réduction de la factorisation de deux à une variables, en représentation dense dans $\mathbb{K}[x, y]$, en caractéristique nulle ou positive suffisamment grande. La technique utilisée est celle de la remontée de Hensel, et la question centrale était, pour un polynôme donné, de déterminer la précision nécessaire en le degré de x pour recombiner les facteurs irréductibles dans $\mathbb{K}[[x]][y]$ en facteurs rationnels, c'est-à-dire dans $\mathbb{K}[x][y]$, avec une bonne complexité. Avec A. Bostan, B. Salvy, É. Schost et B. Wiebelt nous avons prouvé qu'une borne linéaire dans le degré total était suffisante [34]. J'ai ensuite amélioré cette borne dans l'article [190] et ai étendu la méthode à plusieurs variables [191]. Finalement j'ai développé une autre technique nécessitant une précision essentiellement optimale et permettant de traiter tous les cas de caractéristique positive [193].

Avant de pouvoir bénéficier d'un algorithme de remontée de Hensel rapide, il faut s'assurer que le polynôme à factoriser est bien séparable lorsque x est spécialisé à zéro. J'ai donné un algorithme de coût quasi-linéaire pour se ramener à cette situation [192]. J'ai aussi travaillé en collaboration avec J. Berthomieu sur le cas des polynômes creux à deux variables : nous avons montré comment obtenir rapidement une transformation monomiale de sorte à réduire le calcul au cas dense, et obtenir des bornes de complexité en terme du volume du polygone de Newton du polynôme à factoriser.

Enfin, en collaboration avec G. CHÈZE, nous avons abordé la question de la factorisation absolue des polynômes à deux variables sur un corps de caractéristique zéro ou suffisamment grande. Nous avons conçu un nouvel algorithme calquant les précédents qui a permis d'améliorer les bornes de complexité précédemment connues [49]. Plus récemment, dans le cadre de la thèse de doctorat de G. QUINTIN, j'ai aussi travaillé sur le calcul des racines d'un polynôme de $\mathrm{GR}(p^n, k)[x][y]$, où $\mathrm{GR}(p^n, k)$ représente l'anneau de Galois d'ordre $n\,k$ et de caractéristique p^n, défini, à isomorphisme près, comme $(\mathbb{Z}/p^n\,\mathbb{Z})[x](q(x))$ avec q de degré k irréductible modulo p [25]. Nos résultats permettent d'étendre l'algorithme de décodage en liste de V. GURUSWAMI et M. SUDAN [122] au cas des anneaux de Galois.

RÉSOLUTION DES SYSTÈMES ALGÉBRIQUES

Introduite par H. HIRONAKA au milieu des années soixante, la notion de base standard d'un idéal dans un anneau de polynômes connaît, depuis les travaux de B. BUCHBERGER, un engouement particulier en mathématiques et en informatique. La construction effective d'une telle base est désormais une fonctionnalité essentielle de tous les logiciels de calcul formel. Au cœur de ces méthodes, les polynômes à plusieurs variables sont représentés comme vecteurs de leurs coefficients dans la base monomiale usuelle et chaque opération élémentaire d'élimination consiste en une opération du type pivot. Néanmoins le coût de ces méthodes est doublement exponentiel dans le pire des cas. Il faut attendre les années quatre-vingt dix pour que M. GIUSTI et J. HEINTZ développent l'idée que les problèmes d'élimination peuvent être ramenés dans une classe de complexité polynomiale, en représentant les polynômes issus de l'élimination par des calculs d'évaluation. C'est dans cette voie que se sont situés principalement mes premiers travaux de recherche.

À l'issue de ma thèse de doctorat, j'avais publié un algorithme pour calculer la décomposition en composantes équidimensionnelles de la clôture de Zariski de l'ensemble des solutions d'un système d'équations et d'inéquations polynomiales [189]. Sa complexité était polynomiale en un degré propre aux méthodes incrémentales et est toujours la meilleure connue actuellement. Dans le cas de systèmes génériques d'équations de degré deux à coefficients entiers, le coût devient même quasi-linéaire dans la taille de la sortie. J'avais implanté cet algorithme, dit de résolution géométrique, dans le système de calcul formel MAGMA. Ce logiciel a été appelé KRONECKER, en hommage à l'illustre mathématicien pour ses travaux sur l'élimination.

En collaboration avec C. DURVYE, dans le cadre de sa thèse de doctorat, nous avons étendu mon algorithme de décomposition équidimensionnelle au calcul de la décomposition primaire d'un idéal d'un anneau de polynômes. Dans un premier temps nous avons restructuré et simplifié considérablement les preuves de correction de l'algorithme de résolution géométrique [70]. Ensuite C. DURVYE a obtenu une première extension de l'algorithme pour le calcul des composantes primaires isolées de dimension zéro [69].

Plus récemment, j'ai rédigé une version de l'algorithme de résolution géométrique pour le cours *Algorithmes efficaces en calcul formel* dans lequel je suis intervenu en 2013 au Master Parisien de Recherche en Informatique (MPRI) [33]. J'ai aussi réalisé une implantation en C++ de l'algorithme de résolution géométrique au sein de MATHEMAGIX [147], appelé GEOMSOLVEX. Ce travail a été motivé d'une part par le besoin de pouvoir maîtriser les sous-algorithmes de bas niveau, et d'autre part par le souhait de rendre facilement accessible une implantation de cet algorithme indépendante de logiciels commerciaux.

En collaboration avec B. BANK, M. GIUSTI, J. HEINTZ, G. MATERA et P. SOLERNÓ, nous venons d'étendre l'algorithme de résolution géométrique au calcul des lieux de dégénérescence de fibrés vectoriels algébriques définis sur une variété lisse [8]. Un cas

particulier important concerne les variétés polaires, utiles pour des calculs en géométrie algébrique réelle, pour lesquelles nous avons prouvé de nouvelles bornes de complexité. Cette extension est aussi programmée dans la bibliothèque geomsolvex.

Traitement numérique des grappes de racines

D'une façon un peu différente des méthodes symboliques traditionnelles, j'ai poursuivi la voie ouverte par S. Smale et ses collaborateurs, appelée la théorie des zéros approchés. Cette théorie ne concerne que les zéros simples puisqu'elle repose sur l'utilisation de l'opérateur de Newton. En collaboration avec M. Giusti, B. Salvy et J.-C. Yakoubsohn, nous avons porté nos efforts sur la localisation et l'approximation des zéros multiples. Nous avons dégagé un cadre technique original et efficace pour manipuler des objets analytiques par le calcul symbolique tout en préservant des représentations de nombres complexes en multiprécision. Ensuite nous avons montré comment localiser et approcher des grappes de zéros de fonctions analytiques à une variable [108] et de certaines applications analytiques à plusieurs variables [109]. Nos algorithmes d'approximation sont numériquement stables et, en présence d'une grappe de zéros de diamètre strictement positif, les suites des approximations s'arrêtent à une distance de la grappe que nous prouvons être de l'ordre du diamètre de celle-ci. En particulier l'algorithme calcule un encadrement numérique du diamètre.

Mathemagix

Selon la thèse de Church tous les systèmes de programmation usuels sont Turing complets. En théorie il n'y a donc pas de raison de privilégier un langage plutôt qu'un autre. Néanmoins dans les faits, la situation devient très nuancée dès qu'il s'agit de développer des logiciels à une large échelle dans une thématique particulière. Pour les calculs symboliques ou analytiques, il est primordial que le langage permette d'écrire les algorithmes de la façon la plus naturelle et lisible possible. Il est aussi crucial que le langage puisse être compilé pour des raisons d'efficacité. Pour un usage de type calculatrice il est aussi souhaitable de disposer d'un interprète de commandes et d'une interface graphique de qualité pour afficher les formules mathématiques et interagir avec les objets graphiques. C'est exactement dans cet esprit que la plateforme logicielle Mathemagix [147] a été initiée par J. van der Hoeven et continue d'évoluer. Je participe à ce projet depuis 2005 et suis auteur d'un grand nombre d'implantations d'algorithmes sur les polynômes et matrices avec des coefficients symboliques. Les résultats obtenus font l'objet du dernier chapitre de ce mémoire.

Co-encadrements de thèses de doctorat

1. C. Durvye, *Algorithmes efficaces pour calculer la multiplicité d'une composante isolée solution d'un système d'équations polynomiales*. Thèse de doctorat soutenue le 9 juin 2008, à l'université de Versailles Saint-Quentin-en-Yvelines, sous la direction de V. Cossart et la codirection de G. Lecerf.
2. J. Berthomieu, *Contributions à la résolution des systèmes algébriques : réduction, localisation, traitement des singularités, implantations*. Thèse de doctorat soutenue le 6 décembre 2011, à l'École polytechnique, sous la direction de M. Giusti et la codirection de G. Lecerf.
3. G. Quintin, *Sur l'algorithme de décodage en liste de Guruswami-Sudan sur les anneaux finis*. Thèse de doctorat soutenue le 22 novembre 2012, à l'École polytechnique, sous la direction de D. Augot et la codirection de G. Lecerf.

Productions scientifiques

Les résultats scientifiques présentés dans ce mémoire sont issus des productions listées ci-dessous, pour la plupart disponibles depuis http://lecerf.perso.math.cnrs.fr.

Articles dans des revues à comité de lecture

1. M. Giusti, K. Hägele, G. Lecerf, J. Marchand, B. Salvy, *The projective Noether Maple package: computing the dimension of a projective variety*. Journal of Symbolic Computation, vol. 30, n°3, p. 291–307, 2000.
2. M. Giusti, G. Lecerf, B. Salvy, *A Gröbner free alternative for polynomial system solving*. Journal of Complexity, vol. 17, n°1, p. 154–211, 2001.
3. G. Lecerf, *Quadratic Newton iteration for systems with multiplicity*. Foundations of Computational Mathematics, vol. 2, n°3, p. 247–293, 2002.
4. G. Lecerf, *Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers*. Journal of Complexity, vol. 19, n°4, p. 564–596, 2003.
5. G. Lecerf, É. Schost, *Fast multivariate power series multiplication in characteristic zero*. SADIO Electronic Journal on Informatics and Operations Research, vol. 5, n°1, p. 1–10, 2003.
6. M. Giusti, G. Lecerf, B. Salvy, J.-C. Yakoubsohn, *On location and approximation of clusters of zeros of analytic functions*. Foundations of Computational Mathematics, vol. 5, n°3, p. 257–311, 2005.
7. G. Lecerf, *Sharp precision in Hensel lifting for bivariate polynomial factorization*. Mathematics of Computations, vol. 75, p. 921–933, 2006.
8. M. Giusti, G. Lecerf, B. Salvy, J.-C. Yakoubsohn. *On location and approximation of clusters of zeros: Case of embedding dimension one*. Foundations of Computational Mathematics, vol. 7, n°1, p. 1–49, 2007.
9. C. Durvye, G. Lecerf, *A concise proof of the Kronecker polynomial system solver from scratch*. Expositiones Mathematicae, vol. 26, n°2, p. 101–139, 2007.
10. G. Lecerf, *Improved dense multivariate polynomial factorization algorithms*. Journal of Symbolic Computation, vol. 42, n°4, p. 477–494, 2007.
11. G. Chèze, G. Lecerf, *Lifting and recombination techniques for absolute factorization*. Journal of Complexity, vol. 23, n°3, p. 380–420, 2007.
12. G. Lecerf, *Fast separable factorization and applications*. Applicable Algebra in Engineering, Communication and Computing, vol. 19, n°2, p. 135–160, 2008.
13. G. Lecerf. *New recombination algorithms for bivariate polynomial factorization based on Hensel lifting*, Applicable Algebra in Engineering, Communication and Computing, vol. 21, n°2, p. 151–176, 2010.
14. J. Berthomieu, J. van der Hoeven, G. Lecerf, *Relaxed algorithms for p-adic numbers*. Journal de théorie des nombres de Bordeaux, vol. 23, n°3, p. 541–577, 2011.
15. J. van der Hoeven, G. Lecerf, B. Mourrain, Ph. Trébuchet, J. Berthomieu, D. Diatta, A. Mantzaflaris, *Mathemagix, the quest of modularity and efficiency for symbolic and certified numeric computation*. ACM SIGSAM Communications in Computer Algebra, vol. 177, n°3, 2011. Dans la section "ISSAC 2011 Software Demonstrations", éditée par M. Stillman, p. 166–188.
16. J. Berthomieu, G. Lecerf, *Reduction of bivariate polynomials from convex-dense to dense, with application to factorizations*. Mathematics of Computations, vol. 81, n°279, p. 1799–1821, 2012.
17. J. van der Hoeven, A. Grozin, M. Gubinelli, G. Lecerf, F. Poulain, D. Raux, *GNU TEX_{MACS}: a scientific editing platform*. ACM SIGSAM Communications in Computer Algebra, vol. 47, n°2, p. 59–62. Actes de la session des démonstrations logicielles de la conférence ISSAC 2012.

18. J. van der Hoeven, G. Lecerf, *On the bit-complexity of sparse polynomial and series multiplication*. Journal of Symbolic Computation, vol. 50, p. 227–254, 2013.
19. J. Berthomieu, G. Lecerf, G. Quintin, *Polynomial root finding over local rings and application to error correcting codes*. Applicable Algebra in Engineering, Communication and Computing, 2013. http://dx.doi.org/10.1007/s00200-013-0200-5

Articles dans des actes de conférence avec comité de lecture

1. G. Lecerf, *Computing an equidimensional decomposition of an algebraic variety by means of geometric resolutions*. In C. Traverso (éditeur), Proc. ISSAC 2000, p. 209–216, ACM Press, 2000.
2. A. Bostan, G. Lecerf, É. Schost, *Tellegen's principle into practice*. In Hoon Hong (éditeur), Proc. ISSAC 2003, p. 37–44, ACM Press, 2003.
3. A. Bostan, G. Lecerf, B. Salvy, É. Schost, B. Wiebelt, *Complexity issues in bivariate polynomial factorization*. In J. Schicho (éditeur), Proc. ISSAC 2004, p. 42–49, ACM Press, 2004.
4. A. Bostan, F. Chyzak, G. Lecerf, B. Salvy, É. Schost, *Differential equations for algebraic functions*. In Dongming Wang, Proc. ISSAC 2007, p. 25–32, ACM Press, 2007.
5. G. Lecerf, *Mathemagix: towards large scale programming for symbolic and certified numeric computations*. In K. Fukuda, J. van der Hoeven, M. Joswig, N. Takayama (éditeurs), Mathematical Software – ICMS 2010, Third International Congress on Mathematical Software, 2010, vol. 6327 de Lecture Notes in Computer Science, p. 329–332, Springer, 2010.
6. J. van der Hoeven, G. Lecerf, *On the complexity of blockwise polynomial multiplication*. In J. van der Hoeven (éditeur), Proc. ISSAC 2012, p. 211–218, ACM Press, 2012.
7. J. van der Hoeven, G. Lecerf, *Interfacing Mathemagix with C++*. In M. Monagan, G. Cooperman, M. Giesbrecht, Proc. ISSAC 2013, p. 363–370, ACM Press, 2013.

Chapitre de livre

1. E. Kaltofen, G. Lecerf, *Handbook of Finite Fields*, chapitre « Factorization of multivariate polynomials » de G. L. Mullen, D. Panario, p. 382–392, CRC Press, 2013.

Logiciels

1. *The Dynamic Real Closure*, paquetage en `Axiom` de 5000 lignes, écrit à l'université de Bath (UK) lors mon stage de DEA, 1996. *Non maintenu.*
2. *The Projective Noether Package*, paquetage en `Maple` offrant diverses stratégies de calcul de la dimension d'une variété projective [100]. Réalisé au laboratoire GAGE de l'École polytechnique, 1997. *Non maintenu.*
3. *Kronecker*, en Magma, résolution des systèmes algébriques [107, 187, 188, 189].
4. *mml4mgm*, avec X. Suraud, support MathML pour Magma, 2002. *Non maintenu.*
5. *Tellegen for* NTL, en C++, à l'aide de la bibliothèque NTL, opérations transposées de la multiplication, division, évaluation et interpolation des polynômes à une variable [34].
6. *Separable factorization*, en Magma, décomposition séparable des polynômes à une ou deux variables [192].

7. *absfact.lib*, inclus dans la distribution standard de SINGULAR pour le calcul de la factorisation absolue des polynômes à plusieurs variables, septembre 2005. Il s'agit d'un travail en collaboration avec W. DECKER et G. PFISTER.
8. *Absolute factorization*, en MAGMA, en collaboration avec G. CHÈZE, factorisation absolue des polynômes à deux variables [49].
9. *Mode Emacs pour le langage Mathemagix*, en EMACS LISP, pour faciliter l'écriture de programmes dans le langage MATHEMAGIX en proposant du coloriage syntaxique, de l'indentation automatique et de la navigation syntaxique.
10. Bibliothèque NUMERIX de MATHEMAGIX, en collaboration avec J. VAN DER HOEVEN. J'y ai programmé l'arithmétique modulaire pour les entiers.
11. Bibliothèque ALGEBRAMIX de MATHEMAGIX, en collaboration avec J. VAN DER HOEVEN. J'y ai programmé de nombreux algorithmes pour les polynômes, matrices et series [22, 35].
12. Bibliothèque LATTIZ de MATHEMAGIX, algorithmes de reduction de réseaux, avec l'utilisation optionnelle de FPLLL [275].
13. Bibliothèque MULTIMIX de MATHEMAGIX, en collaboration avec J. VAN DER HOEVEN. J'y ai programmé plusieurs algorithmes pour les opérations élémentaires sur les polynômes et séries à plusieurs variables en représentations denses et creuses [144, 146, 195].
14. Bibliothèque FINITEFIELDZ de MATHEMAGIX, en collaboration avec G. QUINTIN, calculs dans les corps finis [25].
15. Bibliothèque FACTORIX de MATHEMAGIX, factorisation des polynômes.
16. Bibliothèque GEOMSOLVEX de MATHEMAGIX, résolution des systèmes algébriques par des méthodes géométriques [8, 70].
17. Bibliothèque MFGB, pour utiliser FGB [74] depuis MATHEMAGIX.
18. Bibliothèque MPARI, pour utiliser PARI [235] depuis MATHEMAGIX.
19. Bibliothèque MBLAD, pour utiliser BLAD [36] depuis MATHEMAGIX.
20. Interprète pour le langage MATHEMAGIX, en collaboration avec J. VAN DER HOEVEN, dans le sous-projet MMCOMPILER [144].
21. AUTOMAGIX, en collaboration avec J. VAN DER HOEVEN et B. MOURRAIN, scripts pour produire et maintenir les fichiers de description des sous-projets de MATHEMAGIX pour le GNU *build system* aussi appelé AUTOTOOLS (`http://www.gnu.org`).
22. Participations ponctuelles à GNU TEXMACS : interface pour DOYXYGEN, interface avec MATHEMAGIX, style ACM pour la conférence ISSAC [141].

PRÉPUBLICATION

1. B. BANK, M. GIUSTI, J. HEINTZ, G. LECERF, G. MATERA, P. SOLERNÓ, *Degeneracy loci and polynomial equation solving*, 23 pages, 2013. Disponible depuis `http://www2.mathematik.hu-berlin.de/publ/pre/2013/P-13-08.pdf`.

THÈSE DE DOCTORAT

1. G. LECERF, *Une alternative aux méthodes de réécriture pour la résolution des systèmes algébriques*, soutenue le 14 septembre 2001 à l'École polytechnique.

SUPPORTS DE COURS ET MANUELS

1. J. VAN DER HOEVEN, G. LECERF, *Mathemagix user guide*, 101 pages, HAL, 2013. Disponible depuis `http://hal.archives-ouvertes.fr/hal-00785549`.
2. A. BOSTAN, F. CHYZAK, M. GIUSTI, R. LEBRETON, G. LECERF, B. SALVY, É. SCHOST, *Algorithmes efficaces en calcul formel*. Notes du cours 2-22 du MPRI (Master Parisien de Recherche en Informatique, Université Denis-Diderot, ENS Paris, ENS Cachan, École polytechnique), 333 pages, version de 2013.

3. G. Lecerf, *Factorisation des polynômes à plusieurs variables*. Support de cours des Journées Nationales de Calcul Formel 2013, Les Cours du CIRM, vol. 3, cours n° 2, p. 1–85, Cedram, 2013.

Remerciements

Je tiens à remercier le CNRS, l'Université de Versailles Saint-Quentin-en-Yvelines, l'Université indépendante de Moscou, et l'École polytechnique pour tous leurs soutiens matériels et financiers. Les financements attribués par l'ANR et le réseau thématique de recherche avancée Digiteo dont j'ai pu bénéficier ont aussi contribué à soutenir mes activités de recherche et je leur en suis reconnaissant. Je remercie aussi le GDS Mathrice du CNRS pour leur *Plateforme en Ligne pour les Mathématiques* (http://plm.math.cnrs.fr), ainsi que les concepteurs du logiciel GNU TEXMACS (http://www.texmacs.org) avec lequel le présent document a été rédigé.

Chapitre I

Opérations élémentaires sur les polynômes et séries

Dans ce chapitre nous présentons les résultats de complexité que nous avons obtenus concernant les opérations élémentaires sur les polynômes et séries à plusieurs variables. Nous commençons par décrire brièvement le modèle de calcul utilisé pour mesurer le coût des algorithmes tout au long de ce mémoire. Ensuite nous rappelons quelques résultats de complexité pour l'évaluation et l'interpolation des polynômes à une variable, qui constituent les briques de base de nos produits rapides à plusieurs variables.

I.1 Modèle de calcul

Le choix d'un modèle de calcul pour présenter et analyser des algorithmes est une question épineuse. Considérer un modèle trop proche des ordinateurs actuels, qui peuvent effectuer plusieurs tâches en parallèle, et qui ont une gestion de la mémoire par niveaux de cache, est bien trop compliqué. À l'inverse, considérer un modèle très simple comme les machines de Turing permet de bien formaliser la notion de programme et de complexité, mais en s'éloignant des architectures matérielles actuelles. Les arbres de calcul que nous allons utiliser sembleront aussi éloignés de la réalité. Mais nous nous efforcerons de décrire les algorithmes dans un langage naturel tout en gardant en tête, d'un point de vue théorique, qu'une telle description explicite la construction d'arbres de calcul pour lesquels nous pouvons facilement énoncer des résultats avec rigueur.

Pour sûr, le travail nécessaire pour passer d'un algorithme à une implantation efficace est souvent important, surtout pour des opérations qui semblent simples comme le produit de polynômes ou de matrices. Ce travail consiste en effet à concevoir de bonnes structures de données, à développer des stratégies d'utilisation de la mémoire favorables à l'architecture utilisée, et enfin à s'assurer que le code exécutable produit par le compilateur exploite correctement les ressources matérielles de calcul.

Cette section reprend les concepts sur les arbres de calcul en suivant la présentation faite dans [38, Chapter 4]. Pour une présentation plus détaillée avec des exemples, je renvoie le lecteur aux notes de mon cours donné lors des Journées Nationales de Calcul Formel en 2013 [194, Chapitre 1].

I.1.1 Syntaxe d'un arbre de calcul

Si $\mathbb{K}$ est un corps, nous définissons $\mathbb{K}^c := \{\mathbb{K}^0 \to \mathbb{K}, () \mapsto a \mid a \in \mathbb{K}\}$ comme l'ensemble des *fonctions d'arité zéro*, et notons Id la fonction identité sur $\mathbb{K}$. Cet ensemble $\mathbb{K}^c$ est isomorphe à $\mathbb{K}$, mais il sera plus confortable par la suite de considérer les constantes comme des éléments de $\mathbb{K}^c$. L'ensemble des *opérations élémentaires* Ω sur $\mathbb{K}$ est $\Omega := \mathbb{K}^c \cup \{\mathrm{Id}, +, -, \times, /\}$. L'ensemble des *prédicats* P est quant à lui défini par $P := \{=, \neq\}$. Plus généralement nous serons amenés à considérer des structures algébriques $\mathbb{A}$ qui ne seront pas des corps et nous adapterons naturellement ces définitions. Par exemple, si $\mathbb{A}$ est un anneau, les opérations élémentaires seront $\mathbb{A}^c \cup \{\mathrm{Id}, +, -, \times\}$. Éventuellement rien n'empêche, si $\mathbb{A}$ est intègre, d'y adjoindre la division exacte. Si $\mathbb{A}$ est ordonné, alors l'ensemble des prédicats peut aussi contenir les tests $\{<, \leqslant, \geqslant, >\}$. Il conviendra, pour chaque énoncé, si le contexte n'est pas clair, de préciser les opérations élémentaires et prédicats utilisés.

Définition I.1. *Un* arbre binaire *est un graphe acyclique orienté dont tous les nœuds sauf un, appelé la* racine*, a un parent et au plus deux fils.*

Un arbre binaire induit un *ordre partiel naturel* $\prec$ sur l'ensemble de ses nœuds, pour lequel la racine est le plus petit élément : $u \prec v \Leftrightarrow u$ est le grand-$\cdots$-grand-parent de v.

Définition I.2. *Un* chemin *est une suite finie de nœuds $v_0, v_1, ..., v_l$ tels que v_i est le parent de v_{i+1} pour tout $i \in \{0, ..., l-1\}$.*

Définition I.3. *Un* segment initial *de longueur n d'un arbre binaire est un ensemble de nœuds $\mathcal{I} = \{v_1, ..., v_n\}$ tel que $v_1 \prec v_2 \prec \cdots \prec v_n$, et tel qu'il existe un nœud u qui est le successeur de v_n et le prédécesseur de tous les nœuds qui ne sont pas dans $\mathcal{I} \cup \{u\}$.*

Soit T un arbre binaire et $\mathcal{I}$ l'un de ses segments initiaux de longueur n. Les éléments de $\mathcal{I}$ sont appelés les *nœuds d'entrée*. L'ensemble des nœuds hors de $\mathcal{I}$ est partitionné selon leur degré sortant d :

- les *nœuds de sortie* $\mathcal{O}$, qui correspondent à $d = 0$,
- les *nœuds de calcul* $\mathcal{C}$, qui correspondent à $d = 1$,
- les *nœuds de branchement* $\mathcal{B}$, qui correspondent à $d = 2$.

Définition I.4. *Un* arbre de calcul *sur une structure $\mathbb{A}$ munie des opérations élémentaires Ω, des prédicats P, et avec une entrée de taille n, est un arbre binaire muni :*

1. *d'une* fonction d'instruction *qui assigne :*
   - *à chaque nœud de calcul v une* instruction de calcul *de la forme $(\omega; u_1, ..., u_m)$, où $\omega \in \Omega$ est d'arité m, et $u_1, ..., u_m \in \mathcal{I} \cup \mathcal{C}$ sont des prédécesseurs de v,*
   - *à chaque nœud de branchement v une* instruction de test *de la forme $(\rho; u_1, u_2)$, où $\rho \in P$ et $u_1, u_2 \in \mathcal{I} \cup \mathcal{C}$ sont des prédécesseurs de v,*
   - *à chaque nœud de sortie v une* instruction de sortie *de la forme $(u_1, ..., u_m)$, où $m \in \mathbb{N}$ et $u_1, ..., u_m \in \mathcal{I} \cup \mathcal{C}$ sont des prédécesseurs de v.*
2. *d'une* partition σ *de l'ensemble des nœuds de sortie telle que la longueur des nœuds de sortie soit constante sur chaque partie de σ.*

I.1.2 Sémantique d'un arbre de calcul

Soit $e \in \mathbb{A}^n$ une *valeur d'entrée* de l'arbre de calcul T de segment initial $\mathcal{I}$ de longueur n. Nous construisons le chemin T_e dans T ainsi que les fonctions suivantes, en commençant par la racine de T :

- $\alpha : \mathcal{I} \cup \{\text{nœuds de calcul de } T_e\} \to \mathbb{A}$,
- $\beta : \{\text{nœuds de branchement de } T_e\} \to \{\text{faux}, \text{vrai}\}$.

Nous commençons par définir $T_e := \mathcal{I} = (v_1, ..., v_n)$ et $\alpha : \mathcal{I} \to \mathbb{A}$, $\alpha(v_i) := e_i$ pour tout $i \in \{1, ..., n\}$. Par récurrence, supposons le chemin construit jusqu'à un certain nœud w. Si w n'est pas un nœud de sortie alors le chemin se prolonge de la façon suivante :

- si w est un nœud de branchement, alors son successeur est son fils gauche ou droit selon la valeur $\beta(w)$;
- si w est un nœud de calcul, ou si $w = v_n$, alors son successeur v est défini comme étant son fils.

Maintenant le successeur v connu, le chemin est prolongé avec v et les fonctions α et β sont prolongées de la façon suivante :

- si v est un nœud de branchement de la forme $(\rho; u_1, u_2)$, alors $\beta(v) := \rho(\alpha(u_1), \alpha(u_2))$,
- si v est un nœud de calcul de la forme $(\omega; u_1, ..., u_m)$, alors si $\omega(\alpha(u_1), ..., \alpha(u_m))$ est défini nous posons $\alpha(v) := \omega(\alpha(u_1), ..., \alpha(u_m))$, sinon le chemin s'arrête à w.

Si T_e s'arrête sur un nœud de sortie, alors T est dit *exécutable* sur e. Si $(u_1, ..., u_m)$ est un nœud de sortie de v, alors $(\alpha(u_1), ..., \alpha(u_m))$ est appelée la *valeur de sortie* de T sur l'entrée e. La partie de σ qui contient v est appelée la *classe de sortie* de e.

Si J est un sous-ensemble de $\mathbb{A}^n$, nous dirons que T est exécutable sur J si, et seulement si, T est exécutable sur chaque élément de J. La partition $\sigma = (\sigma_i)_{i\in\{1,\dots,t\}}$ des nœuds de sortie induit la partition $\pi := \{J_1, \dots, J_t\}$ de J définie par :

$$J_i := \{e \in J \mid \text{la classe de sortie de } e \text{ est } \sigma_i\}.$$

Pour chaque i, nous définissons la fonction $\varphi_i \colon J_i \to \mathbb{A}^{m_i}$ qui retourne la valeur de sortie pour chaque $e \in J_i$. L'unique extension, notée φ, des φ_i définit une fonction sur J qui est appelée la fonction d'évaluation de T sur J.

Définition I.5. *Une* collection *pour un sous-ensemble J de $\mathbb{A}^n$ est la donnée d'une partition $\pi = \{J_1, \dots, J_t\}$ de J et d'une famille de fonctions $\varphi_i \colon J_i \to \mathbb{A}^{m_i}$ pour chaque $i \in \{1, \dots, t\}$. On représente une telle collection par le couple (φ, π) où φ est l'unique fonction qui prolonge les φ_i sur J.*

Un arbre de calcul définit donc une collection naturelle induite par sa fonction d'évaluation.

I.1.3 Fonctions de coût

Continuons de considérer une structure algébrique munie des opérations élémentaires Ω et des prédicats P. Une fonction $c \colon \Omega \cup P \to \mathbb{N}$ est appelée une *fonction de coût*. Nous prendrons dans la suite c constante de valeur 1 : il s'agit de la fonction de *coût total*.

Soit T un arbre de calcul. Le *coût d'un nœud de sortie v* est obtenu en additionnant les coûts des instructions rencontrées sur le chemin partant de la racine et arrivant jusqu'à v. De la sorte, nous construisons une fonction de coût sur l'ensemble des nœuds de sortie de T. Si T est exécutable sur J, la fonction $t \colon J \to \mathbb{N}$, où $t(e)$ est le coût du nœud de sortie sur lequel se termine le chemin T_e, est appelée la fonction de coût sur J.

Définition I.6. *Soit (φ, π) une collection associée à J, soit c une fonction de coût, et soit $\tau \colon J \to \mathbb{N}$. On dit que (φ, π) est* calculable *avec un coût τ lorsqu'il existe un arbre de calcul T qui calcule (φ, π) avec un coût borné par τ sur J.*

La complexité dans le pire cas*, notée $C^c(\varphi, \pi)$, pour une collection (φ, π) est définie comme le plus petit entier naturel r tel que (φ, π) est calculable avec le coût $\tau \colon J \to \mathbb{N}$, $a \mapsto r$.*

Nous dirons qu'un arbre calcule une collection (φ, π) en temps $\tau \colon J \to \mathbb{N}$ si, et seulement si, il a un coût au plus τ sur J. Finalement, dans ce formalisme, un *algorithme* est simplement une description en langage naturel de la façon dont on peut construire une famille d'arbres de calcul pour résoudre un problème donné. Lorsque nous parlerons du coût d'un algorithme, il s'agira du coût des arbres de calcul qu'il décrit. Tout comme les fonctions de coût nous pourrions définir des fonctions d'allocation de mémoire qui nous permettraient d'analyser la consommation mémoire des calculs, mais nous n'aborderons pas ces questions ici.

Dès lors que les fonctions de coût deviennent compliquées, il est d'usage de simplifier les expressions en utilisant les notations suivantes :

Définition I.7. *Si f et g sont deux fonctions de $\mathbb{N}$ dans $\mathbb{R}_+$ définies au voisinage de l'infini, l'écriture $f(n) \in O(g(n))$ signifie qu'il existe deux constantes N et C telles que $f(n) \leqslant C\, g(n)$ pout tout $n \geqslant N$. De plus, nous écrirons $f(n) \in \tilde{O}(g(n))$ lorsque $f(n) \in g(n) \log_2(3 + g(n))^{O(1)}$.*

Exemple I.8. Si $\alpha > 0$, on a $n \log^\alpha n \log\log n \in \tilde{O}(n)$.

Si $f(n) \in \tilde{O}(n)$, alors f est dite au plus *quasi-linéaire* ; si $f(n) \in \tilde{O}(n^2)$, alors f est dite au plus *quasi-quadratique*, etc.

I.1.4 Algorithmes probabilistes

Il existe principalement deux types d'algorithmes *probabilistes*, que nous utiliserons dans ce mémoire. Le premier type, appelé fréquemment *randomized* en anglais, concerne l'utilisation de générateurs de nombres aléatoires pendant les calculs. Le résultat retourné est toujours correct, mais le coût dépend des choix lors de l'exécution. Avec le point de vue des arbres de calcul, un tel algorithme est la description d'une famille d'arbres où les variables aléatoires sont considérées comme des entrées. Pour une instance de problème donnée, le *coût moyen* de l'arbre est alors la moyenne des coûts lorsque les variables aléatoires parcourent toutes les valeurs possibles (pouvant éventuellement dépendre de l'instance). Les coûts liés à la génération de nombres aléatoires seront par conséquent ignorés. L'exemple classique est l'algorithme *quicksort* dont le coût dans le pire des cas est quadratique, mais qui possède une variante en temps moyen quasi-linéaire. Nous rencontrerons principalement ces analyses de coût moyen pour les problèmes de factorisation des polynômes dans le chapitre suivant.

Le deuxième type d'algorithmes probabilistes est aussi amené à faire des choix en court d'exécution mais certains de ces choix conduisent à des résultats erronés ou des erreurs d'exécution. En terme d'arbres de calcul, les variables aléatoires sont aussi considérées comme des entrées, mais nous nous attacherons à borner le *coût dans le pire cas*, défini, pour une instance de problème, comme le maximum des coûts de l'arbre lorsque les variables aléatoires parcourent toutes les valeurs possibles (pouvant éventuellement dépendre de l'instance). Pour une instance de problème donnée, nous analyserons aussi la probabilité que le calcul aboutisse et que le résultat soit correct. Ce type d'algorithme interviendra dans le chapitre III.

I.1.5 Fonctions de produit

Dans la suite de ce mémoire nous analyserons fréquemment le coût de nos algorithmes en fonction du coût du produit des polynômes à une variable. Sauf mention explicite du contraire, nous considérerons qu'un polynôme de degré d à une variable est représenté par le vecteur de ses $d+1$ coefficients, du degré 0 au degré d. Afin d'énoncer les résultats d'une façon flexible, nous utiliserons une fonction de coût $\mathsf{M}\colon \mathbb{N} \to \mathbb{N}$ bornant le coût de l'algorithme choisi pour calculer le produit de deux polynômes à coefficients dans un anneau commutatif unitaire. Pour des raisons techniques, nous supposerons que M satisfait les propriétés suivantes :

- $\mathsf{M}(d)/d$ est croissante,
- $\mathsf{M}(m\,d) \leqslant m^2\,\mathsf{M}(d)$ pour tout $m \geqslant 1$.

Rappelons qu'il existe des algorithmes permettant de choisir $\mathsf{M}(n) \in \tilde{O}(n)$ [42, 52, 135, 251].

Nous supposons tout au long de ce mémoire que les entiers sont représentés en base 2. La taille d'un entier est définie comme étant le nombre de bits utilisés dans cette représentation. En terme d'arbres de calcul, la structure algébrique sous-jacente est celle de $\mathbb{Z}/2\,\mathbb{Z}$. Concernant le coût nous parlerons du nombre d'*opérations sur les bits*. Deux entiers de taille n peuvent être multipliés avec $O(n^2)$ opérations sur les bits par la méthode naïve. Tout comme pour les polynômes nous serons amenés à analyser le coût de certains algorithmes en fonction du coût des opérations élémentaires sur les entiers. Nous introduisons par conséquent la fonction $\mathsf{I}(n)$ pour borner le coût de l'algorithme choisi pour multiplier deux entiers de taille n. Les meilleurs algorithmes connus à ce jour permettent de prendre $\mathsf{I}(n) \in O(n \log n\, 2^{\log^* n})$ [80, 81], où $\log^*$ représente la fonction *logarithme itéré* – précisément $\log^* n$ est zéro si $n \leqslant 1$, et $1 + \log^*(\log n)$ si $n > 1$. Pour des raisons techniques, nous supposerons que I satisfait les propriétés suivantes :

- $\mathsf{I}(n)/n$ est croissante,

– $\mathsf{I}(m\,n) \leqslant m^2\,\mathsf{I}(n)$ pour tout $m \geqslant 1$.

Enfin, nous supposons dans ce mémoire qu'une matrice M de taille $l \times c$ est représentée par le vecteur de ses $l\,c$ coefficients. Nous parlerons ainsi d'une *représentation dense* des matrices. La quantité ω représentera un nombre réel dans l'intervalle $]2,\,3]$ tel que le coût de l'algorithme choisi pour multiplier deux matrices de tailles $n \times n$ sur un anneau commutatif soit au plus $O(n^\omega)$ opérations d'anneau (c'est à dire $+$, $-$, $\times$, $=$). Il est désormais classique que le coût de nombreuses opérations sur les matrices s'expriment simplement en fonction du coût du produit [3, 26, 38, 277].

I.2 Évaluation et interpolation à une variable

Un principe général de transposition de circuits assure que tout circuit linéaire se transforme automatiquement en un circuit linéaire calculant l'application transposée dans le même nombre d'opérations arithmétiques. Cette transformation de circuits trouve ses origines dans le principe de conservation d'énergie des circuits électriques de Tellegen [6, 30, 237, 279]. Ce principe de transposition était connu depuis les années soixante dix comme un résultat de complexité théorique en calcul formel dont l'impact pratique était incertain [18, 41, 169, 260, 262, 308], sauf pour deux exceptions notables [123, 261]. Pour plus de détails historiques nous renvoyons le lecteur à [21].

Le problème de la complexité en mémoire fut explicitement posé par E. Kaltofen [169, Open Problem 6]. Sans entrer dans les détails sur les modèles de calcul propres à ce problème, en collaboration avec A. Bostan et É. Schost, nous avons répondu à cette question en montrant comment réécrire automatiquement un programme linéaire en un autre programme linéaire de coûts essentiellement les mêmes en temps et en espace et calculant l'application transposée [35]. Dans ce même article, nous avons illustré cette technique sur les algorithmes usuels : multiplication, division, interpolation et évaluation en plusieurs points de polynômes à une variable. Faisant chemin inverse cette étude nous a permis de découvrir des algorithmes plus rapides pour la transposée de l'évaluation en plusieurs points.

Théorème I.9. [35] *Un polynôme à une variable de degré d peut être interpolé à partir de $d+1$ couples de points en faisant $5/2\,\mathsf{M}(d)\log d + O(\mathsf{M}(d))$ opérations dans le corps de base. Un polynôme à une variable de degré d peut être évalué en $d+1$ points en faisant $3/2\,\mathsf{M}(d)\log d + O(\mathsf{M}(d))$ opérations dans l'anneau de base.*

Globalement ce résultat a amélioré d'un facteur deux les meilleures bornes précédemment connues, ce qui est loin d'être anodin puisqu'il s'agit de routines qui sont utilisées dans un très grand nombre d'algorithmes, comme nous le verrons dans la suite.

Nous avons généralisé ces techniques pour traiter le problème du calcul des restes dans la division euclidienne par plusieurs polynômes, c'est-à-dire $f \bmod p_1, \ldots, f \bmod p_r$, où f est de degré d et les p_i sont unitaires de degré d_i avec $d_1 + \cdots + d_r \leqslant d$. Ici $f \bmod p_i$ représente le reste de la division de f par p_i.

Théorème I.10. [34] *Les restes $f \bmod p_1$, ..., $f \bmod p_r$ peuvent être calculés en $3\,\mathsf{M}(n)\log(n) + O(\mathsf{M}(n))$ opérations dans l'anneau de base.*

Ce résultat a amélioré d'un facteur presque 4 la précédente borne de complexité connue. Pour factoriser $F \in \mathbb{K}[[x]][y]$ à précision x^{2^l} à partir de la factorisation de $F(0, y)$ et sous réserve que $F(0,\,y)$ soit séparable, nous utilisons une généralisation de l'opérateur de Newton et déduisons du résultat précédent :

Théorème I.11. [34] *Le coût de la remontée des facteurs de F en précision 2^l est dans* $4\,\mathsf{M}(2^l)\,\mathsf{M}(n)\log r + O(\mathsf{M}(n)\,(\mathsf{M}(2^l)+\log n))$.

Ceci a amélioré d'un facteur 3 la meilleure borne précédemment connue. Tous ces algorithmes ont été programmés en C++ et leur efficacité a été parfaitement observée. Notons enfin que ces idées ont été prolongées dans la thèse d'A. Bostan [32], et que le principe de transposition des programmes linéaires tels que nous l'avions décrit dans [34] a été mis en œuvre de façon automatique dans le logiciel Python par L. De Feo (`http://transalpyne.gforge.inria.fr`).

I.3 Produit des polynômes à plusieurs variables

Une *représentation dense* d'un polynôme à plusieurs variables est la donnée de ses degrés partiels et des coefficients de tous ses termes de degrés partiels plus petits. En revanche, une *représentation creuse* est la donnée de ses seuls termes non nuls. L'algorithmique des polynômes et séries en représentation dense à une ou plusieurs variables est désormais arrivée à un niveau d'efficacité élevé, tel qu'il est possible d'observer le coût quasi-linéaire de leur produit à partir de tailles de l'ordre de 128 mots machines. En revanche la représentation creuse des polynômes, qui est très utile dès que le nombre de variables devient grand, est nettement moins bien comprise. Bien que cette algorithmique soit étudiée depuis les années soixante-dix, et soit constamment revisitée au regard des nouvelles architectures matérielles, seuls les algorithmes naïfs, séquentiels ou parallèles, sont véritablement disponibles dans les logiciels actuels [4, 55, 56, 73, 86, 157, 214, 215, 216, 238, 278, 302]. Dans cette section nous présentons nos résultats théoriques et pratiques permettant de disposer de coûts asymptotiquement plus rapides. Depuis, d'autres techniques sont venues compléter les nôtres [142, 148].

I.3.1 Méthodes par blocs

Les premiers résultats que nous présentons ici ont été obtenus en collaboration avec J. van der Hoeven, et peuvent être vus comme une extension à plusieurs variables de l'algorithme bien connu de Karatsuba pour multiplier deux polynômes à une variable de degré au plus d en temps $O(d^{\log_2 3}) \subseteq O(d^{1,5850})$ [51, 124, 281]. Nos méthodes par blocs trouvent leurs origines dans [134, 139, 174, 245]. L'ensemble des monômes en n variables de degré au plus $d-1$ est noté $S_{n,d}$. Rappelons que le cardinal de cet ensemble est $|S_{n,d}| = \binom{n+d-1}{n}$.

Théorème I.12. [143] *Soit $\mathbb{A}$ un anneau unitaire. Deux polynômes dans $\mathbb{A}[z_1,\ldots,z_n]$ de degré total au plus $d-1$ peuvent être multipliés en utilisant au plus $O(|S_{n,2d-1}|^{1.5930})$ opérations dans $\mathbb{A}$.*

Ce résultat peut être amélioré à condition de disposer de suffisamment de « bons points » pour évaluer et interpoler des polynômes à une variable rapidement. Si $\mathbb{A}$ est un anneau on dira qu'un *schéma d'évaluation* de taille s est un morphisme de $\mathbb{A}[x]$ vers $\mathbb{A}^s$ admettant une application inverse à droite, appelée *interpolation*. Dans de nombreux cas, il s'agira simplement de l'évaluation en s valeurs de $\mathbb{A}$.

Théorème I.13. [143] *Supposons que $\mathbb{A}$ admette un schéma d'évaluation de taille $2\,d-1$ et de coût $O(d\log^\nu(d+1))$, pour tout $d\geqslant 1$, et pour une certaine constante $\nu\geqslant 0$. Alors deux polynômes dans $\mathbb{A}[z_1,\ldots,z_n]$ de degré total au plus $d-1$ peuvent être multipliés avec au plus $O(|S_{n,2d-1}|^{1.5337})$ opérations dans $\mathbb{A}$.*

I.3.2 Méthodes par évaluation et interpolation

Contrairement à la représentation dense, les domaines d'efficacité respectifs des différents algorithmes deviennent assez complexes dans le cas creux. Par exemple la méthode naïve de multiplication se comporte en temps quasi-linéaire dès que la taille du support du produit devient du même ordre de grandeur que le produit des tailles des supports des polynômes à multiplier. Par ailleurs les algorithmes tels que la transformée de Fourier rapide ne semblent pas se généraliser au cas creux. En fait, le produit de deux polynômes en représentation creuse se décompose en deux sous-problèmes distincts :

1. détermination du support (ou d'un sur-ensemble) du produit,
2. calcul des coefficients du produit.

En collaboration avec J. van der Hoeven nous avons tout d'abord porté nos efforts sur le calcul des coefficients du produit en supposant son support connu. Cela couvre de nombreux cas tels que le produit de séries à plusieurs variables tronquées en degré total. Par ailleurs il existe en général des algorithmes probabilistes pour deviner le support d'un produit [85, 171, 172, 173]. Le point de départ de notre travail était un article de J. Canny, E. Kaltofen et Y. Lakshman permettant de calculer les coefficients du produit en caractéristique nulle en temps quasi-linéaire [41]. Leur méthode ne semblait pas vraiment utilisable en pratique car elle faisait intervenir des nombres rationnels de très grande taille et elle ne se généralisait pas en caractéristique positive. En combinant des idées de différents auteurs ayant travaillé sur le problème de l'interpolation creuse [119, 120, 296], nous avons conçu des algorithmes se comportant mieux en pratique.

Notons P et Q les polynômes à multiplier. Soit $P \in \mathbb{A}[z_1, ..., z_n]$ donné en représentation creuse par la suite de ses exposants et coefficients $(e, P_e) \in \mathbb{N}^n \times \mathbb{A}$. La *taille en bits* d'un exposant $e \in \mathbb{N}^n$ est notée l_e et est définie comme $n + \sum_{j=1}^{n} l_{e_j}$, où l_{e_j} represente la taille en bits de e_j. Le degré partiel de P en z_i est noté $\deg_{z_i} P$. Nous définissons $s_P := |\operatorname{supp} P|$ le nombre de termes non nuls de P, et $d_P := (\deg_{z_1} P + 1) \cdots (\deg_{z_n} P + 1)$ la taille dense de P. Nous introduisons aussi les quantités $e_P := \sum_{e \in \operatorname{supp} P} l_e$, $e_Q := \sum_{e \in \operatorname{supp} Q} l_e$. Si X est un sous-ensemble fini de $\mathbb{N}^n$, alors nous posons $e_X := \sum_{e \in X} l_e$ et $d_X := (d_{X,1} + 1) \cdots (d_{X,n} + 1)$, où $d_{X,i}$ est le plus grand degré partiel en z_i des monômes de X. Enfin nous aurons besoin des quantités $\sigma := s_P + s_Q + s_X$ et $\epsilon := e_P + e_Q + e_X$ et nous parlerons d'opérations vectorielles dans $\mathbb{A}$ pour faire référence aux additions, soustractions dans $\mathbb{A}$ et produit d'un élément de $\mathbb{A}$ par un élément de $\mathbb{K}$.

Théorème I.14. [146] *Soit $\mathbb{K}$ un corps, soit $\mathbb{A}$ une $\mathbb{K}$-algèbre, soient P et Q deux polynômes dans $\mathbb{A}[z_1, ..., z_n]$, soit $X \subseteq \mathbb{N}^n$ contenant le support de PQ, et supposons donné un élément de $\mathbb{K}$ d'ordre multiplicatif au moins d_X. Alors le produit PQ peut être calculé au moyen de :*

- *$O(\epsilon)$ produits dans $\mathbb{K}$, qui dépendent seulement de $\operatorname{supp} P$, $\operatorname{supp} Q$ et X;*
- *$O(s_X)$ inversions dans $\mathbb{K}$, $O(s_X)$ produits dans $\mathbb{A}$, et $O\left(\frac{\sigma}{s_X} \mathsf{M}(s_X) \log s_X\right)$ opérations vectorielles dans $\mathbb{A}$.*

Théorème I.15. [146] *Soient $P, Q \in \mathbb{F}_{p^k}[z_1, ..., z_n]$, soit $X \subseteq \mathbb{N}^n$ tel que $X \supseteq \operatorname{supp} PQ$, et supposons donné un élément primitif de $\mathbb{F}_{p^k}^*$. Si $p^k - 1 \geqslant d_X$, alors le produit PQ peut être calculé au moyen de :*

- *$O(\epsilon\, \mathsf{M}(k))$ opérations d'anneau dans $\mathbb{F}_p$, qui dépendent seulement de $\operatorname{supp} P$, $\operatorname{supp} Q$ et X;*
- *$O\left(\frac{\sigma}{s_X} \mathsf{M}(s_X\, k) \log s_X + s_X\, \mathsf{M}(k) \log k\right)$ ou $\tilde{O}(\sigma\, k)$ opérations d'anneau dans $\mathbb{F}_p$ et $O(s_X)$ inversions dans $\mathbb{F}_p$.*

Théorème I.16. [146] *Soient $P, Q \in \mathbb{F}_{p^k}[z_1, ..., z_n]$, soit $X \subseteq \mathbb{N}^n$ tel que $X \supseteq \operatorname{supp} PQ$, et supposons donné un élément primitif de $\mathbb{F}_{p^k}^*$. Si $p^k - 1 \geqslant d_X$ et $\log(s_X\, k) \in O(\log p)$, alors le produit PQ peut être calculé au moyen de :*

- *$O(\epsilon\, \mathsf{I}(k \log p))$ opérations sur les bits qui dépendent uniquement de $\operatorname{supp} P$, $\operatorname{supp} Q$ et X;*
- *$O\left(\frac{\sigma}{s_X}\, \mathsf{I}(s_X\, k \log p) \log s_X + s_X\, \mathsf{I}(k \log p) \log k + s_X\, \mathsf{I}(\log p) \log\log p\right)$ ou $\tilde{O}(\sigma\, k \log p)$ opérations sur les bits additionnelles.*

Si P et Q sont à coefficients entiers, notons $h_P := \max_e l_{P_e}$ le maximum des tailles en bits des coefficients de P et posons $h := h_P + h_Q + l_{\min(d_P, d_Q)}$.

Théorème I.17. [146] *Soient $P, Q \in \mathbb{Z}[z_1, ..., z_n]$ et $X \subseteq \mathbb{N}^n$ tels que $X \supseteq \operatorname{supp} PQ$. Si $p > \max(2^{h+1}, d_X)$ et si un élément primitif de $\mathbb{F}_p^*$ est donné, alors le produit PQ peut être calculé au moyen de :*

- *$O(\epsilon\, \mathsf{I}(\log p))$ opérations sur les bits qui dépendent seulement de $\operatorname{supp} P$, $\operatorname{supp} Q$ et X;*
- *$O\left(\frac{\sigma}{s_X}\, \mathsf{I}(s_X \log p) \log s_X + s_X\, \mathsf{I}(\log p) \log\log p\right)$ ou $\tilde{O}(\sigma \log p)$ opérations sur les bits.*

Une suite $p_1 < \cdots < p_r$ de nombres premiers tels que

$$\begin{aligned} p_1 &> d_X, \\ p_1 \cdots p_r &> 2^{h+1}\, n \end{aligned}$$

est dite d'ordre d_X et de capacité 2^{h+1}, si $\log p \in O(h + \log d_X)$, avec $p = p_1 \cdots p_r$.

Théorème I.18. [146] *Soient $P, Q \in \mathbb{Z}[z_1, ..., z_n]$ et $X \subseteq \mathbb{N}^n$ tels que $X \supseteq \operatorname{supp} PQ$, soit $p_1 < \cdots < p_r$ une suite de nombres premiers d'ordre d_X et de capacité 2^{h+1}, et supposons donné un élément de $\mathbb{Z}/p_1 \cdots p_r\,\mathbb{Z}$ d'ordre au moins d_X. Alors il est possible de calculer PQ au moyen de :*

- *$O(\epsilon\, \mathsf{I}(\log p))$ opérations sur les bits qui dépendent seulement de $\operatorname{supp} P$, $\operatorname{supp} Q$, et X;*
- *$O\left(\frac{\sigma}{s_X}\, \mathsf{I}(s_X \log p) \log s_X + s_X\, \mathsf{I}(\log p) \log\log p\right)$ ou $\tilde{O}(\sigma \log p)$ opérations sur les bits.*

Il était connu que les algorithmes de produit rapides de polynômes à plusieurs variables pouvaient être utilisés pour résoudre des systèmes algébriques [41]. Nous avons proposé une autre application pour le comptage des facteurs absolument irréductibles d'un polynôme.

Théorème I.19. [146] *Soit $\mathbb{K}$ un corps de caractéristique nulle, et soit $F \in \mathbb{K}[x_1, ..., x_n, y]$ un polynôme primitif et séparable en y. Supposons donnés l'enveloppe convexe entière S de F, un élément de $\mathbb{K}$ d'ordre au moins $(2 \deg_y F + 1) \prod_{i=1}^n (2 \deg_{x_i} F + 1)$, et un sous-ensemble $\mathcal{S}$ de $\mathbb{K}$ contenant au moins $5\,|2\,S| - 2$ éléments. Alors, le nombre de facteurs absolument irréductibles de F peut être calculé avec un algorithme probabiliste en utilisant $\tilde{O}(|2\,S|^2)$ opérations dans $\mathbb{K}$. La probabilité que le calcul soit correct est au moins $1 - \frac{3}{2}\,|2\,S|\,(|2\,S| + 1)/|\mathcal{S}|$.*

Ce résultat s'est vu ensuite supplanté par celui publié dans [24], que nous présenterons en section II.6.

I.4 Produit des séries à plusieurs variables

Les méthodes par blocs pour les polynômes peuvent s'utiliser naturellement sur les séries et nous avons obtenu les deux résultats suivants en collaboration avec J. van der Hoeven.

Théorème I.20. [143] *Soit $\mathbb{A}$ un anneau unitaire. Deux séries dans $\mathbb{A}[[z_1, ..., z_n]]$ tronquées en précision d peuvent être multipliées en utilisant au plus $O(|S_{n,d}|^{1.5930})$ opérations dans $\mathbb{A}$.*

Théorème I.21. [143] *Supposons que $\mathbb{A}$ admette un schéma d'évaluation de taille $2\,d-1$ et de coût $O(d\log^{\nu}(d+1))$, pour tout $d\geqslant 1$, et pour une certaine constante $\nu\geqslant 0$. Alors deux séries dans $\mathbb{A}[[z_1, ..., z_n]]$ tronquées en précision d peuvent être multipliées avec au plus $O(|S_{n,d}|^{1.5337})$ opérations dans $\mathbb{A}$.*

Concernant les méthodes par évaluation et interpolation, avec É. Schost, nous avons établi le résultat suivant :

Théorème I.22. [195] *Le produit de deux séries à plusieurs variables tronquées en degré total et dont les coefficients sont dans un corps de caractéristique nulle peut se calculer un temps quasi-linéaire.*

Tout comme pour les polynômes les techniques sous-jacentes à ce résultat cachent des nombres entiers de grande taille, qui nuisent très largement aux performances pratiques dans les cas usuels. Dans les résultats suivants nous utilisons la notation $X := \{i\in\mathbb{N}^{n-1}: i_1+\cdots+i_{n-1}\leqslant d-1\}$ En collaboration avec J. van der Hoeven, nous avons proposé les alternatives pratiques suivantes :

Théorème I.23. [146] *Soit $\mathbb{K}$ un corps, $\mathbb{A}$ une $\mathbb{K}$-algèbre, soient P, $Q\in\mathbb{A}[[z_1, ..., z_n]]/(z_1, ..., z_n)^d$, et supposons donné un élément de $\mathbb{K}$ d'ordre au moins d^{n-1}. Alors, le produit PQ peut être calculé au moyen de $O(s_X\,d)$ inversions dans $\mathbb{K}$, $O(s_X\,\mathsf{M}(d))$ opérations d'anneau dans $\mathbb{A}$, et $O(d\,\mathsf{M}(s_X)\log s_X)$ opérations vectorielles dans $\mathbb{A}$.*

Théorème I.24. [146] *Soient $n\geqslant 2$, $p^k-1\geqslant d^{n-1}$, $\log(s_X\,k)\in O(\log p)$, P, $Q\in\mathbb{F}_{p^k}[[z_1, ..., z_n]]/(z_1, ..., z_n)^d$, et supposons donné un élément primitif de $\mathbb{F}_{p^k}^*$. Alors, le produit PQ peut être calculé au moyen de*

$$O\left(d\,\mathsf{I}(s_X\,k\log p)\log s_X + s_X\,d\,(\mathsf{I}(k\log p)\log k + \mathsf{I}(\log p)\log\log p)\right)$$

ou $\tilde{O}(|S_{n,d}|k\log p)$ opérations sur les bits.

Dans le cas de coefficients entiers nous obtenons :

Théorème I.25. [146] *Soit $n\geqslant 2$, $2^{h+1}\geqslant d^{n-1}$, supposons donnée une suite $p_1<\cdots<p_r$ d'ordre au moins d^{n-1} et de capacité 2^{h+1}, et supposons aussi donné un élément de $\mathbb{Z}/p_1\cdots p_r\,\mathbb{Z}$ d'ordre au moins d^{n-1}. Soient P, $Q\in\mathbb{Z}[[z_1, ..., z_n]]/(z_1, ..., z_n)^d$. Le produit PQ peut alors être calculé avec*

$$O\left(d\,\mathsf{I}(s_X\,h)\log s_X + s_X\,d\,\mathsf{I}(h)\log h\right)$$

ou $\tilde{O}(|S_{n,d}|\,h)$ opérations sur les bits.

I.5 Algorithmes détendus

Dans le chapitre suivant nous aborderons des algorithmes de factorisation de polynômes qui reposent sur la méthode de Newton-Hensel pour calculer les approximations successives de facteurs dans $\mathbb{K}[[x]][y]$ ou $\mathbb{Z}_p[y]$ dans les cas réguliers. En collaboration avec J. Berthomieu et J. van der Hoeven, nous avons conçu une algorithmique pour calculer efficacement avec des nombres p-adiques d'une façon détendue [22].

Soit $\mathbb{A}$ un anneau commutatif et (p) un idéal principal propre de $\mathbb{A}$. Nous notons $\mathbb{A}_p$ le complété de $\mathbb{A}$ pour la valuation p-adique. D'un point de vue pratique, il nous faut supposer que nous disposons d'un sous-ensemble M d'éléments de $\mathbb{A}$ tel que la restriction à M de la projection $\mathbb{A}\to\mathbb{A}/(p)$ soit une bijection. Les éléments de $\mathbb{A}_p$ admettent alors une unique représentation de la forme $\sum_{i\geqslant 0}a_i\,p^i$ avec $a_i\in M$. Lorsque $\mathbb{A}=\mathbb{K}[x]$ et $p=x$, nous pouvons prendre $M=\mathbb{K}$. Lorsque $\mathbb{A}=\mathbb{Z}$, nous pouvons prendre $M=\{0,...,p-1\}$. Nous devons aussi disposer de fonctions calculant les restes et quotients par p : $\mathrm{quo}(\cdot,p)$: $\mathbb{A}\to\mathbb{A}$ et $\mathrm{rem}(\cdot,p)$: $\mathbb{A}\to M$ tels que $a=\mathrm{quo}(a,p)\,p+\mathrm{rem}(a,p)$, pour tout $a\in\mathbb{A}$.

Les calculs dans $\mathbb{A}_p$ sont souvent effectués à précision fixe p^n et se réduisent par conséquent directement à des opérations dans $\mathbb{A}$, on parle alors d'*algorithmique zélée*. L'avantage de cette approche est de pouvoir être implantée rapidement *via* des méthodes de type Newton. Une autre approche, dite *paresseuse*, est plus naturelle d'un point de vue mathématique et offre un meilleur confort d'utilisation : une série est représentée par le vecteur de ses coefficients déjà calculés et une fonction pour calculer le premier coefficient manquant. L'algorithmique appelée *détendue*, introduite en calcul formel dans les articles [133, 134, 137] pour les séries, concerne des techniques rapides de calcul dans l'approche paresseuse, avec la contrainte que le n-ième coefficient d'un nombre est calculé en utilisant uniquement les n premiers coefficients des nombres dont il dépend. En collaboration avec J. Berthomieu et J. van der Hoeven, nous avons étendu ces techniques pour calculer dans $\mathbb{A}_p$, et avons porté nos efforts à les rendre très efficaces pour $\mathbb{Z}_p$. Pour les opérations élémentaires nous avons obtenus les résultats suivants.

Théorème I.26. [22] *Si $\mathsf{I}_p(n)$ borne le coût de l'algorithme choisi pour le produit dans $\mathbb{Z}/p^n\mathbb{Z}$, alors le produit et le quotient de deux nombres p-adiques dans le modèle détendu peut se faire en temps $O(\mathsf{I}_p(n)\log n)$.*

Dans notre article [22] nous présentons des méthodes par blocs permettant de calculer plus vite le quotient de deux nombres entiers p-adiques que par la méthode zélée basée sur la bibliothèque GMP [112].

L'algorithmique détendue est globalement pénalisée par rapport à celle zélée. Néanmoins, son utilisation devient spectaculaire pour le calcul de *nombres récursifs* définis par $a = \Phi(a)$ et tels que a_n peut être calculé en fonction de $a_0, \ldots, a_{n-1}$, dès lors que la formule pour Φ comporte peu d'opérations, comme nous l'avons mis en évidence dans [22, Section 5]. Nous avons construit une famille d'exemple pour laquelle notre approche est dix fois plus rapide que les méthodes naives et les méthodes de Newton. Notre travail s'est achevé avec la construction d'opérateurs de point fixe qui s'évaluent bien pour calculer les racines r-ièmes des séries (en caractéristique quelconque) et nombres p-adiques.

Théorème I.27. *Supposons r inversible modulo p, et soit a un nombre inversible de $\mathbb{A}_p$ tel que a_0 est une puissance r-ième modulo p. Alors chacune des racines r-ièmes b_0 de a_0 modulo p peut être remontée de façon unique en une racine r-ième b de a. De plus b est un nombre récursif pour l'équation*

$$b = \frac{a - b^r + r\,b_0^{r-1}\,b}{r\,b_0^{r-1}}.$$

Les n premiers termes de b peuvent être calculés au moyen de

- *$O(\log r\,\mathsf{M}(n)\log n)$ opérations dans $\mathbb{K}$, si $\mathbb{A} = \mathbb{K}[[x]]$ et $p = x$, ou*
- *$O(\log r\,\mathsf{I}_p(n)\log n)$ operations sur les bits, si $\mathbb{A} = \mathbb{Z}$.*

Nous avons aussi obtenu des coûts similaires lorsque $\mathbb{A} = \mathbb{Z}$ et $r = p$. J. Berthomieu et R. Lebreton ont ensuite généralisé ce calcul des racines r-ièmes à celui des racines de n'importe quel système de polynôme [23]. J. van der Hoeven a aussi étendu l'approche aux problèmes différentiels [138]. R. Lebreton a récemment conçu, sur ces bases, une remontée détendue pour les ensembles triangulaires qui peut aussi être utilisée dans l'algorithme de résolution géométrique du chapitre III [186].

Chapitre II

Factorisation des polynômes

La recherche des racines d'un polynôme à une variable et plus généralement la factorisation de tels polynômes en irréductibles trouve ses origines dans des temps très anciens des mathématiques. Contrairement aux opérations élémentaires sur les polynômes telles que la somme, le produit, et le plus grand commun diviseur (noté *p.g.c.d.*), il n'existe pas d'algorithme de factorisation des polynômes de $\mathbb{K}[x]$ qui soit indépendant de $\mathbb{K}$, dans un modèle de calcul comme les *machines de Turing* ou bien, de façon équivalente, les *machines à accès direct* [78, 79]. Il est par conséquent nécessaire d'examiner la nature de $\mathbb{K}$ afin de concevoir des algorithmes de factorisation dans $\mathbb{K}[x]$.

En pratique, il n'est pas franchement restrictif de considérer des corps $\mathbb{K}$ *explicitement finiment engendrés* sur leur sous corps premier $\mathbb{F}$, c'est-à-dire donné comme le corps des fractions de $\mathbb{F}[x_1, ..., x_n]/\mathfrak{P}$, où $\mathfrak{P}$ est un idéal premier de $\mathbb{F}[x_1, ..., x_n]$ lui-même décrit par un ensemble fini de polynômes de $\mathbb{F}[x_1, ..., x_n]$. Avec de tels corps $\mathbb{K}$, de nombreux travaux ont contribué à montrer que la factorisation des polynômes est calculable dans $\mathbb{K}[x]$. Les contributions majeures, d'un point de vue historique, remontent à L. Kronecker [184], puis, quarante ans plus tard, à G. Hermann [129], suivi par B. L. van der Waerden [287, 288], A. Fröhlich et J. C. Shepherdson [78, 79] dans les années cinquante, et enfin A. Seidenberg [256, 257, 258] et F. Richman [212, 244]. Un bref historique se trouve par exemple dans [90]. Une présentation historique plus détaillée sur la factorisation des polynômes se trouve dans les sections « Notes » des chapitres de la partie III du livre [91].

L'approche développée dans ce chapitre s'articule globalement autour d'une récurrence sur la façon dont $\mathbb{K}$ est construit comme suite d'extensions monogènes. Nous supposons disponible la factorisation dans $\mathbb{Q}[x]$ et $\mathbb{F}_{p^k}[x]$. Ensuite, si nous disposons de la factorisation dans $\mathbb{K}[x]$ nous montrons comment en déduire celle dans $\mathbb{K}[x][y]$, puis dans $\mathbb{K}[x_1, ..., x_n]$. Les ingrédients sont essentiellement la factorisation séparable et la remontée de Hensel. Notons ici que la factorisation séparable est au cœur des traitements modernes de la factorisation en mathématiques constructives, telles que développées par F. Richman est ses collaborateurs [213].

La plupart des logiciels de calcul formel offrent des fonctions pour factoriser les entiers et les polynômes. Le cadre général des polynômes à plusieurs variables sur un corps $\mathbb{K}$ explicitement finiment engendré sur son sous-corps premier est disponible en toute généralité dans le logiciel Magma [274]. Parmi les logiciels libres, de nombreux cas particuliers de factorisation sont disponibles dans NTL [263], Pari/gp [235], Sage [276], Singular [64], et plus récemment dans Mathemagix [147].

II.1 Factorisation séparable

Dans cette section $\mathbb{A}$ représente un anneau factoriel, $\mathbb{L}$ son corps des fractions, p sa caractéristique et F un polynôme de $\mathbb{A}[x]$ de degré $d \geqslant 1$. Rappelons que $F \in \mathbb{A}[x]$ est dit *séparable* s'il n'a aucune racine multiple dans une clôture algébrique $\bar{\mathbb{L}}$ de $\mathbb{L}$. Un polynôme F qui n'est pas constant est séparable si, et seulement si, son discriminant est non nul. Dans la suite nous notons $\mathrm{res}(F, G)$ le *résultant* des polynômes F et G. Notons $\mathcal{B} := \{1, p, p^2, p^3, ...\}$ l'ensemble des puissances de p si $p > 0$, et $\mathcal{B} := \{1\}$ si $p = 0$.

La *décomposition séparable* d'un polynôme primitif $F \in \mathbb{A}[x] \setminus \mathbb{A}$, notée $\mathrm{sep}(F)$, est l'ensemble des triplets $(G_1, q_1, m_1), ..., (G_s, q_s, m_s)$ de $\mathbb{A}[x] \times \mathcal{B} \times \mathbb{N}^*$ vérifiant les propriétés suivantes :

(S_1) $F(x) = \prod_{i=1}^{s} G_i^{m_i}(x^{q_i})$,

(S_2) les $G_i(x^{q_i})$ sont premiers entre eux deux à deux,
(S_3) les m_i ne sont pas divisibles par p,
(S_4) les G_i sont primitifs et separables de degré au moins 1,
(S_5) les couples (q_i, m_i) sont deux à deux distincts.

Notons que dans le cas où $p = 0$ la factorisation séparable n'est rien d'autre que la factorisation sans carré. Il est classique que tout polynôme primitif F de $\mathbb{A}[x]$ admet une unique décomposition séparable (à permutation et unités près dans $\mathbb{A}$). Concrètement, il s'agit juste de regrouper les racines en fonction de leur multiplicité et de la valuation p-adique de celle-ci.

L'algorithme de factorisation classique, proposé par P. Gianni et B. Trager [98], est dérivé de celui attribué à R. D. Musser dans le cas de la caractéristique nulle [226], et son coût est en $O(d\,\mathsf{M}(d)\log d)$ opérations dans le corps des coefficients. Nous avons proposé un algorithme plus rapide que l'on peut voir comme une extension de l'algorithme de D. Y. Y. Yun [91, Theorem 14.23]. Par ailleurs, lorsque $\mathbb{A}$ est parfait, il coïncide essentiellement avec l'algorithme de décomposition sans carré proposé dans [177, Section 4.6.3, Exercise 36] ou [91, Exercise 14.30].

Théorème II.1. [192] *La décomposition séparable de $F \in \mathbb{K}[x]$ de degré d peut être calculée avec $O(\mathsf{M}(d)\log d)$ opérations dans $\mathbb{K}$.*

Dans le cas où $\mathbb{A} = \mathbb{K}[t]$, avec $\mathbb{K}$ un corps, nous énonçons ici les résultats de complexité qui nous serons utiles par la suite. L'utilisation directe de l'algorithme rapide est assez délicate si l'on souhaite éviter que des polynômes de grands degrés en t interviennent dans les calculs intermédiaires. Une manière efficace pour obtenir la décomposition séparable consiste à calculer les décompositions séparables de $F(a,x)$ pour suffisamment de valeurs de a, à éliminer celles pour lesquelles la spécialisation en $t = a$ ne commute pas avec le calcul de la décomposition séparable de $F(t,x)$, puis à interpoler les facteurs séparables avec les valeurs restantes. Notons d_t (resp. d_x) le degré partiel de F en t (resp. en x).

Théorème II.2. [192] *Si $\mathbb{K}$ contient au moins $d_t\,(2\,d_x+1)+1$ éléments, alors $\mathrm{sep}(F)$ peut être calculé avec au plus $O(d_x\ (d_x\ \mathsf{M}(d_t)\ \log\ d_t\ +\ d_t\ \mathsf{M}(d_x)\ \log\ d_x))$ ou $\tilde{O}(d_t\ d_x^2)$ opérations dans $\mathbb{K}$.*

Théorème II.3. [192] *Si $\mathbb{K}$ contient au moins $4\,d_t\,d_x$ éléments, alors $\mathrm{sep}(F)$ peut être calculé avec au plus $O(d_x\,\mathsf{M}(d_t)\log d_t + d_t\,\mathsf{M}(d_x)\log d_x)$ ou $\tilde{O}(d_t\,d_x)$ opérations dans $\mathbb{K}$ en moyenne.*

Finalement dans [192], nous montrons que la factorisation irréductible d'un polynôme F de $\mathbb{K}[x]$ de degré d se réduit à la factorisation séparable de F, puis à des factorisations irreducibles de polynômes séparables de $\mathbb{K}[x]$ dont la somme des degrés est au plus d, et enfin à $O(d)$ extractions de racines p-ièmes dans $\mathbb{K}$. Cette réduction au cas séparable s'adapte aussi à la factorisation sans carré.

Il est souvent suggéré, dans des articles en calcul formel, que la factorisation irréductible se réduit à factoriser des polynômes sans carré [20, 57, 88]. Bien que ce point de vue convienne bien à la caractéristique zéro, il conduit souvent à des difficultés techniques en caractéristique positive. En fait si $\mathbb{A} = \mathbb{K}[t]$, avec $\mathbb{K}$ un corps, nous verrons dans la section suivante que nous pouvons réduire la factorisation irréductible dans $\mathbb{A}[x]$ à celle dans $\mathbb{K}[x]$ au moyen de la remontée de Hensel. Si la caractéristique p est nulle ou suffisamment grande et que F est sans carré alors cette réduction commence par choisir un point de spécialisation $a \in \mathbb{K}$ tel que $F(a,x)$ soit sans carré. Maintenant si $p > 0$ cette stratégie ne s'applique plus. Par exemple, considérons $F(t,x) := (x^p + t)\,(x + t^p)$ en prenant pour $\mathbb{K}$ une clôture algébrique de $\mathbb{F}_p$. Il est clair que F est sans carré et que $F(a,x)$ n'est pas sans carré quelle que soit la valeur choisie pour $a \in \mathbb{K}$. Par symmétrie il en va de même

si l'on permute t et x. Par ailleurs nous avons $\mathrm{sep}(F) = \{(x + t^p, 1, 1), (x + t, p, 1)\}$, et pour tout $a \in \mathbb{K}$, les polynômes $x + a^p$ et $x + a$ sont séparables.

Dans leur algorithme de factorisation irréductible des polynômes à plusieurs variables sur un corps fini, L. Bernardin et M. B. Monagan [20], résolvent ce problème du choix de la valeur a pour la remontée de Hensel en utilisant la factorisation séparable de manière un peu cachée. L'utilisation explicite en calcul formel de la factorisation séparable comme un prétraitement systématique à la factorisation irréductible est due à A. Steel [274].

II.2 Factorisation à deux variables

Dans cette section nous nous intéressons à la factorisation d'un polynôme $F \in \mathbb{K}[x, y]$, où $\mathbb{K}$ est un corps commutatif quelconque en supposant que nous disposons déjà d'algorithmes pour factoriser dans $\mathbb{K}[y]$.

II.2.1 Notations

Notons d_x le degré de F en la variable x et d_y celui en la variable y. Nous supposons que F est primitif et séparable vu comme polynôme de $\mathbb{K}[x][y]$. Les facteurs irréductibles de F sont notés $F_1, ..., F_r$. Le coefficient de tête $\mathrm{lc}_y(F)$ de F vu comme polynôme de $\mathbb{K}[x][y]$ est noté c. Les coefficients de têtes $\mathrm{lc}_y(F_i)$ sont notés c_i. Afin de simplifier la présentation nous supposons aussi que les coefficients de têtes de c et des c_i sont tous 1.

Supposons maintenant que l'on puisse trouver un point $a \in \mathbb{K}$ tel que $F(a, y)$ reste de degré d_y et séparable. Quitte à appliquer une translation à la variable x, nous pouvons alors calculer les facteurs irréductibles de F dans $\mathbb{K}[[x]][y]$, que nous notons $\mathfrak{F}_1$, ..., $\mathfrak{F}_s$ et appelons *facteurs analytiques* par opposition aux *facteurs rationnels* F_i. Pour chaque $i \in \{1, ..., r\}$, nous associons l'unique vecteur $\mu_i \in \{0, 1\}^s$ défini par $F_i = c_i \prod_{j=1}^{s} \mathfrak{F}_j^{\mu_{i,j}}$. Nous examinerons plus tard la situation où $\mathbb{K}$ est trop petit pour garantir de trouver un tel point a, mais en attendant nous supposerons les hypothèses suivantes satisfaites :

$$\text{(N)} \quad \begin{cases} \deg\left(F(0, y)\right) = d_y, \\ \mathrm{Res}\left(F(0, y), \frac{\partial F}{\partial y}(0, y)\right) \neq 0, \\ F \text{ est primitif vu dans } \mathbb{K}[x][y]. \end{cases}$$

Si $A = \sum_{i,j \geqslant 0} a_{i,j}\, x^i\, y^j$ représente un polynôme de $\mathbb{K}[[x]][y]$ alors nous notons $[A]_k^l$ la projection sur $\mathbb{K}[x, y]$ suivante :

$$[A]_k^l := \sum_{k \leqslant i \leqslant l-1, j \geqslant 0} a_{i,j}\, x^i\, y^j.$$

Nous introduisons aussi les produits partiels suivants :

$$\hat{F}_i := \prod_{j=1, j \neq i}^{r} F_j = \frac{F}{F_i}, \qquad \hat{\mathfrak{F}}_i := c \prod_{j=1, j \neq i}^{s} \mathfrak{F}_j = \frac{F}{\mathfrak{F}_i}.$$

Le calcul des μ_i à partir des $\mathfrak{F}_i$ est appelé le *problème de recombinaison* des facteurs analytiques. Les $\mathfrak{F}_i$ peuvent être calculés à partir de la factorisation de $F(0, y)$ et de la remontée de Newton-Hensel déjà mentionnée dans le chapitre précédent.

II.2.2 Remarques

Le premier algorithme (en temps exponentiel) de factorisation dans $\mathbb{K}[x, y]$ semble remonter aux travaux de L. Kronecker. L'idée était de factoriser $F\left(y^{d_y+1}, y\right)$ puis de chercher parmi toutes les recombinaisons possibles celle donnant $F_i\left(y^{d_y+1}, y\right)$. Rappelons que l'utilisation de la remontée de Newton-Hensel en calcul formel pour factoriser les polynômes est apparue à la fin des années soixante dans les travaux de H. Zassenhaus [304]. L'application à plusieurs variables a commencé dans [227, 290, 291]. Pendant longtemps l'étape de recombinaison était effectuée par simple recherche exhaus-

tive, qui a un coût exponentiel dans le pire des cas. Néanmoins, si $\mathbb{K}$ est un corps fini, en moyenne sur l'ensemble des polynômes, son coût est essentiellement quadratique [84].

Les premiers algorithmes en temps polynomiaux sont dus à E. Kaltofen au début des années quatre-vingt. Plusieurs autres auteurs ont contribué à ce problème pour couvrir les corps de coefficient usuels : A. L. Chistov, J. von zur Gathen, D. Grigoriev et A. K. Lenstra. Tout comme dans le cas de l'algorithme historique pour le factorisation dans $\mathbb{Q}[x]$ [197], ces algorithmes calculent un facteur rationnel à partir d'un seul facteur analytique, sans utiliser la totalité de la décomposition analytique. Les exposants des bornes de complexité sont plutôt élevés et l'efficacité pratique n'est pas spectaculaire par rapport à la recherche exhaustive en moyenne.

La première réduction de la factorisation de deux à une variable en temps quasi-quadratique, pour la caractéristique nulle ou suffisamment grande, est due à Shuhong Gao [83], en utilisant la *factorisation absolue*. Dans le présent chapitre nous nous concentrons sur les méthodes utilisant la remontée de Newton-Hensel, qui conduisent aux meilleures bornes de complexité connues à l'heure actuelle. Ces méthodes ont été étudiées par D. R. Musser [227], P. S. Wang et L. P. Rothschild [290, 291], puis J. von zur Gathen [92, 93], L. Bernardin et M. B. Monagan [20].

Pour des détails historiques concernant la factorisation des polynômes à plusieurs variables nous renvoyons le lecteur aux livres [91, 224, 309], mais aussi vers les articles [48, 82, 88, 160, 166, 167, 168, 170].

II.2.3 Recombinaison

Dans les articles [248, 249, 250], T. Sasaki et ses collaborateurs avaient étudié une méthode pour ramener le problème de recombinaison des facteurs analytiques à de l'algèbre linéaire, mais malheureusement sans prouver de borne de coût polynomiale (*cf.* l'exemple dans l'introduction de l'article [34]). Une variante de leur algorithme a néanmoins conduit à une borne polynomiale grâce aux travaux de K. Belabas, M. van Hoeij, J. Klüners et A. Steel [17]. L'idée est de construire un système linéaire dont les solutions fournissent les vecteurs μ_i, à partir d'une factorisation analytique à une précision $\sigma \geqslant d\,(d-1)+1$ où d est le degré total de F. Cette borne inférieure de précision se trouve être essentiellement nécessaire lorsque $\mathbb{K}$ est de petite caractéristique positive. Dans les paragraphes suivants nous présentons cette méthode, mais en exprimant la borne en fonction des degrés partiels d_x et d_y de F.

À partir de la dérivée logarithmique de la formule $F_i = c_i \prod_{j=1}^{s} \mathfrak{F}_j^{\mu_{i,j}}$ nous avons :

$$\frac{\frac{\partial F_i}{\partial y}}{F_i} = \sum_{j=1}^{s} \mu_{i,j} \frac{\frac{\partial \mathfrak{F}_j}{\partial y}}{\mathfrak{F}_j}, \text{ et donc } \hat{F}_i \frac{\partial F_i}{\partial y} = \sum_{j=1}^{s} \mu_{i,j}\, \hat{\mathfrak{F}}_j \frac{\partial \mathfrak{F}_j}{\partial y}. \tag{II.1}$$

Comme le degré partiel en x de $\hat{F}_i \frac{\partial F_i}{\partial y}$ est au plus d_x, la dernière inégalité conduit au système linéaire suivant, en les inconnues $\ell_1, \ldots, \ell_s$, et dont les vecteurs μ_i sont solutions :

$$\left[\sum_{j=1}^{s} \ell_j \hat{\mathfrak{F}}_j \frac{\partial \mathfrak{F}_j}{\partial y} \right]_{d_x+1}^{\sigma} = 0. \tag{II.2}$$

Proposition II.4. *Si $\sigma \geqslant d_x\,(2\,d_y - 1) + 1$, alors $\mu_1, \ldots, \mu_r$ forment la base échelonnée réduite (à une permutation près) de l'espace des solutions de (II.2).*

Dans l'article [34], en notant d le degré total de F, il a été prouvé que la borne en précision $d_x\,(2\,d_y - 1) + 1$ peut être ramenée à $3\,d - 2$ si la caractéristique de $\mathbb{K}$ est nulle ou au moins $d\,(d-1)+1$. Dans l'article [190], un autre système linéaire nécessitant une précision $2\,d$ a été proposé sous les mêmes hypothèses. Nous ne présentons pas ces variantes ici, mais une approche sensiblement différente issue de l'article [193] utilisant une précision $d_x + 1$, sans hypothèse sur la caractéristique.

Théorème II.5. [193] *Si* $\mathbb{K}$ *contient au moins* $2\,d_x\,d_y+d_x+1$ *éléments, la décomposition irréductible de* F *peut être obtenue au moyen d'un algorithme qui nécessite de factoriser des polynômes de* $\mathbb{K}[y]$ *dont la somme des degrés n'excède par* d_x+d_y*, plus*

a) si $p=0$ *ou* $p\geqslant d_x\,(2\,d_y-1)+1$*,* $O(d_x\,d_y^\omega)$ *opérations arithmétiques dans* $\mathbb{K}$ *;*

b) si $\mathbb{K}:=\mathbb{F}_{p^k}$*,* $\tilde{O}(k\,d_x\,d_y^\omega)$ *opérations arithmétiques dans* $\mathbb{F}_p$*.*

Remarquons que, sans perte de généralité, nous pouvons supposer que $d_y\leqslant d_x$, ce qui conduit à un coût $\tilde{O}(d_x\ d_y^\omega)\subseteq\tilde{O}((d_x\ d_y)^{(\omega+1)/2})$. De cette façon nous obtenons un algorithme pour réduire la factorisation de deux à une variables en temps moins que quadratique. Le résultat suivant en est une variante probabiliste.

Théorème II.6. [193] *Si* $\mathbb{K}$ *contient au moins* $10\,d_x\,d_y$ *éléments, alors la décomposition irréductible de* F *peut être obtenue au moyen d'un algorithme qui nécessite de factoriser des polynômes de* $\mathbb{K}[y]$ *dont la somme des degrés n'excède pas* d_x+d_y*, plus*

a) si $p=0$ *ou* $p\geqslant d_x\,(2\,d_y-1)+1$*,* $O((d_x\,d_y)^{1,5})$ *opérations en moyenne dans* $\mathbb{K}$ *;*

b) si $\mathbb{K}:=\mathbb{F}_{p^k}$*,* $\tilde{O}(k\,(d_x\,d_y)^{1,5})$ *opérations en moyenne dans* $\mathbb{F}_p$*.*

II.2.4 Petite cardinalité

Si $\mathbb{K}$ est un corps fini $\mathbb{F}_q$ de cardinal trop petit pour pouvoir appliquer les résultats précédents, alors nous pouvons procéder comme suit. Tout d'abord nous construisons une extension algébrique $\mathbb{F}_{q^l}$ de $\mathbb{F}_q$ de degré l sous la forme $\mathbb{F}_q[z]/(\mu(z))$, avec μ irréductible de degré l. Notons α une racine de μ dans $\mathbb{F}_{q^l}$. En prenant $l\in O(\log_q(d_x\,d_y))$ suffisamment grand pour pouvoir factoriser F dans $\mathbb{F}_{q^l}$ avec les algorithmes sous-jacents aux théorèmes précédents, nous pouvons obtenir les facteurs irréductibles $\mathcal{F}_1,...,\mathcal{F}_t$ de F dans $\mathbb{F}_{q^l}$. En prenant les préimages des coefficients des $\mathcal{F}_i$ dans $\mathbb{F}_q[z]$, nous construisons des polynômes $\mathsf{F}_i(x,y,z)$ de degrés au plus $l-1$ en z, tels que $\mathcal{F}_i(x,y)=\mathsf{F}_i(x,y,\alpha)$.

Pour chaque $i\in\{1,...,t\}$, nous pouvons calculer $F_i(x,\,y):=\mathrm{Res}_z(\mathsf{F}_i(x,\,y,\,z),\,\mu(z))$ qui est une puissance d'un facteur irréductible de F. Chaque F_i peut être obtenu par évaluation/interpolation rapide avec $\tilde{O}(\deg_x\,(F_i)\,\deg_y\,(F_i)\,l)$ opérations dans $\mathbb{F}_{q^l}$. Au total tous les F_i sont obtenus en temps quasi-linéaire. Chaque facteur irréductible de F apparaît au plus l fois dans la liste des F_i ainsi calculés. Les redondances sont éliminées simplement en triant la liste des F_i. Au final le surcoût total reste un facteur borné par $\log^{O(1)}(d_x\,d_y)$.

II.3 Factorisation à plusieurs variables

Les algorithmes de factorisation des polynômes à deux variables de la section précédente peuvent être utilisés directement pour traiter le cas à plusieurs variables. En effet, si F est un polynôme de $\mathbb{K}[z_1,\,...,\,z_n,\,y]$, alors il peut être vu comme un polynôme de $\mathbb{K}[z_1,...,z_{n-1}][z_n,y]$, dont nous pouvons factoriser son contenu en y par récurrence sur le nombre de variables, puis factoriser sa partie primitive en y dans $\mathbb{K}(z_1,...,z_{n-1})[z_n,y]$. L'inconvénient de cette approche est double. Premièrement les calculs dans $\mathbb{K}(z_1,\,...,\,z_{n-1})$ sont coûteux car chaque opération élémentaire sur les fractions nécessite un calcul de p.g.c.d. pour s'assurer que les numérateurs et dénominateurs sont premiers entre eux. Deuxièmement, lors de la phase de remontée de Hensel des facteurs dans $\mathbb{K}(z_1,...,z_{n-1})[[z_n]][y]$, la taille des fractions rationnelles croît rapidement. Il est alors préférable de procéder plutôt ainsi :

1. Factoriser le polynôme $F(0,...,0,y)$.
2. Remonter les facteurs de sorte à obtenir la factorisation de F dans $\mathbb{K}[[z_1,...,z_n]][y]$ à la precision $(z_1,...,z_n)^{2d_z+1}$, où d_z est le degré total de F en les seules variables $z_1,...,z_n$.
3. Résoudre le problème de recombinaison pour trouver les facteurs rationnels de F.

En fait, dans cette section, nous présentons une autre technique attribuée à D. Hilbert, assurant que, sous certaines hypothèses, les facteurs irréductibles de F sont en bijection avec ceux de $F(\alpha_1 x, ..., \alpha_n x, y)$ pour presque toutes les valeurs de $\alpha_1, ..., \alpha_n$. La factorisation se ramène alors au cas de deux variables, mais il reste à préciser la probabilité de faire des bons choix pour les α_i.

Nous commençons par examiner le problème de réduction à deux variables sous l'hypothèse suivante, généralisant l'hypothèse (N) précédente :

$$\text{(N)} \quad \begin{cases} \deg\left(F(0, ..., 0, y)\right) = d_y, \\ \operatorname{Res}\left(F(0, ..., 0, y), \frac{\partial F}{\partial y}(0, ..., 0, y)\right) \neq 0, \\ F \text{ est primitif vu dans } \mathbb{K}[z_1, ..., z_n][y]. \end{cases}$$

Définition II.7. *Sous l'hypothèse (N), un point $(\alpha_1, ..., \alpha_n) \in \mathbb{K}^n$ est dit de* Hilbert *si, pour tout facteur irréductible F_i de F, le polynôme $F_i(\alpha_1 x, ..., \alpha_n x, y)$ est irréductible. En d'autres termes, les facteurs irréductibles de F sont en bijection avec ceux de $H(x, y) := F(\alpha_1 x, ..., \alpha_n x, y) \in \mathbb{K}[x, y]$.*

D'un point de vue pratique, la donnée d'un point de Hilbert conduit naturellement à l'algorithme suivant :

1. Factoriser $H(x, y)$ dans $\mathbb{K}[x, y]$.
2. Remonter la factorisation pour retrouver les facteurs $F_1, ..., F_r$ de F.

Le coût dépend alors de celui de l'arithmétique des séries à plusieurs variables. Si cette dernière est en temps quasi-linéaire alors la réduction de n à 1 variable s'effectue en temps quasi-linéaire dès que $n \geqslant 3$.

Le complémentaire de l'ensemble des points de Hilbert de $\mathbb{K}^n$ est noté $\mathcal{H}(F)$. Commençons avec un exemple fournissant une borne inférieure sur la densité de $\mathcal{H}(F)$.

Exemple II.8. [190] Soit $n \geqslant 2$, $\mathbb{K} := \mathbb{C}$, $F := y^d + z_1^{d-1}\, y - z_2^{d-1} - 1$. Un critère d'irréductibilité attribué à S. A. Stepanov et W. M. Schmidt implique que $F(0, z_2, 0, ..., 0, y) = y^d - z_2^{d-1} - 1$ est irréductible, et donc que F lui-même est irréductible [83, p. 503]. Soit S l'ensemble des racines de $z^{d(d-1)} - 1$. Pour n'importe quel point $(\alpha_1, ..., \alpha_n) \in S^n$, le polynôme $y - (\alpha_2/\alpha_1)^{d-1}$ divise $H = y^d - 1 + x^{d-1}\,(\alpha_1^{d-1}\, y - \alpha_2^{d-1})$. Par conséquent, aucun des points de S^n n'est un point de Hilbert pour F, ce qui donne les égalités suivantes :

$$\frac{|\mathcal{H}(F) \cap S^n|}{|S|^n} = \frac{d\,(d-1)}{|S|} = 1.$$

Par le lemme classique de Schwartz-Zippel [255, 306], affirmant qu'un polynôme non nul A en n variables ne peut avoir plus de $\deg(A)|S|^{n-1}$ racines dans S^n, nous en déduisons qu'il n'existe aucun polynôme A de degré au plus $d\,(d-1) - 1$ qui s'annule sur tout les points de $\mathcal{H}(F)$. Le théorème suivant donne une borne supérieure pour le degré d'un tel polynôme A s'annulant sur tout $\mathcal{H}(F)$.

Théorème II.9. [194] *Sous l'hypothèse (N), il existe un polynôme de $\mathbb{K}[a_1, ..., a_n] \setminus \{0\}$ de degré au plus $d_z\,(d_y - 1)\,(2\,d_y - 1)$ qui s'annule sur tout $\mathcal{H}(F)$.*

Théorème II.10. [190, 194] *Sous l'hypothèse (N), si la caractéristique de $\mathbb{K}$ est nulle ou bien au moins $d_z\,(2\,d_y - 1) + 1$, alors il existe un polynôme de $\mathbb{K}[a_1, ..., a_n] \setminus \{0\}$ de degré au plus $(3\,d_z - 1)\,(d_y - 1)$ qui s'annule sur tout $\mathcal{H}(F)$.*

Théorème II.11. [194] *Sous l'hypothèse (N), si la caractéristique p de $\mathbb{K}$ est strictement positive, alors il existe un polynôme de $\mathbb{K}[a_1, ..., a_n] \setminus \{0\}$ de degré au plus $(p\,d_z - 1)\,(d_y - 1)$ qui s'annule sur tout $\mathcal{H}(F)$.*

Corollaire II.12. [190, 194] *Sous l'hypothèse (N), en notant p la caractéristique de $\mathbb{K}$, pour n'importe quel sous-ensemble non vide S de $\mathbb{K}$, nous avons les inégalités suivantes :*

- $\frac{|\mathcal{H}(F)\cap S^n|}{|S|^n} \leqslant \frac{3\,d_z\,d_y}{|S|}$ *si* $p=0$ *ou* $p \geqslant d_z\,(2\,d_y-1)+1$,
- $\frac{|\mathcal{H}(F)\cap S^n|}{|S|^n} \leqslant \frac{d_z\,d_y\min(2\,d_y,p)}{|S|}$ *si* $p>0$.

II.4 Théorème de Bertini-Hilbert

Dans cette section nous abordons une autre façon pour se ramener à l'hypothèse (N). Nous utilisons un changement de variables le plus général possible pour réduire le problème à deux variables. D'un point de vue géométrique cela revient à construire l'équation de l'intersection de l'hypersurface définie par $F=0$ avec un plan générique. Pour éviter d'éventuelles confusions avec les notations, nous considérons un polynôme P de degré $d \geqslant 1$ en les variables $v_1, ..., v_n$ sur $\mathbb{K}$. Pour tout triplet de points $(\alpha_1, ..., \alpha_n)$, $(\beta_1, ..., \beta_n)$ et $(\gamma_1, ..., \gamma_n)$ de $\mathbb{K}^n$, nous définissons le polynôme à deux variables x et y suivant :

$$P_{\alpha,\beta,\gamma} := P(\alpha_1\,x+\beta_1\,y+\gamma_1, ..., \alpha_n\,x+\beta_n\,y+\gamma_n). \tag{II.3}$$

Selon un résultat classique souvent appelé théorème d'irréductibilité de Bertini [259, Chapter II, Section 6.1], si P est irréductible, alors il existe un ouvert de Zariski de $(\mathbb{K}^n)^3$ tel que $P_{\alpha,\beta,\gamma}$ est irréductible pour chaque triplet $(\alpha_1, ..., \alpha_n)$, $(\beta_1, ..., \beta_n)$, $(\gamma_1, ..., \gamma_n)$ dans cet ouvert. Nous dirons qu'un tel triplet est un *point de Bertini* pour P si, pour tout facteur irréductible Q de P, le polynôme $Q(\alpha_1\,x+\beta_1\,y+\gamma_1, ..., \alpha_n\,x+\beta_n\,y+\gamma_n)$ est irréductible de même degré total que Q. En d'autres termes les facteurs irréductibles de P sont en bijection avec ceux de $P_{\alpha,\beta,\gamma}$.

D'un point de vue pratique les coefficients des vecteurs $(\alpha_1, ..., \alpha_n)$, $(\beta_1, ..., \beta_n)$ et $(\gamma_1, ..., \gamma_n)$ doivent être choisis dans un sous-ensemble fini S de $\mathbb{K}$, et il nous faut donner une estimation de la probabilité qu'un triplet pris au hasard dans $(S^n)^3$ soit un point de Bertini pour P.

Théorème II.13. [194] *Soit S un sous-ensemble fini de $\mathbb{K}$. Si $P \in \mathbb{K}[v_1, ..., v_n]$ est séparable en au moins une variable et est de degré total $d \geqslant 1$, alors la proportion $\mathsf{B}(P,S)$ de triplets à coordonnées dans S qui ne sont pas des points de Bertini pour P est au plus*

- $5\,d^2/|S|$ *si la caractéristique p de $\mathbb{K}$ est nulle ou au moins $d\,(2\,d-1)+1$,*
- $d^2\min(2\,d, p+1)/|S|$ *si $p>0$.*

Ce que nous appelons ici théorème de Bertini est plutôt un cas particulier du résultat plus général présenté par exemple dans le livre [259, Chapter II, Section 6.1]. Pour des renseignements historiques et techniques nous renvoyons le lecteur à [159, 176]. En fait, ce cas particulier semble dû à D. Hilbert [131, p. 117], comme souligné dans l'article [168]. Pour cette raison il est fréquent de trouver les terminologies « théorème de Hilbert » ou « théorème de Bertini » dans la littérature sur la factorisation des polynômes.

L'utilisation du théorème de Bertini en calcul formel a été mise en avant pour la première fois dans les articles [128, 161]. Son utilisation s'est rapidement imposée dans de nombreux algorithmes et études de complexité [89, 92, 162, 163, 164, 165]. En caractéristique quelconque, et sous une hypothèse de normalisation similaire à notre hypothèse (N), J. von zur Gathen avait montré que les points qui ne sont pas de Hilbert pour un polynôme donné de degré total d sont inclus dans une hypersurface de degré au plus 9^{d^2} [89]. Lorsque $\mathbb{K}$ est le corps des nombres complexes, l'article [7] obtient la borne $\mathsf{B}(P,\,S) \leqslant (d^4-2\,d^3+d^2+d+1)/|S|$ en suivant la preuve du théorème de Bertini donnée dans [225, Theorem 4.17]. Pour un corps parfait, E. Kaltofen avait obtenu la borne $\mathsf{B}(P,S) \leqslant 2\,d^4/|S|$ dans son article [168]. Lorsque $p=0$ ou $p \geqslant 2\,d^2$, Shuhong Gao avait prouvé la borne $\mathsf{B}(P,S) \leqslant 2\,d^3/|S|$ [82], qui a été ensuite améliorée en $\mathsf{B}(P,\,S) \leqslant d\,(d^2-1)/|S|$ par Chèze [47, Chapitre 1].

II.5 Factorisation absolue

Dans cette section nous supposons que F est un polynôme de $\mathbb{K}[x, y]$ sans carré de degré total d, où $\mathbb{K}$ est un corps de caractéristique p soit nulle soit au moins $d\,(d-1)+1$. En collaboration avec G. Chèze nous avons travaillé sur la *factorisation absolue* de F, c'est-à-dire sa factorisation irréductible dans $\bar{\mathbb{K}}[x, y]$ [49].

Il est connu depuis les travaux d'E. Noether qu'il existe des algorithmes indépendants de $\mathbb{K}$ pour la factorisation absolue [228]. Le calcul d'un facteur possible d'un degré donné se réduit en effet à la résolution d'un système algébrique. Les premiers algorithmes efficaces ont commencé à être étudiés en calcul formel dans les années quatre-vingt, motivés par des questions pratiques d'intégration symbolique [282] et de décomposition absolue des fermés algébriques [121]. Plus récemment les algorithmes de factorisation absolue ont été utilisés pour des problèmes de robotique [271] et ont été mis en lien avec la résolution d'équations différentielles linéaires [37, 132, 266]. J'ai contribué à la programmation d'une petite bibliothèque dans le logiciel Singular [64] pour la factorisation absolue qui a servi à W. Decker pour implanter un algorithme calculant la décomposition primaire d'un ideal en caractéristique zéro dans la clôture algébrique du corps des coefficients.

Dans les paragraphes suivants, nous notons F_1, ..., F_s les facteurs absolument irréductibles de F. Ces facteurs sont représentés en pratique par un ensemble de paires de polynômes $\{(q_1, \mathsf{F}_1), ..., (q_s, \mathsf{F}_s)\}$ satisfaisant les propriétés suivantes :

- Pour chaque $i \in \{1, ..., s\}$, le polynôme q_i appartient à $\mathbb{K}[z]$, est unitaire et séparable de degré $\deg(q_i) \geqslant 1$.
- Pour chaque $i \in \{1, ..., s\}$, le polynôme F_i appartient à $\mathbb{K}[x, y, z]$, et $\deg_z(\mathsf{F}_i) \leq \deg(q_i) - 1$. Le degré total de $\mathsf{F}_i(x, y, \alpha)$ est constant lorsque α parcourt l'ensemble des racines de q_i.
- $\sum_{i=1}^{s} \deg(q_i) = r$ et, à chaque facteur absolument irréductible F_j correspond une unique paire $(i, \alpha) \in \{1, ..., s\} \times \mathbb{K}$ telle que $q_i(\alpha) = 0$ et F_j est proportionnel à $\mathsf{F}_i(x, y, \alpha)$.

Une telle représentation n'est pas unique, mais n'est pas redondante dans le sens où, pour chaque i, les polynômes $\mathsf{F}_i(x, y, \alpha)$ sont deux à deux distincts lorsque α parcourt l'ensemble des racines de q_i.

Théorème II.14. [49] *Soit $\mathbb{K}$ un corps de caractéristique p soit nulle soit au moins $d\,(d-1)+1$. La factorisation absolue d'un polynôme $F \in \mathbb{K}[x, y]$ sans carré de degré total d peut être réalisée au moyen de $O(d^3\,\mathsf{M}(d)\log d)$ opérations arithmétiques dans $\mathbb{K}$.*

Le test d'irréductibilité peut être réalisé un peu plus rapidement que le calcul de tous les facteurs :

Théorème II.15. [49] *Soit $\mathbb{K}$ un corps de caractéristique p soit nulle soit au moins $d\,(d-1)+1$. Tester si un polynôme $F \in \mathbb{K}[x, y]$ sans carré de degré total d est absolument irréductible peut se faire au moyen d'au plus $O(d^{\omega+1} + d^2\,\mathsf{M}(d)\,(\mathsf{M}(d)/d + \log d))$ ou $O(d^{\omega+1})$ opérations arithmétiques dans $\mathbb{K}$.*

Nous avons par ailleurs proposé une variante probabiliste de ces résultats :

Théorème II.16. [49] *Soit $\mathbb{K}$ un corps de caractéristique p soit nulle soit au moins $d\,(d-1)+1$. La factorisation absolue d'un polynôme $F \in \mathbb{K}[x, y]$ sans carré de degré total d peut être réalisée en coût $\tilde{O}(d^3)$ en moyenne.*

Nos méthodes combinent les avantages de la technique classique de remontée de Hensel et de l'algorithme de Shuhong Gao [82] reposant sur le calcul du premier groupe de cohomologie de de Rham du complémentaire de l'hypersurface définie par $F=0$. Pour un état de l'art et de nombreux éléments de comparaisons entre les méthodes connues, nous renvoyons le lecteur à l'introduction de notre article [49].

II.6 Support dense dans le polytope de Newton

Dans le pire des cas, la taille des facteurs irréductibles est exponentielle dans la taille du polynôme en représentation creuse à factoriser.

Exemple II.17. Si p est premier alors $F := x^p - 1 \in \mathbb{Q}[x]$ est composé de seulement deux monômes mais ses facteurs irréductibles sont $x-1$ et $(x^p-1)/x-1$. Ce dernier est composé de p monômes.

Le polytope de Newton de $F \in \mathbb{K}[x_1, ..., x_n]$, noté $N(F)$, est l'enveloppe convexe du support de F. Le nombre de points à coordonnées entières de $N(F)$ est appelé la *taille convexe* de F. Plusieurs algorithmes de factorisation ont un coût qui dépend de la taille convexe du polynôme à factoriser dans le cas à deux variables. Nous allons brièvement présenter l'un d'entre eux, conçu en collaboration avec J. Berthomieu, qui permet de se ramener directement à la situation dense.

Le *groupe affine* de $\mathbb{Z}^2$, noté $\mathrm{Aff}(\mathbb{Z}^2)$, est l'ensemble des applications

$$U\colon (i,j) \mapsto \begin{pmatrix} \alpha & \beta \\ \alpha' & \beta' \end{pmatrix}\begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} \gamma \\ \gamma' \end{pmatrix},$$

avec α, β, γ, α', β', et γ' dans $\mathbb{Z}$, tels que $\alpha\,\beta' - \alpha'\,\beta = \pm 1$. Un sous-ensemble fini S de $\mathbb{Z}^2$ est dit *normalisé* s'il appartient à $\mathbb{N}^2$ et s'il contient au moins un point dans $\{0\} \times \mathbb{N}$ et au moins un point dans $\mathbb{N} \times \{0\}$.

Théorème II.18. [24] *Si S est un sous-ensemble fini normalisé de $\mathbb{Z}^2$ de cardinal σ, de taille convexe π, et inclus dans $[0, d_x] \times [0, d_y]$, alors nous pouvons calculer une application $U \in \mathrm{Aff}(\mathbb{Z}^2)$, ainsi que $U(S)$, tels que $U(S)$ est normalisé et contenu dans $[0, d'_x] \times [0, d'_y]$ avec $(d'_x + 1)\,(d'_y + 1) \leqslant 9\,\pi$, en utilisant $O(\sigma \log^2((d_x+1)\,(d_y+1)))$ opérations sur les bits.*

L'algorithme sous-jacent à ce théorème construit U à partir des trois transformations suivantes : $\lambda\colon (i,j) \mapsto (i-j, j)$, $\mu\colon (i,j) \mapsto (j,i)$, $\tau_k\colon (i,j) \mapsto (i+k, j)$. Une version simplifiée de la méthode peut être décrite comme suit :

1. Initialiser U avec l'identité.
2. Répéter :
   a) Si $d_x < d_y$ alors $S := \mu(S)$, $U := \mu \circ U$, échanger d_x et d_y.
   b) Calculer $b := d_x - \max_{(i,j)\in S}\,(i-j)$, $d := d_x + d_y - \max_{(i,j)\in S}\,(i+j)$, $f := d_y + \min_{(i,j)\in S}(i-j)$, $h := \min_{(i,j)\in S}(i+j)$.
   c) Si $b + f > d_y$ alors
      calculer $S := \tau_{d_y - f} \circ \lambda(S)$, $U := \tau_{d_y-f} \circ \lambda \circ U$, $d_x := d_x + d_y - b - f$.
   sinon si $d + h > d_y$ alors
      calculer $S := \tau_{-h} \circ \lambda^{-1}(S)$, $U := \tau_{-h} \circ \lambda^{-1} \circ U$, $d_x := d_x + d_y - d - h$.
   sinon retourner U.

Exemple II.19. Prenons $F := 1 + x\,y + x^5\,y^2$. Son support $S = \{(0,0), (1,1), (5,2)\}$ est représenté sur la figure II.1 ci-dessous. Après la première étape de l'algorithme, où λ est appliqué, S devient comme sur la partie gauche de la figure II.2. À la deuxième étape λ est appliquée une nouvelle fois puis l'algorithme s'arrête et retourne $U = \tau_1 \circ \lambda^2$, de sorte que $U(F) = x + y + x^2\,y^2$, dont le support est représenté sur la partie droite de la figure II.2.

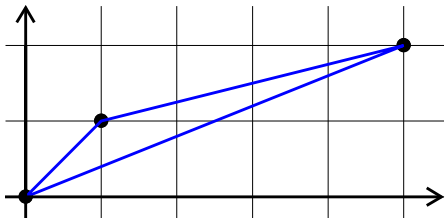

Figure II.1. $S = \{(0,0),(1,1),(5,2)\}$.

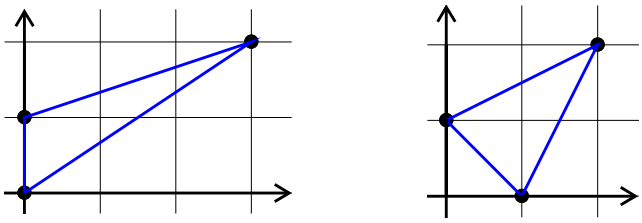

Figure II.2. S après une et deux étapes de l'algorithme de réduction.

L'utilisation du théorème précédent pour la factorisation des polynômes repose juste sur le lemme suivant :

Lemme II.20. *Si $F \in \mathbb{K}[x,y]$ n'est divisible ni par x ni par y, et si $U \in \mathrm{Aff}(\mathbb{Z}^2)$, alors le polynôme*

$$U(F) := \sum_{(e_x,e_y)\in \mathrm{supp}(F)} F_{(e_x,e_y)}\, x^{\alpha e_x+\beta e_y+\gamma}\, y^{\alpha' e_x+\beta' e_y+\gamma'}$$

est irréductible dans $\mathbb{K}[x,y,x^{-1},y^{-1}]$ si, est seulement si, F est irréductible.

Afin de calculer la factorisation irréductible de F, nous pouvons par conséquent calculer une application U *via* le théorème précédent. Ensuite nous calculons la décomposition irréductible de $U(F)$ et appliquons enfin U^{-1} sur chacun des facteurs. De cette façon le coût total dépend de la taille convexe de F au lieu de sa taille dense $(d_x+1)\,(d_y+1)$.

Mentionnons qu'il est aussi possible d'étendre plusieurs des techniques conçues pour le cas dense afin de prendre en compte non seulement le polytope de Newton de F mais aussi sa taille creuse. Nous renvoyons le lecteur vers les articles [1, 2, 87, 92, 165, 294, 295, 306, 307]. L'avantage du théorème précédent est sa simplicité de mise en œuvre, et aussi le fait qu'il peut être utilisé pour les autres types de factorisations.

II.7 Racines d'un polynôme dans un anneau local

La recherche des racines d'un polynôme est un cas particulier de la factorisation qui a comme exemple d'application le décodage en liste des codes de Reed-Solomon. Avec J. Berthomieu et G. Quintin, nous avons travaillé sur cette recherche de racines dans le cas des codes de Reed-Solomon généralisés à un anneau de base tel que $\mathbb{Z}_p/p^k\,\mathbb{Z}$ ou $\mathbb{K}[[x]]/(x^k)$ [25].

II.7.1 Notations

Soit R un *anneau local régulier complet intègre commutatif noetherien et non ramifié* de dimension finie r, avec $\mathfrak{m}$ comme idéal maximal. Notons p la caractéristique de son corps résiduel $\kappa := R/\mathfrak{m}$, et posons $R_i := \mathfrak{m}^i/\mathfrak{m}^{i+1}$, pour tout $i \geqslant 0$. Le fait que R soit non ramifié implique que soit $p=0$ soit p n'appartient pas à $\mathfrak{m}^2$. Plus précisément nous avons l'alternative classique suivante [50] :

- Si R et κ ont la même caractéristique, alors R est isomorphe à l'anneau des séries entières $\kappa[[t_1,\ldots,t_r]]$. Dans ce cas, on identifie R_i au sous-groupe de R des polynômes homogènes en $t_1,\ldots,t_r$ à coefficients dans κ et de degré i, de sorte que $(R_i)_{i\in\mathbb{N}}$ définit une graduation de R.

– Sinon, si R et κ ont des caractéristiques differentes, alors R est isomorphe à l'anneau des séries entières $D[[t_1, ..., t_{r-1}]]$, où D est un anneau à valuation discrète complet dont l'idéal maximal est engendré par p. Chaque élément de R peut s'écrire de façon unique comme $\sum_{e\in\mathbb{N}^r} c_e\, t_1^{e_1}\cdots t_{r-1}^{e_{r-1}}\, p^{e_r}$, avec les c_e dans κ. On identifie toujours R_i au sous-ensemble de R des expressions polynomiales en $t_1, ..., t_{r-1}$ et p de degré i avec des coefficients dans κ, mais $(R_i)_{i\in\mathbb{N}}$ ne définit plus une graduation sur R. Dans ce cas on pose $t_r := p$.

Dans les deux cas la fonction $\nu\colon R\to\mathbb{N}\cup\{+\infty\}$, qui envoie 0 sur $+\infty$, et $a\neq 0$ sur le plus grand entier i tel que $a\in\mathfrak{m}^i$, est une valuation. N'importe quel element a de R peut être représenté de façon unique par la somme convergente $\sum_{i\geqslant 0}[a]_i$, où $[a]_i\in R_i$ est la composante *homogène de valuation* i de a. Les éléments de R_i sont dits *homogènes de valuation* i.

Soit $K := \mathrm{Quot}\, R$ le *corps des fractions* de R. Puisque R est complet alors il en va de même pour K, et la valuation s'étend naturellement sur K. Le sous-ensemble des éléments de K de valuation au moins i est noté O_i. Si a est un élément de K, et si i est un entier, alors on note $a+O_i$ l'ensemble des éléments de K dont le développement coïncide avec celui de a jusqu'en precision i. On dit qu'une telle *classe* $a+O_i$ est une racine de F à la précision n si tous ses éléments annulent F à la precision n.

II.7.2 Calcul des racines

Nous avons étudié un algorithme de recherche des racines, disons naïf, qui est bien adapté à des problèmes de petites tailles, et un algorithme plus sophistiqué faisant intervenir des sous routines, telles que la remontée de Hensel, qui se comportent asymptotiquement en temps quasi-linéaire. Nos algorithmes fonctionnent dans le cas général précédemment présenté, mais pour des raisons de concision nous ne donnons ici que des résultats de complexité dans des cas particuliers.

Nous nous sommes intéressés au calcul de toutes les racines d'un polynôme $F\in R[x]$ donné à précision fixe n, c'est-à-dire modulo $\mathfrak{m}^n$. Les cas usuels concernent $R=\mathbb{Z}_p$ ou $R=\mathbb{K}[[t]]$, pour tout corps $\mathbb{K}$. Nous avons adapté des techniques plus ou moins classiques dans ce cadre général, avons analysé leur complexité et avons observé leur performance pratique en C++ au sein de notre logiciel Mathemagix.

Théorème II.21. [25] *Soit $\mathbb{K}$ un corps, soit R l'anneau des séries $\mathbb{K}[[t]]$, et soit F un polynôme de $R[x]$ de degré au plus d et donné à precision n. Il est possible de calculer un ensemble d'au plus d classes disjointes représentant l'ensemble des racines de F dans R à précision n, au moyen :*

– *du calcul des racines dans $\mathbb{K}$ de polynômes séparables dans $\mathbb{K}[x]$ de degré au moins 2, et dont la somme des degrés n'excède pas $2(d-1)$,*
– *si $p>0$, de l'extraction de racines p-ièmes itérées d'au plus $O(n\,d/p)$ éléments de $\mathbb{K}$,*
– *et de $O(n\,\mathsf{M}(n)\,\mathsf{M}(d)\log d)$ opérations arithmétiques dans $\mathbb{K}$.*

Théorème II.22. [25] *Soit $R=\mathbb{F}_q[[t]]$ l'anneau des séries à coefficients dans le corps fini à $q=p^k$ elements, et soit F un polynôme de $R[x]$ de degré au plus d donné à précision n. Il est possible de calculer un ensemble d'au plus d classes disjointes représentant l'ensemble des racines de F dans R à précision n en utilisant en moyenne*

$$O\left((n\,\mathsf{M}(n)+\log(d\,q))\,\mathsf{M}(d)\log d + n\,\frac{d}{p}\log(q/p)\right)$$

opérations dans $\mathbb{F}_q$.

Nous obtenons ainsi une borne de complexité quasi-linéaire en d et quasi-quadratique en n. Dans notre article [25] nous étudions aussi le cas plus difficile où la dimension de R est supérieure ou égale à deux.

II.7.3 Application

Dans leur article [122], V. Guruswami et M. Sudan ont conçu un algorithme un temps polynomial pour résoudre le problème du décodage en liste des codes de Reed-Solomon sur un corps fini $\mathbb{F}_q$. Leur méthode calcule d'abord un polynôme F de $\mathbb{F}_q[t][x]$ dont les racines dans $\mathbb{F}_q[t]$ contiennent tous les messages transmis possibles. Ensuite il ne reste plus qu'à déterminer ces racines. Nos méthodes de la section précédente peuvent alors être utilisées pour cette deuxième étape.

Soit E une extension non ramifiée de $\mathbb{Z}_p$ de degré k. Le quotient $E/(p^n)$, noté $\mathrm{GR}(p^n, k)$, est appelé l'*anneau de Galois* d'ordre $n\,k$ et de caractéristique p^n. Nous nous sommes intéressés aux codes de Reed-Solomon définis sur $\mathrm{GR}(p^n, k)$ et plus particulièrement au problème du calcul des racines dans $\mathrm{GR}(p^n, k)[t]$ d'un polynôme $F \in \mathrm{GR}(p^n, k)[t][y]$. Si l borne le degré de l'une de ses racines, alors nous montrons comment la calculer comme l'une des racines de F vu comme polynôme de $R[x]$ à précision n où $R := \mathrm{GR}(p^n, k)[x]/\varphi(x)$ avec φ de degré $e = d\,l + d_t + 1$ et irréductible modulo p. Nous pouvons alors utiliser le résultat suivant :

Théorème II.23. [25] *Soit R une extension non ramifiée de $\mathbb{Z}_p$ de degré k, et soit F un polynôme de $R[x]$ de degré au plus d donné à précision n. Il est possible de calculer un ensemble d'au plus d classes disjointes représentant l'ensemble des racines de F dans R à précision n en utilisant en moyenne $\tilde{O}((n + k \log p)\, n\, d\, k \log p)$ opérations sur les bits.*

Enfin, mentionnons que nous avons programmé ces algorithmes dans Mathemagix, ce qui nous a permis d'illustrer leurs performances pour différentes tailles de codes.

Chapitre III

Résolution des systèmes algébriques

Ce chapitre est consacré à la résolution des systèmes algébriques. Dans un premier temps, je rappelle rapidement le cadre historique et les problématiques. J'énonce ensuite les principales définitions et résultats obtenus pendant ma thèse de doctorat. Mes contributions depuis sont de trois natures distinctes : simplification des preuves des algorithmes, implantation en C++, extension pour traiter plus efficacement des problèmes de géométrie algébrique réelle.

III.1 Représentation des polynômes éliminant

La résolution des systèmes algébriques est un axe de recherche majeur en calcul formel depuis les années soixante. Les algorithmes les plus populaires concernent les bases standards, les ensembles triangulaires, et les matrices de résultant, comme en témoignent par exemple plusieurs ouvrages généralistes [16, 53, 54, 72, 91, 113, 218, 289] et logiciels de calcul formel.

Les méthodes que nous venons de mentioner ont toutes en commun l'utilisation de représentations de polynômes à plusieurs variables sous forme développée, dense ou creuse. Les polynômes à plusieurs variables sont alors vus comme vecteurs de leurs coefficients dans la base monomiale usuelle et chaque opération élémentaire d'élimination consiste en une opération de type pivot une fois interprétée dans un cadre d'algèbre linéaire. Pour ces raisons et compte tenu de la parenté de l'algorithme de Buchberger avec l'algorithme de Knuth-Bendix en théorie des langages, ces méthodes sont souvent regroupées sous la terminologie de *méthodes par réécriture* [66]. Leur complexité dans le cas le pire est doublement exponentielle [67, 209].

Une représentation *fonctionnelle* des polynômes consiste à stocker un polynôme sous la forme d'un programme l'évaluant en un point en ne faisant appel qu'à des additions, soustractions et multiplications. Les structures de données usuelles sont des programmes sans branchement et sans division (*straight-line programs* en anglais) ou bien des graphes acycliques orientés. La *taille* d'une telle représentation fonctionnelle correspond essentiellement au nombre d'opérations arithmétiques élémentaires d'anneau (d'arité une ou deux) nécessaires pour évaluer le polynôme en un point. Dans le cas d'une représention par programmes, il s'agit du nombre de ses instructions. Avec une représentation par graphes, il s'agit simplement de la taille du graphe.

L'utilisation de représentations fonctionnelles pour les polynômes éliminant apparaît pour la première fois dans les travaux de M. Giusti, J. Heintz au début des années quatre-vingt dix [101]. Néanmoins le premier algorithme complet exploitant naturellement cette représentation a été l'aboutissement de plusieurs travaux [75, 99, 103, 104, 105, 106, 182, 207, 220, 233], dont une présentation synthétique se trouve par exemple dans l'introduction de [70]. C'est dans cette voie que se sont situés principalement mes travaux de thèse de doctorat : le résultat de complexité le plus général est énoncé dans [189].

Voici un premier exemple très simple pour illustrer l'intérêt des représentations fonctionnelles pour les problèmes d'élimination : la résolution d'un système linéaire carré à coefficients indéterminés par les méthodes par réécriture fera nécessairement apparaître le polynôme déterminant sous forme développée et donc un nombre factoriel de monômes. En revanche, il est bien connu que ce polynôme déterminant s'évalue en temps polynomial, par exemple avec l'algorithme de Berkowitz [19].

Présentons un deuxième exemple plus compliqué mais de façon informelle. Considérons un système algébrique générique composé de n équations de degré d en $2\,n$ variables, alors l'ensemble de ses solutions a un degré d^n et les polynômes éliminant en n variables

sont généralement de degré d^n, et donc composés d'un nombre de monômes qui croît en d^{n^2} pour d fixé, lorsque n tend vers l'infini. Dans cette situation, j'ai montré que de tels polynômes éliminant peuvent être évalués en un nombre d'opérations arithmétiques qui croît seulement avec d^n [189].

III.2 Résolution géométrique

Cette section contient les principales définitions et concepts utiles, ainsi qu'une vue d'ensemble de l'algorithme central dit de résolution géométrique.

III.2.1 Définitions

Soit I un idéal de $\mathbb{K}[x_1, ..., x_n]$ de dimension r.

Définition III.1. *I est dit en* position de Noether *si $\mathbb{K}[x_1, ..., x_r] \cap I = (0)$ et $\mathbb{K}[x_1, ..., x_n]/I$ est une extension entière de $\mathbb{K}[x_1, ..., x_r]$. Cette position est dite* forte *si, de plus, toutes les relations de dépendance intégrales sont unitaires.*

Il est connu que pour presque toute matrice inversible $M \in \mathcal{M}_{n\times n}(\mathbb{K})$, les idéaux $I \circ M = \{f(M(x_1, ..., x_n)) \mid f \in I\}$ sont en position de Noether forte.

Définition III.2. *Soit W un fermé algébrique de $\bar{\mathbb{K}}^n$ défini sur $\mathbb{K}$, équidimensionel de dimension r. Une* fibre de remontée *pour W est la donnée des objets suivants :*

- *un système de remontée $g_1, ..., g_{n-r}$ tel que W est une réunion de composantes isolées simples de $\mathcal{V}(g_1, ..., g_{n-r})$.*
- *une matrice $n \times n$ inversible M à coefficients dans $\mathbb{K}$ telle que les coordonnées y définies par $y := M^{-1}x$ soient en position de Noether forte pour W ;*
- *un point dit de remontée $a \in \mathbb{K}^r$ tel que la matrice jacobienne de $g_1, ..., g_{n-r}$ en les variables $y_{r+1}, ..., y_n$ soit inversible en chacun des points de $W^p = W \cap \mathcal{V}(y_1 - a_1, ..., y_r - a_r)$.*
- *une forme linéaire $u := \lambda_{r+1}\, y_{r+1} + \cdots + \lambda_n\, y_n$, à coefficients λ_i dans $\mathbb{K}$, séparant les points de W^p, c'est à dire prenant des valeurs distinctes en des points distincts ;*
- *le polynôme minimal q de u dans $\mathbb{K}[W^p]$;*
- *les paramétrisations $v_{r+1}, ..., v_n$ des coordonnées des points de W^p :*
$$W^p = \{(a_1, ..., a_r, v_{r+1}(\alpha), ..., v_n(\alpha)) \mid q(\alpha) = 0\},$$
où les v_i appartiennent à $\mathbb{K}[T]$ et ont un degré en T strictement inférieur à $\deg(W)$.

Notons $\mathbb{K}[x_1, ..., x_n]_g$ l'anneau des fractions $\mathbb{K}[x_1, ..., x_n]$ ayant une puissance de g comme dénominateur.

Définition III.3. *Une suite de polynômes $f_1, ..., f_s$ est dite* régulière *si f_{i+1} n'est pas diviseur de zéro dans $\mathbb{K}[x_1, ..., x_n]_g/(f_1, ..., f_i)$, pour $i \in \{0, ..., s-1\}$. Une telle suite est dite de plus* réduite *si les idéaux $(f_1, ..., f_i)$ sont radicaux pour tout $i \in \{1, ..., s-1\}$.*

Si $f_1, ..., f_n$ est une suite régulière réduite dans $\mathbb{K}[x_1, ..., x_n]_g$, alors $V_i := \overline{\mathcal{V}(f_1, ..., f_i) \setminus \mathcal{V}(g)}$ est vide ou de codimension i.

III.2.2 Complexité

Pour les estimations de complexité nous adoptons les notations suivantes :

- d est un majorant du degré de $f_1, ..., f_n$;
- L est un majorant de la taille en évaluation de $f_1, ..., f_n, g$;
- δ est le maximum des degrés des V_i, pour $i \in \{1, ..., n\}$.

L'algorithme Kronecker calcule alors une fibre de remontée pour V_{i+1} à partir d'une fibre de V_i, pour i de 1 à n, par la méthode des *courbes de remontée* que l'on peut résumer comme suit, en supposant dès le départ les coordonnées $x_1, ..., x_n$ suffisamment génériques :

- *Remontée :* à partir d'une fibre de remontée correspondant à l'ensemble des points de $V_i \cap \mathcal{V}(y_1 - a_1, ..., y_{n-i} - a_{n-i})$, on calcule une paramétrisation de la *courbe de remontée* $V_i \cap \mathcal{V}(y_1 - a_1, ..., y_{n-i-1} - a_{n-i-1})$. Cette partie du calcul repose essentiellement sur une variante de la méthode de Newton. L'hypothèse de radicalité de $(f_1, ..., f_i) : g^\infty$ est cruciale pour pouvoir prendre $f_1, ..., f_i$ comme système de remontée pour la fibre de V_i.
- *Intersection :* on intersecte cette courbe avec l'hypersurface définie par l'équation suivante $f_{i+1} = 0$. Sous l'hypothèse de régularité cette intersection est un nombre fini de points.
- *Localisation :* la fibre de remontée de V_{i+1} est déduite de la précédente intersection en supprimant les points contenus dans l'hypersurface $\mathcal{V}(g)$.

Théorème III.4. [107] *Soient $\mathbb{K}$ est un corps de caractéristique zéro et $f_1, ..., f_n$ et g des polynômes de $\mathbb{K}[x_1, ..., x_n]$ de degré au plus d, de taille L en évaluation et tels que $f_1, ..., f_n$ forment une suite régulière réduite dans $\mathbb{K}[x_1, ..., x_n]_g$. Alors nous disposons d'un algorithme probabiliste calculant une fibre de remontée de $\mathcal{V}(f_1, ..., f_n) \setminus \mathcal{V}(g)$ en $n\,(n\,L + n^4)\tilde{O}(d^2\,\delta^2)$ opérations arithmétiques sur le corps $\mathbb{K}$.*

La probabilité de succès repose sur des choix d'éléments de $\mathbb{K}$ et peut être uniformément minorée. Les mauvais choix sont contenus dans des fermés algébriques stricts.

Lorsque le corps de base $\mathbb{K}$ est le corps des nombres rationnels $\mathbb{Q}$ et que $(f_1, ..., f_n) : g^\infty$ est radical, la résolution s'effectue d'abord modulo un nombre premier de petite taille en conservant la complexité ci-dessus, puis la résolution modulaire est remontée en une résolution rationnelle en $n^2\,(L + n^4)\tilde{O}(D\,\eta)$ opérations sur les bits, où D représente le nombre de solutions du système et η la taille des nombres entiers de la fibre de remontée finale sur $\mathbb{Q}$. Par exemple si $f_1, ..., f_n$ sont de degré $d = 2$ et de taille de coefficients h, alors $L \in O(n^3)$, le nombre de solutions est en général $D = 2^n$, la résolution modulo un nombre premier coûte alors $\tilde{O}(D^{2n})$ et la remontée des solutions sur $\mathbb{Q}$ utilise $\tilde{O}(D\,\eta)$ opérations sur les bits. Par ailleurs les équivalents arithmétiques de la borne de Bézout montrent que η croît avec $D\,h$ [183, 210], ce qui implique que le coût de l'algorithme de résolution géométrique est quasi-linéaire dans ce cas.

Aussi, j'ai implanté cet algorithme dans le système de calcul formel Magma [31] pendant ma thèse. La bibliothèque a été appelée Kronecker en hommage aux travaux fondateurs de L. Kronecker pour la théorie de l'élimination [184]. Plus récemment je l'ai reprogrammé d'une façon autonome en C++, dans la bibliothèque geomsolvex de Mathemagix [147].

III.2.3 Remarques

Précisons qu'une autre étude de complexité et de coût en mémoire de l'algorithme de résolution géométrique a été réalisée indépendamment par [127]. Les méthodes par évaluation ont été ensuite utilisées avec succès pour résoudre des problèmes surdéterminés [110], paramétrés [29, 125, 252], de type Pham [234], creux [153], ainsi que sur des corps finis [39, 40]. Elles sont aussi bien adaptées aux problèmes de géométrique algébrique réelle [9, 10, 11, 247]. Mon implantation de l'algorithme en Magma a été utilisée pour résoudre des problèmes relevant de la cryptographie [96], pour un modèle de compression d'image inspiré des mécanismes de perception rétinienne [205], ainsi que pour la conception de bases d'ondelettes [196]. Il est aussi intéressant de pointer ici vers une adaptation numé-

rique de l'approche de résolution incrémentale [272]. Des comparaisons théoriques entre techniques numériques et formelles peuvent être trouvées dans [44, 45, 46, 58]. Enfin, quant aux questions liées à l'optimalité de l'algorithme de résolution géométrique, nous renvoyons le lecteur vers les articles [43, 75, 102, 126, 233]. En un mot, sous des hypothèses techniques pour modéliser la notion d'algorithmes généraux, le résultat principal de [43] nous apprend que l'algorithme de résolution géométrique appartient bien à une certaine classe de complexité optimale.

III.3 Décomposition équidimensionnelle

Après l'algorithme de résolution géométrique, j'ai porté mes efforts dans deux directions, théorique et pratique, pour lever les hypothèses de régularité et de radicalité. Dans [187], je propose une première solution reposant sur l'utilisation du premier théorème de Bertini : les équations d'origine sont remplacées par des combinaisons linéaires génériques $g_i := t_{i,1}\, f_1 + \cdots + t_{i,s}\, f_s$ pour i de 1 à $n+1$. Les variétés intermédiaires V_i deviennent $V_i := \mathcal{V}(g_1, ..., g_i)$, l'algorithme reste essentiellement incrémental, la grandeur δ est alors le maximum des degrés géométriques des variétés intermédiaires.

Théorème III.5. [187] *Soient $f_1, ..., f_s$ des polynômes de $\mathbb{K}[x_1, ..., x_n]$ avec $\mathbb{K}$ un corps de caractéristique nulle. Il existe un sous-ensemble dense G de $\mathbb{K}^{(n+1)s}$ et un algorithme tels que : pour tout $(t)_{i,j}$ dans G, l'algorithme calcule une représentation minimale des solutions du système $f_1 = \cdots = f_s = 0$ avec un coût non-uniforme polynomial en $L\, n\, s\, d\, \delta$. Si α est un entier positif et si Ω est un ensemble de points de taille $3\,\alpha\,(d+1)^{3n}$ dans $\mathbb{K}$, un point $(t)_{i,j}$ choisi uniformément dans $\Omega^{(n+1)s}$ est dans G avec une probabilité au moins $1 - 2\,(n+1)/\alpha$.*

Des variations et améliorations de ce résultat ont été ensuite proposées dans [152, 154, 155, 156]. D'un point de vue pratique, les composantes de dimension positive solutions du système sont codées par des fibres de remontée et deux possibilités sont alors offertes à l'utilisateur :

- une *fonction de passage* permet de calculer d'autres fibres ;
- une *fonction d'écriture* permet de calculer efficacement des polynômes éliminant codés par des polynômes à plusieurs variables en représentation classique (dense ou creuse). Ici le coût des opérations élémentaires sur les séries à plusieurs variables est crucial.

L'algorithme sous-jacent au précédent théorème présente l'inconvénient de devoir remplacer les équations de départ par des combinaisons linéaires génériques de celles-ci : la complexité d'évaluation d'une seule équation devient celle de tout le système d'entrée. Pour remédier à cet inconvénient, j'ai proposé une généralisation de l'opérateur de Newton adaptée aux situations singulières, afin de s'affranchir des hypothèses de radicalité. L'algorithme de résolution en découlant est bien plus naturel, élégant et efficace. Plus précisément, il calcule la décomposition équidimensionnelles de chaque

$$V_i = \overline{\mathcal{V}(f_1, ..., f_i) \setminus \mathcal{V}(g)} \quad \text{en} \quad V_i = \cup_{j=0}^{i} W_{i,j},$$

où $W_{i,j}$ représente la composante équidimensionnelle de codimension j de V_i. En notant $\deg(W)$ le degré d'une composante irréductible W et $\mathrm{mult}(W; f_1, ..., f_i)$ sa multiplicité comme solution du système $f_1 = \cdots = f_i = 0$ (multiplicité au point générique) nous définissons le *degré algébrique* δ_i de V_i comme

$$\delta_i := \sum_{W \in \text{Composantes irréductibles de } V_i} \mathrm{mult}(W; f_1, ..., f_i) \deg(W).$$

En prenant cette fois-ci pour δ le maximum de ces δ_i, nous obtenons le résultat suivant :

Théorème III.6. [189] *Soit* $\mathbb{K}$ *un corps de caractéristique zéro,* f_1, ..., f_s *et* g *des polynômes de* $\mathbb{K}[x_1, ..., x_n]$ *de taille en évaluation* L. *Il existe un algorithme probabiliste qui calcule la décomposition équidimensionnelle de la clôture de Zariski de l'ensemble des zéros du système* $f_1 = \cdots = f_s = 0$, $g \neq 0$ *avec une complexité, en cas de succès, dans*

$$s\, n^4\, (n\, L + n^{\Omega})\, \tilde{O}(d^3\, \delta^3).$$

La probabilité de succès repose sur des choix d'éléments de $\mathbb{K}$. *Les mauvais choix sont contenus dans des fermés algébriques stricts.*

Pour s'affranchir de l'hypothèse de radicalité des idéaux $(f_1, ..., f_i) : g^{\infty}$, ma généralisation de la méthode de Newton publiée dans l'article [188] peut se résumer comme suit : un point singulier isolé de multiplicité m peut-être approché de façon quadratique en temps quadratique en m par un algorithme qui généralise l'opérateur de Newton. Il s'agit d'une méthode dite de *déflation* : par dérivations successives, on construit une suite de systèmes qui ont le même zéro mais avec une multiplicité qui décroît. Mentionnons ici que le produit de séries utilisé dans [188] a été amélioré dans [253]. Le théorème précédent à été utilisé dans [10, 246] pour des problèmes de géométrie algébrique réelle.

Une fois connue la décomposition équidimensionnelle, nos algorithmes de factorisation présentés dans le chapitre précédent peuvent être utilisés directement pour en déduire la décomposition irréductible. Avant que je ne commence à travailler sur la complexité de la factorisation, les coûts théoriques et pratiques de la factorisation étaient plus élevés que le calcul des composantes équidimensionnelles (informellement au moins en δ^4 avec la méthode de [82]).

III.4 Décomposition primaire

Après mes travaux sur les décompositions équidimensionnelles et irréductibles, est venue naturellement la question de la décomposition primaire des idéaux de polynômes. Avec V. Cossart nous avons proposé ce sujet de thèse à C. Durvye.

Dans un premier travail en collaboration avec C. Durvye, nous donnons une nouvelle preuve simplifiée et concise de l'algorithme Kronecker [70]. Nous montrons aussi comment nos preuves constructives permettent de retrouver plusieurs résultats classiques élémentaires de façon purement géométrique, sans l'utilisation par exemple du polynôme du Hilbert. L'approche mathématique pour la dimension utilise la position de Noether d'un idéal et les outils classiques sur les bases de transcendance des corps. Quant au degré et à l'inégalité de Bézout nous utilisons simplement les représentations des fermés algébriques par fibres de remontée et étudions les propriétés des polynômes caractéristiques des morphismes de multiplication dans les anneaux de coordonnées comme expliqué dans les paragraphes suivants. Ce point de vue est assez proche de celui du livre classique de I. R. Shafarevich [259]. Au final l'algorithme Kronecker peut être désormais présenté au niveau de formation d'un master en mathématique ou informatique théorique sans prérequis en géométrie algébrique [33].

Supposons maintenant que I soit en position de Noether forte et de *dimension pure* r (c'est à dire que tous ses premiers associés sont de dimension exactement r). Soit $f \in \mathbb{K}[x_1, ..., x_n]$ et notons χ le polynôme caractéristique de la multiplication par f dans $\mathbb{K}(x_1, ..., x_r)[x_{r+1}, ..., x_n]/I$. On peut montrer que χ appartient en fait à $\mathbb{K}[x_1, ..., x_r][T]$. En notant χ_0 le coefficient constant de χ, nous avons le résultat suivant qui nous fournit une manière précise pour calculer les polynômes éliminant nécessaires aux fibres de remontées successives dans l'algorithme de résolution géométrique :

Théorème III.7. [70] *Soit* I *un idéal de dimension pure* r *en position de Noether forte, et soit* f *un non diviseur de zéro dans* $\mathbb{K}[x_1, ..., x_n]/I$ *tel que* $J := I + (f)$ *soit propre et en position de Noether forte. Alors* $\chi_0(x_1, ..., x_{r-1}, T)$ *est proportionnel sur* $\mathbb{K}$ *au polynôme caractéristique de* x_r *dans* $\mathbb{K}(x_1, ..., x_{r-1})[x_r, ..., x_n]/J$.

Outre les aspects algorithmiques nous retrouvons dans cet énoncé une forme faible de l'inégalité de Bézout puisque le degré total de χ_0 est au plus $\deg I \deg f$.

Si f_1, ..., f_n forment une suite régulière réduite, la dernière étape de l'algorithme de résolution consiste à calculer l'intersection de la courbe $\overline{\mathcal{V}(f_1,...,f_{n-1}) \setminus \mathcal{V}(g)}$ avec l'hypersurface $\mathcal{V}(f_n)$. Le théorème précédent nous permet alors d'obtenir le polynôme caractéristique Q de la multiplication par x_1 dans $\mathbb{K}[x_1, ..., x_n]/(f_1, ..., f_n) : g^\infty$. Si les puissances de x_1 engendrent l'algèbre réduite $\mathbb{K}[x_1, ..., x_n]/\sqrt{(f_1,...,f_n) : g^\infty}$, alors les multiplicités des solutions du système $f_1 = \cdots = f_n = 0$, $g \neq 0$ se déduisent directement de celles de Q [70]. Par conséquent l'algorithme de résolution géométrique calcule naturellement, et sans surcoût, les multiplicités des solutions du système.

L'information sur les multiplicités ne conduit néanmoins pas directement à la décomposition primaire de $(f_1,...,f_n) : g^\infty$. Pour sûr des calculs de bases standard locales [206, 219, 222, 305] permettent d'obtenir une description de l'anneau local, mais pas dans un coût polynomial en la multiplicité à ma connaissance. La solution proposée par C. Durvye passe par une version algorithmique du théorème précédent.

Théorème III.8. [69] *Soit $\mathbb{K}$ un corps de caractéristique zéro. Supposons que $f_1,...,f_s$, g soient $s+1$ polynômes de $\mathbb{K}[x_1,...,x_n]$ de taille L donnés en évaluation, et tels que le système $f_1 = \cdots = f_s = 0$, $g \neq 0$ admette un ensemble fini de solutions. Si les degrés $d_1,...,d_s$ respectifs de $f_1,...,f_s$ sont dans l'ordre $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_s > 1$, en posant $D = d_1 \cdots d_n$, alors on peut calculer une fibre de remontée de $(f_1,...,f_s) : g^\infty$, ainsi qu'une décomposition de l'algèbre $\mathbb{K}[x_1,...,x_n]/(f_1,...,f_s) : g^\infty$ en sous-algèbres pour lesquelles sont données une base monomiale et la matrice de multiplication de chacune des variables dans cette base, en temps $\tilde{O}(D^{11} + (L + n\,s)\,D^6)$ avec un algorithme probabiliste de probabilité d'erreur bornée par $1/2$.*

Les exposants dans la borne de coût de ce théorème semblent relativement élevés, mais il faut les relativiser un peu car d'une part ils reposent sur des algorithmes plutôt naïfs pour le calcul des réductions de Hermite et formes de Smith dans des anneaux locaux, et d'autre part ces exposants opèrent principalement sur les seules racines multiples. Il est en effet rare en pratique qu'un système ait beaucoup de telles racines avec des multiplicités élevées.

À l'heure actuelle ce résultat n'a pas été étendu au cas général. L'une des obstructions est de trouver de bonnes représentations des algèbres locales associées aux composantes de dimension positive. L'autre difficulté est le calcul des idéaux primaires immergés avec une bonne complexité. Mentionnons ici que ces derniers peuvent être obtenus en utilisant les méthodes par déflation : plus précisément un premier immergé peut être obtenu comme un premier isolé d'un système déflaté *ad hoc* du système initial [198].

III.5 Lieux de dégénérescence

Dans cette section nous abordons un type de calcul assez général qu'une extension l'algorithme de résolution géométrique couvre naturellement. Il s'agit d'un travail récent en collaboration avec B. Bank, M. Giusti, J. Heintz, G. Matera, P. Solernó [8].

III.5.1 Définition

Supposons donnée une suite de polynômes $g_1,...,g_p,h$ de $\mathbb{Q}[x_1,...,x_n]$ telle que $g_1,...,g_p$ forment une suite régulière réduite dans $\mathbb{Q}[x_1,...,x_n]_h$ et que $(g_1,...,g_p) : h^\infty$ soit radical. Nous notons V le sous-ensemble de $\mathbb{C}^n$ défini par $V := \mathcal{V}(g_1,...,g_p) \setminus \mathcal{V}(h)$, supposé lisse et tel que $\bar{V}$ est équidimensionel de dimension $r := n - p$.

Soit s un entier tel que $s \geqslant p+r$, et supposons donnée une matrice $F = (f_{k,l})_{k,l}$ de polynômes de $\mathbb{Q}[x_1, ..., x_n]$ de taille $p \times s$. Pour tout $z \in V$, notons $\mathrm{rank}(F(z))$ le rang de la matrice F évaluée au point z, et définissons $W := \{z \in V \,|\, \mathrm{rank}(F(z)) = p\}$. Fixons ensuite une matrice complexe de taille $(s-p) \times s$ que l'on supposera de rang maximal, et suffisamment générique au fur et à mesure de nos besoins

$$a := \begin{bmatrix} a_{1,1} & \cdots & a_{1,s} \\ \vdots & & \vdots \\ a_{s-p,1} & \cdots & a_{s-p,s} \end{bmatrix} \in \mathbb{C}^{(s-p)\times s}.$$

Posons, pour tout $i \in \{1, ..., r+1\}$,

$$a_i := \begin{bmatrix} a_{1,1} & \cdots & a_{1,s} \\ \vdots & & \vdots \\ a_{s-p-i+1,1} & \cdots & a_{s-p-i+1,s} \end{bmatrix} \in \mathbb{C}^{(s-p-i+1)\times s}.$$

Nous avons $\mathrm{rank}(a_i) = s-p-i+1$. Enfin notons

$$T(a_i) := \begin{bmatrix} F \\ a_i \end{bmatrix} \quad \text{et} \quad W(a_i) := \{z \in W \,|\, \mathrm{rank}(T(a_i)(z)) < s-i+1\}.$$

En utilisant [71, Theorem 3] ou [208, Theorem 13.10], nous obtenons que toutes les composantes irréductibles de $\overline{W(a_i)}$ sont de codimension au plus i dans $\bar{W}$. Pour i de 1 à r, les $W(a_i)$ forment une chaine décroissante:

$$W \supseteq W(a_1) \supseteq \cdots \supseteq W(a_r).$$

La variété $W(a_i)$ est appelée le i-ième *lieu de dégénérescence* de (V, F) pour la matrice a.

Exemple III.9. Afin d'illustrer les notations prenons $p=1$, $n=3$, $g_1 := x_1^2+x_2^2+x_3^2-1 \in \mathbb{Q}[x_1, x_2, x_3]$, et $h=1$. Alors $V := \mathcal{V}(g)$ est une sous-variété irréductible lisse de $\mathbb{C}^3$ de dimension $r := 2$. Penons pour $F = [\, 2\,x_1 \;\; 2\,x_2 \;\; 2\,x_3 \,]$ la matrice jacobienne de g. Avec ces quantités nous avons

$$T(a_1) = \begin{bmatrix} 2\,x_1 & 2\,x_2 & 2\,x_3 \\ a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}, \qquad T(a_2) = \begin{bmatrix} 2\,x_1 & 2\,x_2 & 2\,x_3 \\ a_{1,1} & a_{1,2} & a_{1,3} \end{bmatrix}.$$

Par conséquent $W = V$, $W(a_1) = \{z \in V \,|\, \det(T(a_1)(z)) = 0\}$ et $W(a_2) = \{(z_1, z_2, z_3) \in V \,|\, a_{1,2}\,z_1 - a_{1,1}\,z_2 = 0, a_{1,3}\,z_1 - a_{1,1}\,z_3 = 0, a_{1,3}\,z_2 - a_{1,2}\,z_3 = 0\}$. Le fait de supposer que la matrice $[a_{i,j}]_{1\leq i\leq 2, 1\leq j\leq 3}$ est générique, implique que $W(a_1)$ est équidimensionnelle de dimension 1 et que $W(a_2)$ est la variété polaire classique de la sphère V, qui peut être paramétrée de la façon suivante :

$$W(a_2) = \mathcal{V}\left(\left(\frac{a_{1,1}^2}{a_{1,3}^2} + \frac{a_{1,2}^2}{a_{1,3}^2} + 1\right) x_3^2 - 1, x_1 - \frac{a_{1,1}}{a_{1,3}}\,x_3, x_2 - \frac{a_{1,2}}{a_{1,3}}\,x_3\right).$$

III.5.2 Algorithme

L'algorithme de calcul des lieux de dégénérescence, que nous avons conçu sous ces hypothèses, procède de la façon suivante :

1. Calculer une fibre de remontée de $\bar{V}$ avec l'algorithme de résolution géométrique.
2. Avec une forte probabilité de succès nous pouvons choisir des matrices b_1, ..., b_{r+1} de taille $s \times s$ à coefficients dans $\mathbb{Q}$, puis les mineurs principaux maximaux Δ_1, ..., Δ_{r+1} des matrices $F \cdot b_1$, ..., $F \cdot b_{r+1}$, de sorte que l'on ait $W = V_{\Delta_1} \cup \cdots \cup V_{\Delta_{r+1}}$, où $V_{\Delta_i} := V \setminus \mathcal{V}(\Delta_i)$. Nous pouvons en déduire aisément une fibre de remontée pour chaque $\overline{V_{\Delta_i}}$.
3. Pour chaque $k \in \{1, ..., r+1\}$:
   a) Soit $m_{k,i}$ le mineur principal de taille $s-i$ de $T(a_i)\,b_k$.
   Calculer une fibre de remontée de $\overline{W(a_1)_{\Delta_k \cdot m_{k,1}}}$ à partir de la formule $W(a_1)_{\Delta_k} = V_{\Delta_k} \cap \mathcal{V}(\det(T(a_1)))$.

b) Pour chaque i de 1 à $r-1$, calculer une fibre de remontée de $\overline{W(a_{i+1})_{\Delta_k \cdot m_{k,i+1}}}$ à partir de $W(a_i)_{\Delta_k \cdot m_{k,i}} \cap \mathcal{V}(m_{k,i})$.

4. Calculer la réunion de tous les points de $W(a_r)_{\Delta_1}, ..., W(a_r)_{\Delta_{r+1}}$ de sorte à obtenir une paramétrisation sans redondance de $W(a_r)$.

Supposons maintenant que $g_1, ..., g_q, h$, $f_{k,l}$, pour $1 \leq k \leq p$ et $1 \leq l \leq s$ sont donnés dans une représentation fonctionnelle de taille L. Soit d une borne supérieure des degrés de $g_1, ..., g_q, h$ et $f_{k,l}$, pour $1 \leq k \leq p$ et $1 \leq l \leq s$. Soit δ^* le maximum des degrés des $\overline{W(a_i)}$ pour $1 \leq i \leq r$, soit $\delta_g := \max\{\deg(\overline{\mathcal{V}(g_1, ..., g_i) \setminus \mathcal{V}(h)}) \mid 1 \leq j \leq p\}$, et soit $\delta := \max\{\delta_g, \delta^*\}$.

Théorème III.10. [8] *Sous les hypothèses précédentes, une fibre de remontée de $W(a_r)$ peut être calculée par un algorithme probabiliste en utilisant au plus $L\ (s\ n\ d)^{O(1)}\ \delta^2$ opérations arithmétiques dans $\mathbb{Q}$. La probabilité de succès repose sur des choix d'éléments de $\mathbb{K}$ et peut être uniformément minorée. Les mauvais choix sont contenus dans des fermés algébriques stricts.*

Les constructions, énoncés et preuves sous-jacentes à ce résultat généralisent plusieurs des arguments des articles [11, 12, 13].

III.5.3 Variétés polaires

Une des applications importante du résultat précédent concerne le calcul des variétés polaires utilisées en géométrie algébrique réelle pour calculer un point par composante connexe. Soit $g_1, ..., g_p$ une suite régulière réduite de polynômes de $\mathbb{Q}[x_1, ..., x_n]$. La matrice jacobienne de $g_1, ..., g_p$ est notée

$$J(g_1, ..., g_p) := \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial g_p}{\partial x_1} & \cdots & \frac{\partial g_p}{\partial x_n} \end{bmatrix}.$$

Nous pouvons trouver une suite de matrices régulières $(b_1, ..., b_{r+1})$ de $\mathbb{Q}^{n \times n}$ et des p-mineurs $\Delta_1, ..., \Delta_{r+1}$ de $J(g_1, ..., g_p) \cdot b_1, ..., J(g_1, ..., g_p) \cdot b_{r+1}$ tels que

$$\bigcup_{1 \leq j \leq r+1} \mathcal{V}(g_1, ..., g_p) \setminus \mathcal{V}(\Delta_j)$$

coïncide avec la partie lisse de $\mathcal{V}(g_1, ..., g_p)$. Prenons alors $H := \sum_{1 \leq j \leq r+1} \Delta_j^2$ et supposons que

$$\Gamma := \mathcal{V}(g_1, ..., g_p) \cap \mathbb{R}^n$$

soit non vide, lisse et compacte. En prenant $J(g_1, ..., g_p)$ pour F, nous avons que V est non vide, équidimensionnelle de dimension r, lisse et contient Γ. En utilisant [11, Proposition 1] ou [12, Proposition 1], si $a \in \mathbb{Q}^{r \times n}$ est suffisamment générique, alors $W(a_r)$ contient au moins un point par composante connexe de Γ. Le théorème précédent implique que l'on peut calculer au moins un point par composante connexe de Γ en temps $L\ (n\ d)^{O(1)}\ \delta^2 \subseteq (n\ d)^{O(n)}$ au moyen d'un algorithme probabiliste. Ce résultat améliore les bornes de complexité de [11, Theorem 11] et [12, Theorem 13] par un facteur essentiellement $\binom{n}{p}$.

Chapitre IV

Traitement numérique des grappes de racines

Dans les années quatre-vingt, M. Shub et S. Smale ont tissé des liens étonnants entre la théorie de la complexité et l'étude des systèmes dynamiques [28, Chapter 8]. Un de ces liens repose sur une théorie quantitative de l'opérateur de Newton, appliqué à des fonctions analytiques à plusieurs variables. Ces liens ont eu un impact fort sur l'algorithmique des variétés analytiques et algébriques. Ils ont aussi été la source de nombreux résultats théoriques et ont permis une meilleure compréhension des notions de robustesse, certification et complexité en analyse numérique [264, 265, 267, 268].

L'essentiel de l'attention de ces méthodes a d'abord porté sur les systèmes non dégénérés ne comprenant que de solutions simples. Les premiers travaux donnant des résultats quantitatifs quant à l'existence de zéros de multiplicité deux sont dus à J.-P. Dedieu et M. Shub. Motivé par les idées contenues dans [188], j'ai cherché à généraliser mon algorithme symbolique de déflation pour l'approximation des zéros isolés des applications analytiques à plusieurs variables ; l'un des objectifs à plus long terme étant la conception d'un algorithme de résolution unifié numérique et formel dans l'esprit de l'algorithme de résolution géométrique du chapitre précédent.

Les résultats présentés dans ce chapitre ont été obtenus en collaboration avec M. Giusti, B. Salvy et J.-C. Yakoubsohn. Dans [108] nous traitons le cas des fonctions analytiques à une variable : nous proposons des algorithmes certifiés pour la localisation de grappes de zéros et leur approximation. Notre algorithme d'approximation converge quadratiquement vers la grappe et s'arrête à une distance proportionnelle au diamètre de celle-ci. En particulier, à la fin de l'itération nous obtenons une estimation du diamètre à des constantes multiplicatives universelles près. Dans l'article suivant [109] nous généralisons ces résultats au cas des applications analytiques à plusieurs variables sous l'hypothèse que les grappes de zéros sont dans des ouverts connexes où l'application différentielle a un corang un. Ceci inclut les déformations de zéros multiples de corang un. Une refonte complète des résultats connus sur les zéros simples est présentée dans ce travail. Elle est basée sur l'utilisation systématique des techniques de séries majorantes. Plusieurs d'entre eux sont améliorés comme l'estimation d'une série majorante géométrique pour l'inversion locale.

IV.1 Cas régulier à plusieurs variables

L'algèbre des séries entières à rayon de convergence strictement positif est noté $\mathbb{R}\{t\}$. Nous munissons $\mathbb{R}\{t\}$ de l'ordre partiel $\leq$ défini comme suit : $\sum_i a_i\, t^i \leq \sum_i b_i\, t^i$ si $a_i \leq b_i$ pour tout $i \geq 0$. Nous munissons $\mathbb{C}^n$ de sa norme hermitienne canonique et prenons le supremum sur la boule unité comme norme pour les applications linéaires.

Dans la suite f représente une application analytique définie sur un ouvert U de $\mathbb{C}^n$ à valeurs dans $\mathbb{C}^n$ et a est un point de U tel que $D\,f(a)$ est inversible. Nous utiliserons les notations de S. Smale pour les quantités suivantes que nous appellerons *estimations ponctuelles* (*point estimates* en anglais), par opposition à des estimations que l'on peut obtenir par intégration ou *via* des modèles de Taylor [136, 203, 204]. La première quantité $\gamma(f;a)$ permet un contrôle local de f :

$$\gamma(f;a) := \sup_{k\geq 2} \left\| D\,f(a)^{-1}\, \frac{f^{(k)}(a)}{k!} \right\|^{\frac{1}{k-1}}.$$

En particulier, le rayon de convergence du développement en série entière de f en a est au moins $1/\gamma$. La deuxième quantité est la distance parcourue lors d'une itération de Newton : $\beta(f;\, a) := \|D\ f(a)^{-1}\ f(a)\|$. Enfin la troisième est le produit des deux

premières : $\alpha(f;a) := \beta(f;a)\,\gamma(f;a)$. D'autre part nous utiliserons la notation suivante :

$$|f|_a := \sum_{k\geq 0} \|D^k f(a)\| \frac{t^k}{k!} \in \mathbb{R}\{t\}.$$

Nous dirons qu'une série $F \in \mathbb{R}\{t\}$ est majorante pour f au point a si $|f|_a \leq F$. Enfin, l'opérateur de Newton est noté :

$$N(f;x) := x - D\,f(x)^{-1}\,f(x).$$

Il est bien connu que les séries majorantes ont de bonnes propriétés de transfert pour les opérations usuelles (multiplication, dérivation, etc). Contrairement à la présentation faite dans le livre [28], cette machinerie permet une présentation rapide et simple de plusieurs résultats regroupés sous la dénomination α-théorie (ou théorie des zéros approchés). Nous n'énoncerons pas ces résultats mais pointons sur une première généralisation obtenue en termes de séries majorantes :

Théorème IV.1. [109] *Soit $F \in \mathbb{R}\{t\}$ tel que $|D\,f(x_0)^{-1}\,f|_{x_0} \leq F$. On pose $\tilde{F} := F - (1 + F'(0))\,t$. Soit $r > 0$ un réel inférieur au rayon de convergence de F et tel que $\bar{B}(x_0, r) \subseteq U$ et $\tilde{F}(r) < 0$. Alors f admet exactement un zéro simple ζ dans $\bar{B}(x_0, r)$. La suite $(x_k)_{k\in\mathbb{N}}$, définie par la récurrence $x_{k+1} := N(f; x_k)$, et la suite $(t_k)_{k\in\mathbb{N}}$, définie par $t_0 = 0$ et $t_{k+1} := N(\tilde{F}; t_k)$, sont bien définies : x_k appartient à $\bar{B}(x_0, r)$, pour tout $k \geq 0$, converge quadratiquement vers ζ, t_k est croissante et converge quadratiquement vers la première racine positive r^- de $\tilde{F}$. De plus on a :*

a) $\|x_k - x_{k+1}\| \leq t_{k+1} - t_k$;
b) $\|D\,f(x_0)^{-1}\,f(x_k)\| \leq t_{k+1} - t_k$;
c) $\|x_k - \zeta\| \leq r^- - t_k$.

Ce résultat généralise au cas des séries majorantes celui de [292], qui donne la valeur optimale $\alpha_0 := 3 - 2\sqrt{2}$ pour l'énoncé suivant de S. Smale : si $\alpha(f; x_0) < \alpha_0$ alors f admet un zéro simple dans $\bar{B}(x_0, \left(1 + \sqrt{2}/2\right)\,\beta(f; x_0))$ et la suite des itérés de Newton de x_0 converge vers ce zéro.

Un autre résultat important est une version quantitative du théorème d'inversion locale :

Théorème IV.2. [109] *Soit f une application analytique d'un ouvert $U \subseteq \mathbb{C}^n$ vers $\mathbb{C}^n$. Soit $\zeta \in U$ tel que $D\,f(\zeta)$ est inversible et $\sigma \geq \|D\,f(\zeta)^{-1}\|$, $\gamma \geq \gamma(f; \zeta)$, $B_f := B\Big(\zeta, \frac{1 - \sqrt{2}/2}{\gamma}\Big)$. Si $B_f \subseteq U$ alors il existe une unique application g ayant les propriétés suivantes :*

a) *g est définie et analytique sur $B_g := B\Big(f(\zeta), \frac{1}{(3 + 2\sqrt{2})\,\sigma\,\gamma}\Big)$;*
b) $g(B_g) \subseteq B_f$;
c) $f \circ g(b) = b$, *pour tout $b \in B_g$;*
d) *Pour tout $b \in B_g$ il existe un unique $a \in B_f$ tel que $f(a) = b$. De plus on a $a = g(b)$ et $\|a - \zeta\| \leq (1 + \sqrt{2}/2)\,\beta\,(f - b; \zeta)$;*
e) $\|g - \zeta\|_{f(\zeta)} \leq \dfrac{\sigma\,t}{1 - (3 + 2\sqrt{2})\,\sigma\,\gamma\,t}$.

Les parties (a), (b), (c) et (d) de ce résultat figurent déjà dans le livre [28]. Le point (e) généralise et améliore une majoration due à [65]. Ces techniques calculatoires sont très mécaniques et ont des champs d'applications très vastes. Par exemple, les deux résultats précédents sont utilisés dans la thèse de G. Chèze pour certifier un algorithme numérique qui calcule la factorisation absolue (sur $\mathbb{C}$) d'un polynôme à deux variables à coefficients dans $\mathbb{Q}$ [47]. Le coût du contrôle d'erreur est très faible en terme de temps de calcul.

IV.2 Cas singulier à une variable

En présence de multiplicité le théorème IV.1 est loin d'être immédiat à généraliser. De plus, d'un point de vue pratique il est plus pertinent de considérer les *grappes de zéros* plutôt que les zéros multiples. Ces grappes apparaissent souvent comme la déformation d'un zéro multiple sous l'effet des arrondis. En théorie, une grappe est juste un synonyme d'*ensemble*. On utilise ce terme lorsque le diamètre de l'ensemble est petit devant la distance aux autres zéros. Bien entendu ceci est une question d'échelle et la principale difficulté est de concevoir un formalisme pour rendre précises et manipuler ces idées. Dans ce sens, nous introduisons les quantités suivantes :

$$\begin{aligned}\gamma_m(f;a) &:= \sup_{k\geq m+1}\left(\frac{m!\,\left|f^{(k)}(a)\right|}{k!\,\left|f^{(m)}(a)\right|}\right)^{\frac{1}{k-m}},\\ \beta_{m,l}(f;a) &:= \sup_{l\leq k\leq m-1}\left(\frac{m!\,\left|f^{(k)}(a)\right|}{k!\,\left|f^{(m)}(a)\right|}\right)^{\frac{1}{k-m}},\\ \alpha_{m,l}(f;a) &:= \gamma_m(f;a)\,\beta_{m,l}(f;a).\end{aligned}$$

Ces quantités sont dues à J.-C. Yakoubsohn [299] et correspondent aux estimations ponctuelles précédentes lorsque $m=1$ et $l=0$. Concernant la localisation des grappes nous avons montré :

Théorème IV.3. [108] *Soit f une fonction analytique définie sur un ouvert connexe $U\subseteq\mathbb{C}$, soit $m\geq 1$ un entier, $l\in\{0,\ldots,m-1\}$ et $z\in U$ tels que $f^{(m)}(z)\neq 0$,*

$$\frac{m-l}{m}\,\frac{m+1}{m+1-l}\,\alpha_{m,l}(f;z)\leq\frac{1}{9}$$

et $\bar{B}\Big(z,3\,\frac{m-l}{m}\,\beta_{m,l}(f;z)\Big)\subseteq U$. Alors $f^{(l)}$ admet $m-l$ zéros, en comptant les multiplicités, dans $\bar{B}\Big(z,3\,\frac{m-l}{m}\,\beta_{m,l}(f;z)\Big)$ et aussi dans $\bar{B}\Big(z,\frac{m+1-l}{3\,(m+1)\,\gamma_m(f;z)}\Big)\cap U$.

Ce résultat est en fait une conséquence d'un critère de Pellet [236] obtenu principalement *via* le théorème de Rouché en suivant des idées de [299, 300]. Notre critère est plus général et améliore ceux de [280, 299, 300]. Bien sûr il existe d'autres façons moins contraignantes pour trouver des grappes de racines en particulier par des calculs d'intégrales sur des chemins [179, 180, 181], mais ceux-ci sont nettement plus coûteux. Rappelons aussi que la présente approche ne concerne pas seulement les polynômes pour lesquels il existe des méthodes plus particulières mais que ne nous discuterons pas ici. Plusieurs éléments de comparaisons sont donnés dans notre article [108]. Ensuite nous avons montré une borne dans l'autre sens :

Théorème IV.4. [108] *Soient $m\geq 1$ un entier et $\zeta\in U$ tels que $f^{(m)}(\zeta)\neq 0$. Si $m\,\alpha_m(f;\zeta)<1/12$ et $\bar{B}(\zeta,3\,\beta_m(f;\zeta))\subseteq U$ alors f admet m zéros Z dans $\bar{B}(z,3\,\beta_m(f;z))$, en comptant les multiplicités. De plus, si ζ est dans l'enveloppe convexe de Z alors $\beta_m(f;\zeta)\leq 24\,m^2\,D$, où D représente le diamètre de Z.*

De façon informelle nous pouvons dire que β_m permet d'obtenir une bonne estimation du diamètre de la grappe.

Concernant l'algorithme d'approximation nous sommes partis de l'opérateur de Schröder :

$$N_m(f;x) := x - m\,\frac{f(x)}{f'(x)}.$$

Il est bien connu que la suite des itérés de Schröder converge quadratiquement en présence d'un zéro de multiplicité m [254]. En présence d'une grappe le comportement est chaotique : loin de la grappe la convergence est quadratique et la difficulté pratique réside dans la détection du moment où cette convergence cesse. Ce comportement se remarque bien avec l'exemple très simple $f(x)=x^2-\varepsilon^2$: si $|z|\gg|\varepsilon|$ alors $N_2(f;z)=\varepsilon^2/z$ se rapproche de la grappe $\{\varepsilon,-\varepsilon\}$, si $|z|\ll|\varepsilon|$ alors $N_2(f;z)$ s'éloigne de cette grappe, et si $|z|\approx|\varepsilon|$ alors $N_2(f;z)$ reste au voisinage de la grappe.

Nous proposons un algorithme pour stopper l'itération à une distance de la grappe qui est de l'ordre de son diamètre. Notre méthode est numériquement stable et ne demande pas de calculer des dérivés d'ordre plus grand que un. Nous la présentons d'une manière informelle dans les paragraphes suivants et renvoyons le lecteur vers [108] pour les détails techniques et les justifications.

Tout d'abord nous considérons des classes de fonctions analytiques pour lesquelles les quantités ponctuelles α_m, β_m et γ_m sont estimables, au sens ou il est possible d'en obtenir des approximations à une précision souhaitée. Soit f une telle fonction analytique et soit $z_0\in\mathbb{C}$, tels que $\alpha_m(f;z_0)<\hat{\alpha}_m$ pour une constante $\hat{\alpha}_m$ calculable. Nous pouvons calculer des quantités r et $\mathcal{G}$, puis nous définissons par récurrence une suite $(z_k)_{k\in\mathbb{N}}$ de la façon suivante. Une fois z_k calculé, nous distinguons différents cas :

1. Si $f'(z_k)=0$ alors on pose $z_j:=z_k$ pour tout $j\geqslant k+1$.
2. Sinon on calcule $y_k:=z_k-m\,f(z_k)/f'(z_k)$. Si $|y_k-z_k|>r$ alors on pose $z_j:=z_k$ pour tout $j\geqslant k+1$.
3. Sinon, si $\beta_m(f;y_k)\geqslant\mathcal{G}\,\|y_k-z_k\|^2$ alors
   a) si $\beta_m(f;z_k)<\beta_m(f;y_k)$ alors on pose $z_j:=z_k$ pour tout $j\geqslant k+1$,
   b) sinon on pose $z_j:=y_k$ pour tout $j\geqslant k+1$.
4. Sinon on pose $z_{k+1}:=y_k$.

Nous obtenons de cette façon une suite $(z_k)_k$ qui converge quadratiquement vers une grappe de m racines Z. En notant ζ la limite nous avons montré que la distance de ζ à Z, notée $\mathrm{dist}(\zeta,Z)$, satisfait $\varsigma_0\,\mathrm{diam}(Z)\leqslant\mathrm{dist}(\zeta,Z)\leqslant\varsigma_1\,\mathrm{diam}(Z)$ pour des constantes universelles ς_0 et ς_1, et où $\mathrm{diam}(Z)$ représente le diamètre de Z.

Dans [108] nous illustrons ces résultats avec une implantation que nous avons réalisée dans le système de calcul formel Maple et avons constaté un très bon comportement en pratique. Par exemple, avec $f(x):=(x^m+10^{-mN})\,(x^m-1)$, nous avons une grappe de m racines au voisinage de 0. Les autres racines sont simples sur le cercle unité. Dans le tableau suivant nous illustrons le comportement de la suite $(z_k)_k$ précédemment définie avec $m=4$, en fonction de N. Plus précisément nous donnons la valeur K à partir de laquelle la suite se stabilise à partir de $K+1$, grisons la case de la valeur limite retenue par l'algorithme et donnons les approximations des valeurs β_m correspondantes, qui sont bien du même ordre de grandeur que le diamètre de la grappe.

N	$\lvert z_0\rvert$	K	$\lvert z_K\rvert$	$\lvert y_{K+1}\rvert$	$\beta_m(f;z_K)$	$\beta_m(f;y_{K+1})$
4	$4,88\,10^{-4}$	0	$4,88\,10^{-4}$	$8,58\,10^{-7}$	$1,95\,10^{-3}$	$9,99\,10^{-5}$
8	$4,88\,10^{-4}$	1	$2,77\,10^{-17}$	$4,67\,10^{17}$	$1,00\,10^{-8}$	∞
16	$4,88\,10^{-4}$	1	$2,77\,10^{-17}$	$4,67\,10^{-15}$	$1,11\,10^{-16}$	$1,87\,10^{-14}$
32	$4,88\,10^{-4}$	1	$2,77\,10^{-17}$	$4,67\,10^{-79}$	$1,11\,10^{-16}$	$1,00\,10^{-32}$
64	$4,88\,10^{-4}$	2	$1,64\,10^{-83}$	$2,23\,10^{-8}$	$9,99\,10^{-65}$	$8,94\,10^{-8}$
128	$4,88\,10^{-4}$	2	$1,64\,10^{-83}$	$2,23\,10^{-264}$	$6,58\,10^{-83}$	$1,00\,10^{-128}$

Tableau IV.1. Approximation numérique d'une grappe de quatre racines.

Il s'agit à mon sens d'une avancée majeure en algorithmique numérique : à ma connaissance aucune autre méthode avec des propriétés similaires n'était connue, seules des heuristiques sont utilisées dans les logiciels pour limiter les effets chaotiques produits à l'approche d'une grappe. Les principaux travaux réalisés sur ce sujet sont [59, 60, 62, 63, 114, 115, 116, 231, 240, 241, 242, 254, 286, 299, 303] (certains d'entre-eux concernent aussi le cas à plusieurs variables).

IV.3 Cas singulier à plusieurs variables

Nous avons généralisé les résultats précédents à plusieurs variables pour localiser et approcher les grappes de zéros obtenues par déformation des zéros multiples d'applications analytiques de corang 1. L'archétype immédiat est

$$(x_1, ..., x_{n-1}, y) \mapsto (x_1, ..., x_{n-1}, y^m),$$

qui admet l'origine comme zéro de multiplicité m de corang 1.

Plus précisément nous définissons une grappe de *dimension de plongement* 1 de la façon suivante : soit F une application analytique définie sur un ouvert connexe maximal $U \subseteq \mathbb{C}^n$ à valeurs dans $\mathbb{C}^n$. Un ensemble de zéros de F est appelé une *grappe de zéros de dimension de plongement* 1 s'il existe deux sous-espaces vectoriels S et T de $\mathbb{C}^n$ de codimension 1 et une boule $B(\zeta, r) \subseteq U$ contenant la grappe et tels que $\mathrm{Pr}_T \circ D\,F(x)$ est inversible sur S, pour tout $x \in B(\zeta, r)$, où Pr_T représente la projection orthogonale sur T.

Dans la suite nous considérons que F est donnée par (f, g) avec $f: U \to \mathbb{C}^{n-1}$ et g: $U \to \mathbb{C}$ deux applications analytiques. Pour tout $(x_0, y_0) \in U$ tel que $D_x f(x_0, y_0)$ est inversible, nous posons

$$\begin{array}{rccl} \Sigma(x, y): & U & \to & \mathbb{C}^n \\ & (x, y) & \mapsto & (D_x f(x_0, y_0)^{-1} f(x, y), y), \end{array}$$

qui est localement inversible au voisinage de (x_0, y_0). Son inverse est notée $\Phi(z, y)$ et est analytique au voisinage de (z_0, y_0), avec $z_0 := D_x f(x_0, y_0)^{-1} f(x_0, y_0)$. Nous posons aussi

$$h(z, y) := g(\Phi(z, y)).$$

Dans un certain voisinage de (x_0, y_0) le système $f(x, y) = g(x, y) = 0$ est équivalent à

$$h(0, y) = 0, \quad x = \Phi(0, y),$$

nous sommes ainsi ramenés à une situation à une variable complexe.

Pour tout $l \geq 0$, nous introduisons la l-ième *application déflatée* (*deflated map* en anglais) $(f, g^{[l]})$ définie comme suit:

$$g^{[0]} = g, \quad g^{[l+1]} = \frac{\det\left(D(f, g^{[l]})\right)}{\det\left(D_x f\right)}, \quad \text{pour } l \geq 0,$$

où det représente l'application déterminant, la base implicitement considérée étant la base canonique donnée par (x, y). Si ζ est un zéro de dimension de plongement 1 et de multiplicité m de $f = g = 0$ alors ζ est un zéro de dimension de plongement 1 et de multiplicité $m - l$ de $f = g^{[l]} = 0$, pour tout $0 \leq l \leq m - 1$.

Théorème IV.5. [109] *Soit $m \geq 1$, $l \in \{0, ..., m-1\}$, κ le premier entier tel que $2^\kappa \geq m - l$, $x_0' := N^\kappa(f(., y_0); x_0)$ et $z_0' := D_x f(x_0, y_0)^{-1} f(x_0', y_0)$. Soient λ_g, ρ_g, β_x, γ_x, σ_x, $\beta_{m,l}$, γ_m, σ_m des nombres réels donnés. On introduit les quantités suivantes :*

$\lambda_\Phi := \sigma_x$, $\rho_\Phi := (3 + 2\sqrt{2})\,\sigma_x\,\gamma_x$,

$\lambda := \lambda_g\,\lambda_\Phi$, $\rho := \rho_\Phi + \lambda_\Phi\,\rho_g$,

$$\bar{\lambda} := \frac{\lambda}{(1 - \rho\,(\beta_x + \|z_0'\|))^2},\ \bar{\rho} := \frac{\rho}{1 - \rho\,(\beta_x + \|z_0'\|)},$$

$$\bar{\mu} := \frac{\bar{\rho}}{1 - \bar{\rho}\,\|z_0'\|},$$

$$\bar{e} := ((m+1)\,\sigma_m\,\bar{\lambda}\,||z_0'|/(1 - \bar{\rho}\,\|z_0'\|))^{1/m},$$

$$\bar\beta_{m,l} := \frac{\beta_{m,l} + \frac{m}{m+1}\,\bar\mu^{\frac{l}{m-l}}\,\bar e^{\frac{m}{m-l}}}{1-(\bar\mu\,\bar e)^m},\ \bar\gamma_m := \frac{\gamma_m + \frac{m+2}{m+1}\,\bar\mu}{1-(\bar\mu\,\bar e)^m},$$
$$r_y^- := 3\,\frac{m-l}{m}\,\bar\beta_{m,l},\ r_y^+ := \frac{m+1-l}{3\,(m+1)\,\bar\gamma_m},$$
$$r_x^- := \frac{\lambda_\Phi\,(\beta_x + r_y^-)}{1-\rho_\Phi\,(\beta_x+r_y^-)}.$$

Si

$$|g - g(x_0,y_0)|_{(x_0,y_0)} \le \frac{\lambda_g\, t}{1-\rho_g\, t},$$
$$\sigma_x \ge \|D\,\Sigma(x_0,y_0)^{-1}\|,$$
$$\beta_x \ge \beta\,(\Sigma - (0,y_0); x_0, y_0),\ \gamma_x \ge \gamma(\Sigma; x_0, y_0),$$
$$\rho\,(\beta_x + \|z_0'\|) < 1,$$
$$|D_y^m\, h(z_0', y_0)| \ne 0,\ \sigma_m \ge m!/|D_y^m\, h(z_0', y_0)|,$$
$$\beta_{m,l} \ge \beta_{m,l}(h(z_0', .); y_0),\ \gamma_m \ge \gamma_m(h(z_0', .); y_0),$$
$$\bar\rho\,\bar z_0' < 1,\ \bar\mu\,\bar e < 1,$$
$$\frac{m-l}{m}\,\frac{m+1}{m+1-l}\,\bar\beta_{m,l}\,\bar\gamma_m \le 1/9,$$

alors $(f, g^{[l]})$ *admet* $m-l$ *zéros, en comptant les multiplicités, dans*

$$B_\Sigma \cap B_g \cap (\bar B(x_0, r_x^-) \times \bar B(y_0, r_y^-))$$

mais aussi dans

$$B_\Sigma \cap B_g \cap (\mathbb{C}^{n-1} \times \bar B(y_0, r_y^+)),$$

où

$$B_\Sigma := B\left((x_0,y_0), \frac{1-\sqrt{2}/2}{\gamma_x}\right), B_g := B\left((x_0,y_0), \frac{1}{\rho_g}\right).$$

Comme dans le cas à une variable nous proposons un algorithme d'approximation. Dans la suite (x_0, y_0) représente le point initial. Pour tout $l \le m-1$ et $l' \le l$ nous décrivons un algorithme permettant d'approcher une grappe de zéros de $(f,\ g^{[l']})$ en utilisant l'opérateur Schröder sur $D_y^l h$.

En pratique il n'est pas toujours possible de calculer les quantités $\beta_{m,l'}$ il nous faut prendre en compte d'éventuelles erreurs d'approximation : pour cela nous paramétrons l'algorithme par une fonction $\mathcal{B}_{m,l'}(h(z_k', .),\, y_k;\, y_k')$ qui retourne une approximation de $\beta_{m,l'}(h(z_k', .);\, y_k')$ à partir d'estimations ponctuelles en y_k seulement. L'algorithme est aussi paramétré par trois réels positifs r_y, $\mathcal{G}_y$ et $\mathcal{G}_z$.

L'étape k de l'algorithme d'approximation consiste à calculer $(x_{k+1},\, y_{k+1})$ à partir de $(x_k,\, y_k)$. Grossièrement nous procédons comme suit : nous utilisons l'opérateur de Newton sur $f(.,\, y_k)$ pour calculer une valeur x_k' à partir de x_k. Ensuite nous calculons z_k' comme les $n-1$ premières coordonnées de $\Sigma(x_k', y_k)$ puis l'opérateur de Schröder est utilisé avec $h(z_k, .)$ et y_k pour obtenir y_k'. Puis, il s'agit de déterminer lequel de y_k et y_k' donnera y_{k+1}. À la fin, x_{k+1} est obtenu en appliquant l'opérateur de Newton sur $f(.,\, y_{k+1})$ depuis x_k. De façon plus précise, notre algorithme fonctionne comme suit, où κ représente le plus petit entier tel que $2^\kappa \ge 2\,(m-l')$:

1. $x_k' := N^\kappa(f(., y_k); x_k)$;
2. $z_k' := D_x\, f(x_0,y_0)^{-1}\, f(x_k', y_k)$;
3. Si $D_y^{l+1}\, h(z_k', y_k) = 0$
4. alors
5. $\quad y_{k+1} := y_k$;
6. sinon
7. $\quad y_k' := N_{m-l}\,(D_y^l\, h(z_k', .); y_k)$;
8. $\quad$ if $y_k' \notin \bar B(y_k, 2\,r_y)$

9. alors
10. $y_{k+1} := y_k$;
11. sinon
12. si $\mathcal{B}_{m,l'}(h(z'_k, .), y_k; y'_k) >_y |y_k - y'_k|^2$
13. alors
14. si $\mathcal{B}_{m,l'}(h(z'_k, .), y_k; y'_k) < \beta_{m,l'}(h(z'_k, .); y_k)$
15. alors
16. $y_{k+1} := y'_k$;
17. sinon
18. $y_{k+1} := y_k$;
19. alors
20. $y_{k+1} := y'_k$;
21. $x_{k+1} := N(f(., y_{k+1}); x_k)$;

Nous pouvons préciser informellement l'idée de l'analyse de convergence. Nous notons ζ un point de la grappe de zéros de $(f, g^{[l']})$. Par construction de x'_k nous montrons que $\|z'_k\|^{1/(m-l')}$ est du second ordre, i.e. dans $O(\|(x_k, y_k) - \zeta\|^2)$. Ensuite nous appliquons nos résultats précédents à une variable avec $D^l_y h(z'_k, .)$ et y_k. Les différents cas s'interprètent de la façon suivante : l'étape (10) correspond au cas où y'_k s'éloigne fortement de la grappe de $D^l_y h(z'_k, .)$, ce qui implique que y_k en est déjà très proche. L'étape (20) correspond au cas où y'_k est une meilleure approximation de la grappe au second ordre. Entre ces deux cas, le test de l'étape (14) détermine lequel entre y_k et y'_k est le plus proche de la grappe. À la fin, la correction entre y_{k+1} et y_k est simplement propagée dans les coordonnées en x à l'étape (21).

L'énoncé complet du résultat avec toutes les formules pour calculer les bonnes valeurs des paramètres est trop long pour être écrit ici. D'une façon informelle nous pouvons dire qu'il existe un algorithme permettant de décider si la suite des itérés d'un point (x_0, y_0) via l'opérateur ci-dessus converge quadratiquement vers une grappe de zéros. En présence d'une grappe de zéros de diamètre strictement positif, l'itération peut-être arrêtée à une distance de celle-ci qui est de l'ordre de son diamètre.

J'ai programmé cet algorithme en Maple. Son comportement est bien conforme à notre attente et tous les cas apparaissent bien en pratique. À l'heure actuelle nous ne voyons pas comment généraliser ces résultats pour des grappes de racines quelconques en conservant une complexité polynomiale en la multiplicité.

IV.4 Remarques

Pour conclure ce chapitre nous mentionnons quelques autres techniques. La littérature sur le traitement des multiplicités et grappes de racines et très variées. Nous nous restreignons ici aux travaux les plus proches des nôtres et renvoyons le lecteur à [108, 109] pour d'avantage de références.

IV.4.1 Localisation et approximation de grappes de racines

L'analyse précise du comportement de la méthode de Newton en présence de singularité est un problème épineux à plusieurs variables. Motivé par les travaux d'A. M. Ostrowski [231], L. B. Rall a commencé à obtenir des résultats dans des cas particuliers [240]. Ensuite, et sur la base de travaux de G. R. Reddien [241, 242], D. W. Decker et C. T. Kelley [60, 61] ont pu préciser les convergences pour des problèmes singuliers de premier et second ordres. A. Griewank et M. R. Osborne [114, 115, 116] ont ensuite étendu ces résultats en précisant les domaines de convergence. Signalons que le cas à une variable, qui est plutôt bien compris [284, 285], a pu être étendu à plusieurs variables pour des systèmes de Pham [178].

Corriger l'itération de Newton pour rétablir une convergence quadratique a fait l'objet de nombreux travaux. Mentionnons une première famille de techniques dites par *régularisation*, pour lesquelles nous renvoyons le lecteur à [5, Theorem 1]. Ensuite les techniques dites de *systèmes bordant* consistent à construire de nouveau systèmes n'ayant que des racines simples. Elles ont été développées pour les racines doubles [115, 149, 150, 211, 293, 301] mais aussi en corang un [283]. La construction de *systèmes augmentés* est une alternative qui introduit de nouvelles variables pour les coordonnées de vecteurs dans le noyau de jacobiennes bien choisies [111, 117, 118, 130, 185, 239].

Enfin les méthodes par *déflations* consistent à ajouter au système de nouvelles équations obtenues par dérivations des équations initiales. Elles ont d'abord été étudiées d'un point de vue mixte symbolique et numérique [229, 230]. Dans [199], cette technique est améliorée et le nombre d'étapes de déflation est prouvé borné par la multiplicité (néanmoins le coût total ne l'est pas). La difficulté à trouver un bon compromis entre efficacité et stabilité numérique a motivé d'autres approches [200, 201, 202, 298]. Les questions de complexité et stabilité de ces méthodes mixtes ne sont pas bien connues. La situation est un peu plus simple dans un cadre non-archimédien (pour les séries ou nombres p-adiques), et mon algorithme [188] a une convergence quadratique et un coût qui reste polynomial dans la multiplicité, sans pour autant calculer l'algèbre locale. L'idée des résultats de ce chapitre est d'avantage d'obtenir une bonne complexité et de pouvoir traiter les grappes de racines que d'optimiser le conditionnement de la méthode qui est sans doute perfectible.

Dans les grandes lignes on peut observer plusieurs similarités dans toutes ces méthodes. D'un point de vue algébrique la connaissance de l'algèbre locale d'une racine multiple permet de construire des opérateurs de type Newton avec convergence quadratique. Néanmoins, avec les techniques connues, cette algèbre locale ne s'obtient en général pas en temps polynomial dans la multiplicité. Par conséquent la quantité d'information algébrique minimale pour restaurer la convergence quadratique dans de bonnes conditions numériques ne semble pas connue. Par ailleurs, le traitement des grappes de racines pour la plupart des algorithmes mentionnés ici a peu été étudié [175].

IV.4.2 Point de vue global

Les méthodes précédentes sont en général utilisées au sein d'algorithmes complets pour résoudre par exemple numériquement ou symboliquement des systèmes algébriques. Lorsque la dimension réelle de l'espace ambient est petite, disons deux ou trois, les méthodes par subdivisions sont déjà efficaces en pratique et les méthodes de localisation et approximation de grappes sont très utiles pour accélérer ces algorithmes [232, 243].

Un autre cadre d'application important concerne les méthodes de résolution des systèmes algébriques par déformation homotopique. On commence par un construire système d'une certaine nature similaire, de sorte à pourvoir calculer ses racines soit par récurrence, soit par une formule explicite, en fonction de la géométrie considérée. Ensuite on suit des chemins dans l'espace des systèmes et de leurs solutions. Les chemins sont normalement des courbes algébriques de multiplicité un et la méthode de Newton peut donc être utilisée. Si le système à résoudre possède des multiplicités alors celles-ci sont observées à l'extrémité terminale des chemins et des techniques spécifiques peuvent être utilisées. Lorsque les systèmes ne sont pas de dimension zéro, des algorithmes de décomposition équidimensionelle similaires au mien [189] ont été adaptés d'un point de vue numérique. Il font intervenir des suivis de chemin avec multiplicité. L'homotopie est très efficace en pratique pour des systèmes bien conditionnés. Nous renvoyons le lecteur par exemple à [14, 15, 198, 221, 269, 270, 273].

Chapitre V

Mathemagix

Mathemagix est un ensemble de logiciels, diffusé *via* http://www.mathemagix.org, initié et coordonné par J. van der Hoeven depuis les années quatre-vingt dix, et visant la conception d'un langage de programmation scientifique adapté au calcul symbolique et analytique robuste. Ses composantes sont des bibliothèques efficaces de calcul écrites en C++, un compilateur et un interprète de commandes du langage de programmation appelé aussi Mathemagix, et enfin de liens entre ces bibliothèques et ce langage. Une interface de qualité avec le logiciel GNU $\mathrm{TEX}_{\mathrm{MACS}}$ est disponible pour l'interprète. Le code source est ouvert et distribué essentiellement sous license GNU GPL depuis le site Internet https://gforge.inria.fr/projects/mmx.

Le projet Mathemagix regroupe actuellement une trentaine de développeurs et le nombre effectif de lignes de code C++ est d'environ 240 000, auxquelles viennent s'ajouter environ 150 000 lignes pour une version C++ nommée linalg de la bibliothèque Lapack (http://www.netlib.org/lapack), développée par Ph. Trébuchet, et permettant d'utiliser les fonctionnalités de cette dernière avec des types numériques définis par l'utilisateur – par exemple en précision arbitraire.

Depuis 2005, je participe activement au développement de Mathemagix principalement en collaboration avec J. van der Hoeven. L'essentiel de mes algorithmes récents y sont disponibles, ainsi que ceux des étudiants que j'ai coencadrés en thèse. D'une façon générale, je pense nécessaire que les algorithmes publiés soient programmés et diffusés d'une façon à pouvoir être testés et comparés aisément.

V.1 Bibliothèques de calcul

Plusieurs choix de conception ont conduit Mathemagix a être un projet de nature plutôt autonome en terme de ses dépendances sur d'autres logiciels. Nous fournissons en effet des bibliothèques C++ performantes pour les objets algorithmiques élémentaires, l'arithmétique de base sur les polynômes et séries à une ou plusieurs variables, ainsi que les opérations de base en algèbre linéaire. Du code optimisé est disponible pour différents anneaux de coefficients : entiers, rationnels, entiers modulaires, types numériques, nombres complexes, intervalles, boules, et coefficients génériques. Cette approche nous a certes imposé de réécrire beaucoup d'algorithmes classiques, mais elle nous a surtout permis d'implanter nos propres algorithmes avec une grande généricité. Au final, Mathemagix nécessite peu de temps de maintenance inhérent aux dépendances, et surtout propose un cadre uniforme pour la totalité de ses structures de données.

V.1.1 fonctionnalités

Les bibliothèques C++ de calcul au cœur de Mathemagix, qui sont développées principalement par J. van der Hoeven et moi-même, sont les suivantes :

basix. Cette bibliothèque contient tous les types de données classiques tels que les vecteurs, listes, tables de hachage, mais aussi les flux d'entrée et sortie, les sockets, un système de gestion de mémoire par compteurs de références, et des fonctions pour distribuer des calculs sur plusieurs processeurs au sein d'un même ordinateur. Notons que ces implantations ne reposent pas sur la bibliothèque standard de C++.

numerix. Les types numériques élémentaires, tels que les nombres entiers et à virgule flottante, intervalles, boules, nombres complexes et nombres modulaires sont disponibles dans cette bibliothèque, qui utilise GMP [112] et GNU MPFR [76].

algebramix. Cette bibliothèque contient les polynômes et séries à une variable, les matrices et les entiers p-adiques. Essentiellement toutes les opérations élémentaires sont disponibles de façons génériques et particulières, en intégrant plusieurs variantes d'algorithmes utilisant les ressources vectorielles de calcul des processeurs et prenant en compte l'organisation de la mémoire vive par niveaux de cache. Pour les polynômes sont disponibles : FFT, TFT, produits de A. Schönhage et V. Strassen, produit rapide générique de D. G. Cantor et E. Kaltofen [42], évaluation, interpolation, restes chinois et sous-résultants.

analyziz. Les algorithmes sur les polynômes, séries et matrices assurant une bonne stabilité pour les calculs numériques sont regroupés dans cette bibliothèque.

multimix. Les polynômes, jets, et séries à plusieurs variables sont disponibles dans cette bibliothèque, qui propose les algorithmes classiques en représentations dense, creuse et fonctionnelle, ainsi que nos techniques récentes [143, 144].

Sur cette base, voici les autres bibliothèques spécialisées développées au sein du projet :

lattiz par G. Lecerf. Réduction de réseau (algorithme LLL). Exploite FPLLL [275] si disponible, et utilise une version intégrale classique sinon.

finitefieldz par G. Lecerf, G. Quintin. Corps finis. Utilise MPFQ [95] si disponible.

factorix par G. Lecerf. Factorisation des polynômes.

symbolix par J. van der Hoeven. Calculs avec des expressions symboliques.

asymptotix par J. van der Hoeven. Développements asymptotiques et transséries.

holonomix par J. van der Hoeven. Fonctions holonômes.

geomsolvex par G. Lecerf. Résolution géométrique des systèmes algébriques.

continewz par J. van der Hoeven. Résolution numériques des systèmes algébriques par homotopie.

graphix par J. van der Hoeven. Tracés 2D et 3D.

linalg par Ph. Trébuchet. Version C++ de Lapack.

borderbasix par B. Mourrain, Ph. Trébuchet. Résolution des systèmes algébriques par bases bordantes.

realroot par B. Mourrain, J.-P. Pavone, E. Tsigaridas, A. Mantzaflaris. Structures algébriques et numériques pour la résolution réelle des systèmes algébriques, incluant des méthodes très efficaces par subdivision.

shape par B. Mourrain, A. Mantzaflaris. Structures de données et algorithmes de tracés pour les courbes et surfaces. Interface avec le logiciel de visualisation Axel [223].

columbus par J. van der Hoeven. Représentation graphique et exploration de fonctions analytiques.

D'autres bibliothèques plus expérimentales permettent de diffuser des implantations à usage plus restreint en relation avec nos articles de recherche : gregorix, jorix, quintix.

V.1.2 Compilation et installation

Au total Mathemagix se compose actuellement d'une trentaine de sous-projets pouvant être installés et utilisés séparément (en respectant néanmoins les dépendances). Chaque sous-projet dispose d'outils de compilation automatique produits par GNU Autotools (`http://www.gnu.org/software/autoconf`). Les bibliothèques sont structurées de façon relativement identiques et les fichiers d'entrée d'Autotools sont maintenus automatiquement par un logiciel, appelé automagix, écrit dans le langage Mathe-

MAGIX. Je participe à la réalisation de ces outils permettant de compiler et installer MATHEMAGIX sur les systèmes d'exploitation LINUX, MAC OS X et WINDOWS. J'ai réalisé le portage sous WINDOWS avec D. RAUX en utilisant les logiciels disponibles *via* MINGW (http://www.mingw.org). Mentionnons que la plupart des sous-projets peuvent aussi être compilés avec CMAKE (http://www.cmake.org) et que les fichiers correspondant de description sont produits par un logiciel similaire à AUTOMAGIX appelé MMXTOOLS. Cette alternative est développée par B. MOURRAIN.

V.1.3 EXEMPLES

Les exemples suivants montrent les différentes facettes du travail de programmation auquel j'ai participé pour les bibliothèques C++. Les temps sont mesurés sur mon ordinateur portable (MACBOOK PRO™, processeur INTEL CORE I7™ à 2,6 GHz), en utilisant le compilateur i686-apple-darwin11-llvm-g++-4.2 (GCC) 4.2.1 (Based on Apple Inc. build 5658, LLVM build 2336.11.00).

V.1.3.1 MODULARITÉ

L'une des fonctionnalités de nos bibliothèques C++ est de permettre aux utilisateurs de pouvoir redéfinir les sous-algorithmes utilisés dans des calculs *a posteriori*. La plupart des types de données paramétrés permettent en effet de décrire les structures de base et algorithmes sur lesquels ils s'appuient. Par exemple le type pour les polynômes à une variable d'ALGEBRAMIX est

```
template<typename C, typename V> class polynomial;
```

Ici `C` désigne le type des coefficients et `V` la variante d'implantation, c'est-à-dire un type représentant les algorithmes à utiliser pour calculer avec ces polynômes. Par exemple si `C` est un corps, `polynomial<C,polynomial_naive>` utilise les algorithmes naïfs pour la multiplication, division, p.g.c.d., etc. La représentation des polynômes est indépendante des variantes d'algorithmes. Il s'agit essentiellement du vecteur des coefficients sous la forme d'un pointeur `C*` sur le premier coefficient et de la taille du polynôme. L'opération de produit est implantée comme suit :

```
template<typename C, typename V>
polynomial<C,V> operator * (const polynomial<C,V>& P1, const polynomial<C,V>& P2) {
  typedef implementation<polynomial_multiply,V> Pol;
  nat n1= N(P1), n2= N(P2);
  if (n1 == 0 || n2 == 0) return Polynomial (CF(P1));
  nat l= aligned_size<C,V> (n1+n2-1);
  C* r= mmx_formatted_new<C> (l, CF(P1));
  Pol::mul (r, seg (P1), seg (P2), n1, n2);
  return Polynomial (r, n1+n2-1, l, CF(P1));
}
```

Sans entrer dans les détails, nous voyons que le calcul du produit est effectué sur les représentations par la fonction `mul` appartenant à la structure `Pol` qui ressemble à ceci lorsque `V` est `polynomial_naive` :

```
template<typename V>
struct implementation<polynomial_multiply, V, polynomial_naive> {
  template<typename C> static inline void
  mul (C* dest, const C* s1, const C* s2, nat n1, nat n2) { ... }
};
```

Par conséquent, pour utiliser sa propre fonction de produit, l'utilisateur peut simplement définir une nouvelle variante, notée par exemple `my_variant`, selon le procédé suivant :

```
DEFINE_VARIANT(my_variant,
               polynomial_dicho<polynomial_naive> >);

template<typename V>
struct implementation<polynomial_multiply,V,my_variant> {
  template<typename C> static inline void
  mul (C* dest, const C* s1, const C* s2, nat n1, nat n2) { ... }
};
```

Ensuite, le type `polynomial<C,my_variant>` permet d'utiliser systématiquement le produit spécifique, lors d'appels directs ou bien *via* les autres opérations (division, p.g.c.d., etc) définies en amont car provenant d'autres implantations associées dans le cas présent à la variante `polynomial_dicho<polynomial_naive> >`. Ce mécanisme est bien sûr utilisé dans algebramix pour les polynômes, séries et matrices à coefficients dans $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\,\mathbb{Z}$, etc. Il permet aussi d'importer rapidement des fonctionnalités provenant de bibliothèques externes spécialisées pour certains types de coefficients, comme dans le cas de la bibliothèque MGF2X pour les polynômes de $\mathbb{Z}/2\,\mathbb{Z}[x]$ qui importe le produit très efficace de GF2X [94].

Finalement, dans ce cadre, le fait d'exprimer le coût des algorithmes en fonction de celui de tel ou tel produit n'est pas une simple vue de l'esprit.

V.1.3.2 Efficacité de la FFT

La transformée de Fourier rapide (d'acronyme *FFT* en anglais) est une opération critique pour de nombreux algorithmes. Dans le tableau V.1 ci-dessous, nous comparons les temps obtenus en utilisant une seule file d'exécution (*thread* en anglais) avec MATHEMAGIX (variante de l'algorithme appelée *split radix* en anglais) d'une part et la bibliothèque FFTW3 [77, 158] d'autre part pour les nombres complexes construits sur le type C++ `double`. Bien que nos temps de calcul ne rivalisent pas avec ceux de cette bibliothèque très optimisée, il n'en sont pas pour autant éloignés. Pour les petites tailles, FFTW3 utilise directement des arbres de calcul ad hoc sans boucle ni branchement, ce qui n'est pas le cas de notre implantation.

taille	128	256	512	1024	2048	4096	8192	16384
FFTW3	0,33	0,82	1,9	4,4	10	24	58	123
ALGEBRAMIX	0,42	0,94	2,1	4,7	11	27	61	141

Tableau V.1. Temps de calcul de la FFT en microsecondes.

Dans le tableau V.2 nous montrons les temps nécessaires pour multiplier deux polynômes de $\mathbb{Z}/p\,\mathbb{Z}$, avec $p = 469762049$, en utilisant directement la FFT d'une part et la substitution de Kronecker d'autre part. Cette dernière se réduit au produit de deux entiers en utilisant la bibliothèque GMP version 5.0.5 [91, Chapter 8].

taille	128	256	512	1024	2048	4096	8192
FFT	0,0067	0,014	0,025	0,052	0,11	0,22	0,46
Kronecker	0,021	0,050	0,12	0,28	0,69	1,7	3,4

Tableau V.2. Produit dans $\mathbb{Z}/469762049\,\mathbb{Z}[x]$, en millisecondes.

V.1.3.3 Produit matriciel

Le produit de deux matrices est une opération difficile à implanter efficacement. Les performances des bibliothèques spécialisées commerciales ou libres sont déjà très élevées, et dans une moindre mesure nous montrons dans le tableau V.3 que les performances obtenues avec nos propres implantations génériques écrites en C++ sont raisonnables et ont l'avantage de part leur généricité d'offrir des performances élevées sur des types de coefficients (tel que `short int` dans notre exemple) qui ne sont pas supportés directement

par d'autres bibliothèques qui privilégient les types numériques. Pour ces comparaisons nous avons utilisé Atlas 1.10.0 et Eigen 3.1.2 installés *via* MacPorts et n'utilisons qu'une seule file d'exécution.

n	128	256	512	1024	2048	4096	8192
Atlas, `double`	0,36	2,7	20	165	1320	10628	86120
Eigen3, `double`	0,38	2,9	23	180	1420	11371	90121
algebramix, `double`	0,47	3,8	32	255	1893	13460	96839
Eigen3, `int`	0,19	1,4	11	89	710	5475	43573
algebramix, `int`	0,23	1,8	15	125	893	6486	46984
Eigen3, `short int`	0,88	6,7	52	417	3330	26566	214029
algebramix, `short int`	0,12	0,93	7,7	63	473	3506	24644

Tableau V.3. Produit de deux matrices de tailles $n \times n$, en millisecondes.

Dans le tableau V.4 nous nous intéressons au temps du produit de matrices de tailles $n \times n$ dont les coefficients sont des polynômes de $\mathbb{Z}/p\,\mathbb{Z}$, avec $p = 469762049$, de degré d. La ligne intitulée Kronecker effectue le produit naïf des matrices en utilisant la substitution de Kronecker pour multiplier les polynômes. La ligne FFT commence par calculer les transformées de Fourier des matrices, effectue ensuite les produits de toutes les matrices à coefficient dans $\mathbb{Z}/p\,\mathbb{Z}$, et calcule la transformée de Fourier inverse. La méthode par FFT est efficace dès les petites tailles.

n	8	16	32	64	128
FFT, $d = 15$	0,14	0,5	2,2	12	84
Kronecker, $d = 15$	0,49	4,0	31	254	2103

Tableau V.4. Produit de deux matrices polynomiales de tailles $n \times n$, en millisecondes.

V.1.3.4 Produit de polynômes à plusieurs variables

multimix a représenté un travail important sur les plans théoriques et pratiques. Il n'existe pas à notre connaissance de bibliothèque équivalente libre en C++ pour les polynômes, jets et séries à plusieurs variables. Nous comparons ici les performances de nos implantations avec Maple 16 [97] intégrant les travaux récents décrits dans [215, 216, 217]. Néanmoins à l'heure actuelle multimix ne permet pas encore de réaliser des calculs sur plusieurs files d'exécution. Les polynômes sont en représentation creuse de taille s composés de monômes pris au hasard de degrés partiels au plus 10000 et dont les coefficients sont des entiers signés pris aléatoirement sur 31 bits. Dans le cas présent Mathemagix utilise une représentation compacte des monômes, l'arithmétique sur les entiers 128 bits fournie par le compilateur et l'algorithme de fusion des vecteurs ordonnés de [27].

s	80	160	320	640	1280	2560
Maple 16	0,77	3,2	15	70	279	1268
algebramix	0,63	2,7	12	58	238	1061

Tableau V.5. Produit de deux polynômes creux à 3 variables de taille s, en millisecondes.

V.1.3.5 Résolution des systèmes algébriques

Nous considérons ici un système en les variables $x_1, \ldots, x_n$ et composé de n polynômes f_i. Chaque f_i est construit comme $\left(\sum_{j=1}^{n} \mu_{i,j} x_j\right)\left(\sum_{j=1}^{n} \nu_{i,j} x_j\right) - c_i$, où les coefficients $\mu_{i,j}$, $\nu_{i,j}$, c_i sont des entiers positifs pris aléatoirement sur 30 bits. Dans le tableau V.6 ci-dessous nous indiquons les temps obtenus pour calculer une base standard pour l'ordre du degré lexicographique inverse (DRL en anglais) avec FGb (version interne 8780) [74]

d'une part, et le temps pour obtenir une fibre de remontée avec l'algorithme de résolution géométrique de GEOMSOLVEX. Précisons qu'il s'agit des temps dits utilisateurs selon la terminologie habituelle des systèmes UNIX. Le temps réel pour FGB lorsque $n=5$ est de $1,8$ s et pour $n=6$ de plus de 2 minutes. Lorsque $n=6$, FGB consomme jusqu'à 5 Go de mémoire vive. Le cas $n=7$ n'aboutit pas sur mon ordinateur qui dispose de 8 Go. L'occupation mémoire pour GEOMSOLVEX est juste de 150 Mo pour $n=6$ et 400 Mo pour $n=7$.

n	5	6	7
FGB	0.9	24	∞
GEOMSOLVEX	3,6	30	240

Tableau V.6. Résolution de systèmes algébriques de degré 2 en n variables, en secondes.

V.2 Langage Mathemagix

Les algorithmes pour les entiers, matrices et polynômes peuvent être implantés avec succès dans des langages compilés tels que C ou C++. Malheureusement l'utilisation de différentes bibliothèques avec des conventions variées se révèle compliquée pour un utilisateur non spécialiste. Certaines bibliothèques C++ comme les nôtres font alors office d'intergiciel (*middleware* en anglais) offrant un cadre uniforme pour de telles bibliothèques.

Les algorithmes de haut niveaux développés pour les mathématiques, par exemple pour la géométrie algébrique, sont plus difficilement programmables en C++ de façon générique. D'une part la gestion de la généricité *via* des types paramétrés conduit rapidement à des temps de compilation élevés. D'autre part, les messages d'erreur à la compilation rendent difficilement utilisables des bibliothèques de conception complexe. Il est donc d'usage de fournir des interfaces plus conviviales dans des langages interprétés, comme PYTHON pour SAGE [276], ou bien MAPLE [97] qui intègre des bibliothèques spécialisées comme FGB [74] ou BLAD [36].

Dans les paragraphes suivants nous expliquons comment le language MATHEMAGIX s'intègre dans ce paysage. Nous présentons rapidement les motivations qui ont poussé J. VAN DER HOEVEN à développer un nouveau langage. Ensuite je détaille ma participation dans l'écriture de l'interprète `mmi`, discute les performances atteintes, et présente quelques points sur l'interface avec C++ extraits de l'article [144].

V.2.1 Motivations

Les logiciels commerciaux de calcul formel tels que MAPLE [97], MATHEMATICA [297] ou bien encore MAGMA [31], fournissent un très vaste choix de fonctionnalités. Le cadre de programmation n'est néanmoins pas très bien adapté à la programmation générique et l'absence de compilateur ne permet pas de produire des exécutables autonomes et rapides. Par ailleurs il n'est pas permis à l'utilisateur d'inspecter le code source des fonctionnalités offertes et encore moins de remplacer des routines existantes par du code de son choix.

Le projet SAGE [276] est désormais vu comme un intergiciel incontournable offrant un cadre de programmation confortable et moderne au travers de l'interprète PYTHON. Néanmoins ce dernier ne dispose ni d'un bon compilateur et ni de mécanismes de catégories tels que développées par la communauté de calcul formel au travers des logiciels AXIOM [151] et ALDOR [68].

Le langage MATHEMAGIX a pour but de fournir facilement à l'utilisateur la possibilité de compiler ses programmes, de les interpréter, d'en compiler une partie et d'interpréter le reste, et enfin d'importer et exporter des fonctions depuis et vers le C++. Le langage utilise un *système de type fort* : toutes les instances d'objet ont un type déterminé à

la compilation, et les fonctions peuvent être surchargées. L'utilisateur peut définir de nouvelles catégories, de nouveaux types et classes et aussi définir les héritages depuis l'extérieur des classes. Les types et fonctions peuvent être paramétrés par des catégories telles qu'un anneau, un module sur un anneau, etc. Les détails du langage Mathemagix sont présentés dans le manuel [145].

V.2.2 Compilateur

Ma participation au développement du langage Mathemagix s'est concentrée essentiellement sur l'écriture de l'interprète mmi. Le premier interprète, nommé mmx-light, concernait un sous-ensemble du langage. Il avait été programmé en C++ par J. van der Hoeven. Il a avant tout été un terrain d'expérience pour les questions de calcul symbolique et les mécanismes d'interface avec le C++. Le compilateur actuel étant écrit en Mathemagix, mmx-light a été utilisé pour le compiler avant qu'il ne puisse se compiler par lui-même.

L'interprète mmi développé actuellement intervient au même niveau que la production de code C++ dans le processus de compilation. Le travail que j'ai effectué est d'une certaine manière l'écriture d'un interprète pour C++. La différence est que d'une part seul un sous-ensemble de C++ est spécifiquement concerné et que d'autre part plusieurs objets dont on connaît la provenance peuvent être traités de façons particulières. Les fichiers d'entrées, d'extension .mmx produisent des programmes de type appelé `MMX` à l'issue de l'étape classique d'analyse syntaxique. L'étape suivante consiste à déterminer les types de toutes les sous-expressions et de résoudre les problèmes induits par la surcharge. Pour les détails nous renvoyons le lecteur à l'article [140]. Les programmes *bien typés* résultant de cette étape sont dits de type `PRG`. Des étapes de simplifications et optimisations servent ensuite à réécrire les programmes vers un ensemble d'instructions réduit. Le compilateur mmc permet de produire un programme équivalent en C++, alors que l'interprète construit un programme en mémoire vive, dit *exécutable* de type noté `EXE`, destiné à être exécuté pour obtenir le résultat attendu. La figure V.1 résume le processus de compilation.

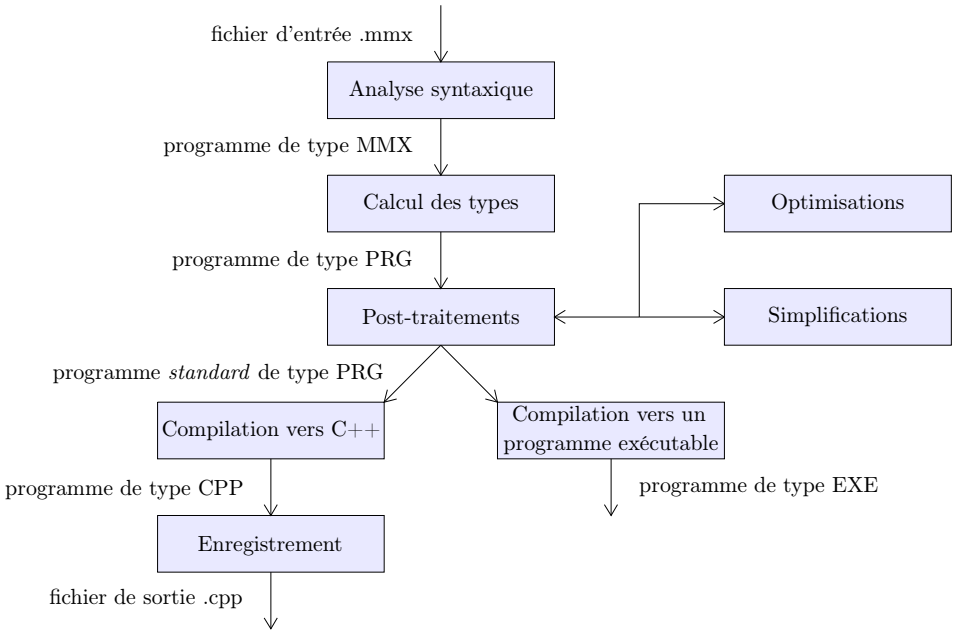

Figure V.1. Processus de compilation de Mathemagix.

Traditionnellement les interprètes sont des programmes qui simulent le comportement d'un processeur rudimentaire pouvant traiter un certain nombre de types prédéfinis souvent élémentaires mais éventuellement complexes. Les choix de conception que nous avons fait avec J. VAN DER HOEVEN ne sont pas allés dans ce sens mais ont été guidés par l'objectif d'efficacité et d'interopérabilité naturelle avec le C++, comme nous l'expliquons dans les paragraphes suivants.

V.2.2.1 PROGRAMMES EXÉCUTABLES

Dans un premier temps j'ai réalisé des structures de données en C++ pour représenter les *programmes exécutables* selon le format de classe abstraite suivant (simplifié de l'implantation réelle pour des raisons de concision) :

```
struct exe_program_rep {
  inline exe_program_rep () {}
  inline virtual ~exe_program_rep () {}
  virtual void Evaluate (void* ptr) const = 0;
};
```

Le type d'un programme exécutable, notée `exe_program` en C++, correspond essentiellement à un pointeur vers une représentation concrète dérivée de `exe_program_rep` et d'un compteur de référence utilisé pour la gestion de la mémoire. La méthode `Evaluate` effectue l'évaluation du programme et stocke le résultat à l'adresse indiquée par l'argument `ptr`. Comme premier exemple voici un programme représentant une constante :

```
template<typename C>
struct exe_program_constant_rep: public exe_program_rep {
  C val;
  exe_program_constant_rep (const C& c) : val (c) {}
  inline void Evaluate (void* ptr) const { *((C*) ptr) = val; }
};
```

La création du programme exécutable correspondant est réalisée par la fonction suivante :

```
template<typename C> inline exe_program
exe_constant (const C& c) {
  return exe_program ((exe_program_rep*) new exe_program_constant_rep<C> (c)); }
```

Comme deuxième exemple, la structure de controle habituelle *if ... then ... else* est implantée comme suit :

```
struct exe_program_if_then_else_rep : exe_program_rep {
  exe_program c, t, e;
  exe_program_if_then_else_rep (const exe_program& c2,
                                const exe_program& t2, const exe_program& e2)
    : c (c2), t (t2), e (e2) {}
  inline void Evaluate (void* ptr) const {
    bool b; c -> Evaluate ((void*) &b);
    if (b) t -> Evaluate (ptr); else e -> Evaluate (ptr); }
};
```

Les programmes exécutables disponibles servent à représenter des variables, des affectations, des boucles, les instructions de branchement, les clôtures, les applications de fonction, etc. Il est possible à tout moment d'ajouter de nouvelles sortes de programmes exécutables sans qu'il y ait la moindre conséquence sur l'efficacité des autres programmes. En fait tous les types de bases et opérations élémentaires utilisées par mmi sont importés depuis le C++ au travers de ce mécanisme. Par exemple l'addition de deux entiers est représentée par un programme exécutable *ad hoc*.

V.2.2.2 Interprète

L'interprète mmi est écrit en Mathemagix. Une première composante concerne l'import du type `exe_program` et de toutes les fonctions permettant de construire les programmes élémentaires. Une deuxième composante est dédiée à la construction de programmes moins élémentaires de plus haut niveau. La composante majeure est ensuite la construction d'un programme exécutable à partir d'un programme bien typé de type `PRG`. Enfin la dernière composante concerne les interfaces utilisateur, l'une textuelle et l'autre avec TEX_{MACS}. Nous pouvons illustrer les performances de notre approche avec le test classique de calcul naïf de la suite de Fibonacci définie comme suit en Mathemagix :

```
fib (n: Int): Int == if n <= 1 then 1 else fib (n-1) + fib (n-2);
```

Dans le tableau V.7 ci-dessous nous donnons les temps mesurés sur mon ordinateur portable décrit précédemment. Les versions d'Ocaml et Python sont celles provenant des MacPorts (`http://www.macports.org`). Les temps avec mmi sont mesurés en activant l'option --optimize. Pour information, nous donnons aussi les temps du code C compilé. Sur la deuxième ligne nous donnons les temps obtenus en calculant avec le type d'entier machine sur 32 bits `int`, et sur la troisième le temps sur le type des nombres à virgule flottante en double précision `double`.

	GCC	Ocaml 4.00.1	Mathemagix 1.0.2	Python 2.7.3
`int`	0,15 s	1,9 s	3,7 s	17,1 s
`double`	0,15 s	3,0 s	3,7 s	21,1 s

Tableau V.7. Temps en secondes pour le calcul de `fib 38`.

Les types `Int` et `Double` de mmi sont importés du C++ *via* les fichiers `basix/int.mmx` et `basix/double.mmx`. Les performances sont similaires avec ces deux types car traités de la même façon par l'interprète. De cette façon l'utilisateur peut obtenir facilement de très bonnes performances pour d'autres types de son choix tels que des nombres à virgule flottante en quadruple précision ou encore des entiers modulaires, etc.

Précisons ici que l'interprète mmi n'est pas vraiment dédié à un usage de type calculatrice car il est nécessaire de systématiquement préciser les types des variables et des fonctions déclarées. Par conséquent un environnement dédié au calcul symbolique est aussi développé indépendamment par J. van der Hoeven au sein du sous-projet caas de Mathemagix. Les performances sont moins élevées que mmi puisque les types concrets des objets symboliques doivent être analysés en cours d'exécution.

V.3 Interface avec C++

Contrairement à C++, le compilateur Mathemagix est capable de gérer la notion de projet. En d'autres termes aucun recours à des systèmes externes comme les GNU Autotools, Cmake ou autre environnement de développement n'est nécessaire. La seule commande de compilation du fichier principal du programme a pour effet de produire directement les fichiers exécutables ou les bibliothèques dynamiques désirées. Par conséquent les fichiers sources doivent contenir toute l'information nécessaire pour leur compilation.

V.3.1 Description

Un fichier d'interface avec C++ commence par définir les informations utiles sur les fichiers C++. Par exemple le fichier `numerix/integer.mmx` pour importer les entiers de taille arbitraire définis dans le fichier d'entête `numerix/integer.hpp` de numerix commence par

```
foreign cpp import {
  cpp_flags   "`numerix-config --cppflags`";
  cpp_libs    "`numerix-config --libs`";
  cpp_include "numerix/integer.hpp";

  class Integer == integer;
  literal_integer: Literal -> Integer == make_literal_integer;

  prefix -: Integer -> Integer == prefix -;
  infix +: (Integer, Integer) -> Integer == infix +;
  infix -: (Integer, Integer) -> Integer == infix -;
  infix *: (Integer, Integer) -> Integer == infix *;
  ...
}
```

À l'installation de la bibliothèque NUMERIX, le script numerix-config est installé et est supposé ensuite accessible pour l'utilisateur. La fonction `literal_integer` permet d'écrire directement des entiers de taille arbitraire. Son argument de type `Literal` est produit lors de la phase d'analyse syntaxique. Pour utiliser les entiers longs depuis MATHEMAGIX il suffit tout simplement de procéder comme avec le fichier `exemple.mmx` suivant :

```
include "basix/port.mmx";
include "numerix/integer.mmx";
a: Integer == 10;
mmout << a! << lf;
```

La compilation et l'exécution se déroulent comme suit :

```
Shell] mmc exemple.mmx
Shell] ./exemple
```

```
3628800
```

L'interprète mmi peut être utilisé directement depuis TEXMACS via le menu Insérer→Session→Mmi, comme illustré par les calculs suivants :

```
Mmi] include "basix/port.mmx"
Mmi] include "numerix/integer.mmx"
Mmi] include "numerix/rational.mmx"
Mmi] include "numerix/floating.mmx"
Mmi] a: Integer == 100
Mmi] a!
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414\
639761565182862536979208272237582511852109168640000000000000000000000000
```

```
Mmi] probable_next_prime a!
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414\
639761565182862536979208272237582511852109168640000000000000000000000229
```

```
Mmi] help cos
```

```
cos : Double -> Double                    (/Users/lecerf/tmp/mmx/basix/mmx/double.mmx:42:4)
cos : Floating -> Floating                (/Users/lecerf/tmp/mmx/numerix/mmx/floating.mmx:49:2)
```

```
Mmi] cos (1.0 :> Double)
```

```
0.54030230586814
```

```
Mmi] [ cos (i :> Floating) | i: Int in 0..10 ]
```

[1.00000000000000000000, 0.540302305868139717414, −0.416146836547142387008, −0.98999249660\
0445457278, −0.653643620863611914628, 0.283662185463226264461, 0.960170286650366020529, 0.753\
902254343304638155, −0.145500033808613525867, −0.911130261884676988346]

Lors de leur première utilisation les fichiers inclus qui importent des fonctionnalités du C++ sont compilés sous la forme d'une bibliothèque dynamique qui est ensuite chargée par l'interprète.

V.3.2 TYPES PARAMÉTRÉS

L'une des forces du mécanisme d'interface est de pouvoir importer depuis le C++ des types et fonctions paramétrées. Dans l'exemple suivant nous commençons par exporter la catégorie `Ring` vers C++. Ensuite nous importons notre classe de polynômes d'ALGEBRAMIX pour être utilisée depuis MATHEMAGIX.

```
foreign cpp export
category Ring == {
  convert: Int -> This == keyword constructor;
  prefix -: This -> This == prefix -;
  infix +: (This, This) -> This == infix +;
  infix -: (This, This) -> This == infix -;
  infix *: (This, This) -> This == infix *;
}
...
foreign cpp import {
  ...
  class Pol (R: Ring) == polynomial R;

  forall (R: Ring) {
    pol: Tuple R -> Pol R == keyword constructor;
    upgrade: R -> Pol R == keyword constructor;

    deg: Pol R -> Int == deg;
    postfix []: (Pol R, Int) -> R == postfix [];

    prefix -: Pol R -> Pol R == prefix -;
    infix +: (Pol R, Pol R) -> Pol R == infix +;
    infix -: (Pol R, Pol R) -> Pol R == infix -;
    infix *: (Pol R, Pol R) -> Pol R == infix *;
    ...
  }
}
```

Le fichier d'exemple `pol.mmx` suivant

```
include "basix/port.mmx";
include "algebramix/polynomial.mmx";
p0: Polynomial Int == polynomial (1, 2, 3);
mmout << "p0= " << p0 << lf;
p1: Polynomial Int == p0 * p0;
mmout << "p1= " << p1 << lf;
```

s'utilise comme suit :

```
Shell] mmc pol.mmx
Shell] ./pol
```

```
p0= 3 * x^2 + 2 * x + 1
p1= 9 * x^4 + 12 * x^3 + 10 * x^2 + 4 * x + 1
```

Nous ne décrirons pas ici les mécanismes utilisés pour importer des types paramétrés depuis C++ et renvoyons le lecteur vers notre article [144].

V.3.3 INTERFACES EXISTENTES

Disposer de mécanismes confortables pour importer des bibliothèques dans MATHEMAGIX ne règle pas pour autant toutes les questions du point de vue d'un utilisateur souhaitant utiliser plusieurs bibliothèques en même temps. Par exemple il faut que les différentes bibliothèques partagent les mêmes types de base. J'ai réalisé pour MATHEMAGIX des interfaces pour les bibliothèques BLAD [36], FGB [74] et PARI/GP [235]. Dans l'exemple `exemple_pari.mmx` suivant nous montrons comment utiliser la fonction `nfbasis` pour calculer une base de la clôture intégrale du corps de nombre défini par le polynôme $x^2+x-1001$:

```
include "basix/fundamental.mmx";
include "mpari/pari.mmx";
Pol ==> Polynomial Integer;
x: Pol == polynomial (0 :> Integer, 1 :> Integer);
p: Pol == x^2 + x - 1001;
mmout << pari_nfbasis p << lf;
```

```
Shell] mmc exemple_pari.mmx
Shell] ./exemple_pari
[1, 1 / 3 * x - 1 / 3]
```

Références

1 F. K. Abu Salem. An efficient sparse adaptation of the polytope method over $\mathbb{F}_p$ and a record-high binary bivariate factorisation. *J. Symbolic Comput.*, vol. 43, n°5, 2008, p. 311–341.

2 F. K. Abu Salem, S. Gao, A. G. B. Lauder. Factoring polynomials via polytopes. In J. Schicho (éd.), *Proc. ISSAC '04*. p. 4–11. ACM, 2004.

3 A. V. Aho, J. E. Hopcroft, J. D. Ullman. *Design and Analysis Of Computer Algorithms*. Pearson Education, 1974.

4 V. S. Alagar, D. K. Probst. A fast, low-space algorithm for multiplying dense multivariate polynomials. *ACM Trans. Math. Softw.*, vol. 13, n°1, 1987, p. 35–57.

5 E. L. Allgower, K. Böhmer, A. Hoy, V. Janovský. Direct methods for solving singular nonlinear equations. *ZAMM Z. Angew. Math. Mech.*, vol. 79, n°4, 1999, p. 219–231.

6 A. Antoniou. *Digital Filters: Analysis and Design*. McGraw-Hill Book Co., 1979.

7 C. Bajaj, J. Canny, T. Garrity, J. Warren. Factoring rational polynomials over the complex numbers. *SIAM J. Comput.*, vol. 22, n°2, 1993, p. 318–331.

8 B. Bank, M. Giusti, J. Heintz, G. Lecerf, G. Matera, P. Solernó. Degeneracy loci and polynomial equation solving. 2013. http://www2.mathematik.hu-berlin.de/publ/pre/2013/P-13-08.pdf.

9 B. Bank, M. Giusti, J. Heintz, G. M. Mbakop. Polar varieties, real equation solving, and data structures: the hypersurface case. *J. Complexity*, vol. 13, n°1, 1997, p. 5–27.

10 B. Bank, M. Giusti, J. Heintz, G. M. Mbakop. Polar varieties and efficient real elimination. *Math. Z.*, vol. 238, n°1, 2001, p. 115–144.

11 B. Bank, M. Giusti, J. Heintz, L. M. Pardo. Generalized polar varieties and an efficient real elimination. *Kybernetika*, vol. 40, n°5, 2004, p. 519–550.

12 B. Bank, M. Giusti, J. Heintz, L. M. Pardo. Generalized polar varieties: geometry and algorithms. *J. Complexity*, vol. 21, n°4, 2005, p. 377–412.

13 B. Bank, M. Giusti, J. Heintz, M. Safey El Din, É. Schost. On the geometry of polar varieties. *Appl. Algebra Eng. Commun. Comput.*, vol. 21, n°1, 2010, p. 33–83.

14 D. J. Bates, J. D. Hauenstein, A. J. Sommese, C. W. Wampler. Software for numerical algebraic geometry: A paradigm and progress towards its implementation. In M. Stillman, J. Verschelde, N. Takayama (éd.), *Software for Algebraic Geometry. The IMA Volumes in Mathematics and its Applications*, vol. 148, p. 1–14. Springer-Verlag, 2008.

15 D. Bates, J. Hauenstein, A. Sommese. Efficient path tracking methods. *Numer. Algorithms*, vol. 58, n°4, 2011, p. 451–459.

16 T. Becker, V. Weispfenning. *Gröbner bases. A computational approach to commutative algebra*. Springer-Verlag, 1993, *Graduate Texts in Mathematics*, vol. 141.

17 K. Belabas, M. van Hoeij, M., J. Klüners, A. Steel. Factoring polynomials over global fields. *J. Théor. Nombres Bordeaux*, vol. 21, n°1, 2009, p. 15–39.

18 M. Ben-Or, P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In J. Simon (éd.), *Proc. STOC '88*. p. 301–309. ACM, 1988.

19 S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inform. Process. Lett.*, vol. 18, 1984, p. 147–150.

20 L. Bernardin, M. B. Monagan. Efficient multivariate factorization over finite fields. In T. Mora, H. F. Mattson (éd.), *Proc. AAECC-12. Lect. Notes Comput. Sci.*, vol. 1255, p. 15–28. Springer-Verlag, 1997.

21 D. Bernstein. The transposition principle, 2006. http://cr.yp.to/transposition.html.

22 J. Berthomieu, J. van der Hoeven, G. Lecerf. Relaxed algorithms for p-adic numbers. *J. Théor. Nombres Bordeaux*, vol. 23, n°3, 2011, p. 541–577.

23 J. Berthomieu, R. Lebreton. Relaxed p-adic Hensel lifting for algebraic systems. In J. van der Hoeven, van Hoeij M. (éd.), *Proc. ISSAC '12*. p. 59–66. ACM, 2012.

24 J. Berthomieu, G. Lecerf. Reduction of bivariate polynomials from convex-dense to dense, with application to factorizations. *Math. Comp.*, vol. 81, n°279, 2012, p. 1799–1821.

25 J. Berthomieu, G. Lecerf, G. Quintin. Polynomial root finding over local rings and application to error correcting codes. *Appl. Alg. Eng. Comm. Comp.*, vol. 24, n°6, 2013, p. 413–443. http://dx.doi.org/10.1007/s00200-013-0200-5.

26 D. Bini, V. Pan. *Polynomial and matrix computations (vol. 1): fundamental algorithms*. Birkhäuser, 1994.

27 G. E. Blelloch, P. B. Gibbons, H. V. Simhadri. Low depth cache-oblivious algorithms. In F. Meyer auf der Heide, C. Phillips (éd.), *Proc. SPAA '10*. p. 189–199. ACM, 2010.

28 L. Blum, F. Cucker, M. Shub, S. Smale. *Complexity and real computation*. Springer-Verlag, New York, 1998.

29 A. Bompadre, G. Matera, R. Wachenchauzer, A. Waissbein. Polynomial equation solving by lifting procedures for ramified fibers. *Theoret. Comput. Sci.*, vol. 315, n°2-3, 2004, p. 334–369.

30 J. L. Bordewijk. Inter-reciprocity applied to electrical networks. *Appl. Sci. Res. B.*, vol. 6, 1956, p. 1–74.

31 W. Bosma, J. Cannon, C. Playoust. The Magma algebra system. I. the user language. *J. Symbolic Comput.*, vol. 24, n°3-4, 1997, p. 235–265.

32 A. Bostan. *Algorithmique efficace pour des opérations de base en calcul formel*. Thèse de doctorat, École polytechnique, France, 2003.

33 A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, É. Schost. Algorithmes efficaces en calcul formel, notes du cours 2-22 du MPRI, année 2013–2014, version de 2013. https://wikimpri.dptinfo.ens-cachan.fr/doku.php?id=cours:c-2-22.

34 A. Bostan, G. Lecerf, B. Salvy, É. Schost, B. Wiebelt. Complexity issues in bivariate polynomial factorization. In J. Schicho (éd.), *Proc. ISSAC '04*. p. 42–49. ACM, 2004.

35 A. Bostan, G. Lecerf, É. Schost. Tellegen's principle into practice. In H. Hong (éd.), *Proc. ISSAC '03*. p. 37–44. ACM, 2003.

36 F. Boulier, *et al.* Bibliothèques lilloises d'algèbre différentielle (BLAD), 2004–2013. http://www.lifl.fr/~boulier.

37 M. Bronstein. Computer algebra algorithms for linear ordinary differential and difference equations. In C. Casacuberta, R. M. Miro-Roig, J. Verdera, S. Xambo-Descamps (éd.), *European Congress of Mathematics, Vol. II (Barcelona, 2000). Progr. Math.*, vol. 202, p. 105–119. Birkhäuser, 2001.

38 P. Bürgisser, M. Clausen, M. A. Shokrollahi. *Algebraic complexity theory*. Springer-Verlag, 1997.

39 A. Cafure, G. Matera. Fast computation of a rational point of a variety over a finite field. *Math. Comp.*, vol. 75, n°256, 2006, p. 2049–2085.

40 A. Cafure, G. Matera. Improved explicit estimates on the number of solutions of equations over a finite field. *Finite Fields Appl.*, vol. 12, n°2, 2006, p. 155–185.

41 J. Canny, E. Kaltofen, Y. Lakshman. Solving systems of non-linear polynomial equations faster. In G. H. Gonnet (éd.), *Proc. ISSAC '89*. p. 121–128. ACM, 1989.

42 D. G. Cantor, E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Infor.*, vol. 28, n°7, 1991, p. 693–701.

43 D. Castro, M. Giusti, J. Heintz, G. Matera, L. M. Pardo. The hardness of polynomial equation solving. *Found. Comput. Math.*, vol. 3, n°4, 2003, p. 347–420.

44 D. Castro, J. L. Montaña, L. M. Pardo, J. San Martín. The distribution of condition numbers of rational data of bounded bit length. *Found. Comput. Math.*, vol. 2, n°1, 2002, p. 1–52.

45 D. Castro, L. M. Pardo, K. Hägele, J. E. Morais. Kronecker's and Newton's approaches to solving: a first comparison. *J. Complexity*, vol. 17, n°1, 2001, p. 212–303.

46 D. Castro, L. M. Pardo, J. San Martín. Systems of rational polynomial equations have polynomial size approximate zeros on the average. *J. Complexity*, vol. 19, n°2, 2003, p. 161–209.

47 G. Chèze. *Des méthodes symboliques-numériques et exactes pour la factorisation absolue des polynômes en deux variables*. Thèse de doctorat, Université de Nice-Sophia Antipolis, France, 2004.

48 G. Chèze, A. Galligo. Four lectures on polynomial absolute factorization. In A. Dickenstein, I. Z. Emiris (éd.), *Solving polynomial equations: foundations, algorithms, and applications*. *Algorithms Comput. Math.*, vol. 14, p. 339–392. Springer-Verlag, 2005.

49 G. Chèze, G. Lecerf. Lifting and recombination techniques for absolute factorization. *J. Complexity*, vol. 23, n°3, 2007, p. 380–420.

50 I. S. Cohen. On the structure and ideal theory of complete local rings. *Trans. Amer. Math. Soc.*, vol. 59, 1946, p. 54–106.

51 S. A. Cook. *On the minimum computation time of functions*. Thèse de doctorat, Harvard University, USA, 1966.

52 J. W. Cooley, J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Computat.*, vol. 19, 1965, p. 297–301.

53 D. A. Cox, J. Little, D. O'Shea. *Ideals, varieties, and algorithms. An introduction to computational algebraic geometry and commutative algebra*. Springer-Verlag, 1997, 2^e édition, *Undergraduate Texts in Mathematics*.

54 D. A. Cox, J. Little, D. O'Shea. *Using algebraic geometry*. Springer-Verlag, 2005, 2^e édition, *Graduate Texts in Mathematics*.

55 S. Czapor, K. Geddes, G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

56 J. H. Davenport, Y. Siret, É. Tournier. *Calcul formel : systèmes et algorithmes de manipulations algébriques*. Masson, 1987.

57 J. H. Davenport, B. M. Trager. Factorization over finitely generated fields. In P. S. Wang (éd.), *Proc. SYMSAC'81*. p. 200–205. ACM, 1981.

58 M. De Leo, E. Dratman, G. Matera. Numeric vs. symbolic homotopy algorithms in polynomial system solving: a case study. *J. Complexity*, vol. 21, n°4, 2005, p. 502–531.

59 D. W. Decker, H. B. Keller, C. T. Kelley. Convergence rates for Newton's method at singular points. *SIAM J. Numer. Anal.*, vol. 20, n°2, 1983, p. 296–314.

60 D. W. Decker, C. T. Kelley. Newton's method at singular points. I. *SIAM J. Numer. Anal.*, vol. 17, n°1, 1980, p. 66–70.

61 D. W. Decker, C. T. Kelley. Newton's method at singular points. II. *SIAM J. Numer. Anal.*, vol. 17, n°3, 1980, p. 465–471.

62 D. W. Decker, C. T. Kelley. Convergence acceleration for Newton's method at singular points. *SIAM J. Numer. Anal.*, vol. 19, n°1, 1982, p. 219–229.

63 D. W. Decker, C. T. Kelley. Expanded convergence domains for Newton's method at nearly singular roots. *SIAM J. Sci. Statist. Comput.*, vol. 6, n°4, 1985, p. 951–966.

64 W. Decker, G.-M. Greuel, G. Pfister, H. Schönemann. Singular 3-1-6 — A computer algebra system for polynomial computations, 2012. http://www.singular.uni-kl.de.

65 J.-P. Dedieu, M.-H. Kim, M. Shub, F. Tisseur. Implicit gamma theorems. I. Pseudoroots and pseudospectra. *Found. Comput. Math.*, vol. 3, n°1, 2003, p. 1–31.

66 M. Demazure. Réécriture et bases standard. Notes informelles de calcul formel. Centre de Mathématiques, École polytechnique, Palaiseau, France, 1985. http://www.stix.polytechnique.fr/publications/1984-1994.html.

67 M. Demazure. Le théorème de complexité de Mayr et Meyer. In *Géométrie algébrique et applications, I (La Rábida, 1984)*. *Travaux en Cours*, vol. 22, p. 35–58. Hermann, 1987.

68 L. Dragan, S. Huerter, C. Oancea, S. Watt. Aldor programming language. http://www.aldor.org, 2007.

69 C. Durvye. Evaluation techniques for zero-dimensional primary decomposition. *J. Symbolic Comput.*, vol. 44, n°9, 2009, p. 1089–1113.

70 C. Durvye, G. Lecerf. A concise proof of the Kronecker polynomial system solver from scratch. *Exposition. Math.*, vol. 26, n°2, 2007, p. 101–139.

71 J. A. Eagon, D. G. Northcott. Ideals defined by matrices and a certain complex associated with them. *Proc. Roy. Soc. Ser. A*, vol. 269, 1962, p. 188–204.

72 D. Eisenbud. *Commutative algebra. With a view toward algebraic geometry*. Springer-Verlag, 1995, *Graduate Texts in Mathematics*.

73 R. Fateman. Polynomial multiplication, powers and asymptotic analysis: some comments. *SIAM J. Comput.*, vol. 3, n°3, 1974, p. 4–15.

74 J.-C. Faugère. FGb: A Library for Computing Gröbner Bases. In K. Fukuda, J. van der Hoeven, M. Joswig, N. Takayama (éd.), *Mathematical Software - ICMS 2010*. *Lect. Notes Comput. Sci.*, vol. 6327, p. 84–87. Springer-Verlag, 2010.

75 N. Fitchas, M. Giusti, F. Smietanski. Sur la complexité du théorème des zéros. In M. Florenzano, J. Guddat, *et al.* (éd.), *Approximation and optimization in the Caribbean, II (Havana, 1993)*. *Approx. Optim.*, vol. 8, p. 274–329. Lang, Frankfurt am Main, 1995.

76 L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Software*, vol. 33, n°2, 2007. Software available at http://www.mpfr.org.

77 M. Frigo, S. G. Johnson. The design and implementation of FFTW3. *Proc. IEEE*, vol. 93, n°2, 2005, p. 216–231.

78 A. Fröhlich, J. C. Shepherdson. On the factorisation of polynomials in a finite number of steps. *Math. Z.*, vol. 62, 1955, p. 331–334.

79 A. Fröhlich, J. C. Shepherdson. Effective procedures in field theory. *Philos. Trans. Roy. Soc. London. Ser. A.*, vol. 248, 1956, p. 407–432.

80 M. Fürer. Faster integer multiplication. In D. Johnson, U. Feige (éd.), *Proc. STOC '07*. p. 57–66. ACM, 2007.

81 M. Fürer. Faster integer multiplication. *SIAM J. Comput.*, vol. 39, n°3, 2009, p. 979–1005.

82 S. Gao. Factoring multivariate polynomials via partial differential equations. *Math. Comp.*, vol. 72, n°242, 2003, p. 801–822.

83 S. GAO. Absolute irreducibility of polynomials via Newton polytopes. *J. Algebra*, vol. 237, n°2, 2001, p. 501–520.

84 S. GAO, A. G. B. LAUDER. Hensel lifting and bivariate polynomial factorisation over finite fields. *Math. Comp.*, vol. 71, n°240, 2002, p. 1663–1676.

85 S. GARG, É. SCHOST. Interpolation of polynomials given by straight-line programs. *Theoret. Comput. Sci.*, vol. 410, n°27-29, 2009, p. 2659–2662.

86 M. GASTINEAU, J. LASKAR. Development of TRIP: Fast sparse multivariate polynomial multiplication using burst tries. In V. N. ALEXANDROV, G. D. VAN ALBADA, P. M. A. SLOOT, J. J. DONGARRA (éd.), *Computational Science - ICCS 2006. Lect. Notes Comput. Sci.*, vol. 3992, p. 446–453. Springer-Verlag, 2006.

87 J. VON ZUR GATHEN. Factoring sparse multivariate polynomials. In M. BLUM, *et al.* (éd.), *Proc. FOCS '83*. p. 172–179. IEEE, 1983.

88 J. VON ZUR GATHEN. Hensel and Newton methods in valuation rings. *Math. Comp.*, vol. 42, n°166, 1984, p. 637–661.

89 J. VON ZUR GATHEN. Irreducibility of multivariate polynomials. *J. Comput. System Sci.*, vol. 31, n°2, 1985, p. 225–264.

90 J. VON ZUR GATHEN. Who was who in polynomial factorization. In B. TRAGER (éd.), *Proc. ISSAC '06*. p. 1–2. ACM, 2006.

91 J. VON ZUR GATHEN, J. GERHARD. *Modern computer algebra*. Cambridge Univ. Press, 2003, 2^e édition.

92 J. VON ZUR GATHEN, E. KALTOFEN. Factoring sparse multivariate polynomials. *J. Comput. System Sci.*, vol. 31, n°2, 1985, p. 265–287.

93 J. VON ZUR GATHEN, E. KALTOFEN. Factorization of multivariate polynomials over finite fields. *Math. Comp.*, vol. 45, n°171, 1985, p. 251–261.

94 P. GAUDRY, R. BRENT, P. ZIMMERMANN, E. THOMÉ. gf2x, a C/C++ software package for fast arithmetic in GF(2)[x], 2012–2013. http://gf2x.gforge.inria.fr.

95 P. GAUDRY, L. SANSELME, E. THOMÉ. MPFQ: Fast finite fields, 2010–2013. http://mpfq.gforge.inria.fr.

96 P. GAUDRY, É. SCHOST. Modular equations for hyperelliptic curves. *Math. Comp.*, vol. 74, n°249, 2005, p. 429–454.

97 K. GEDDES, G. GONNET, MAPLESOFT. Maple. http://www.maplesoft.com/products/maple, 1980–2013.

98 P. GIANNI, B. TRAGER. Square-free algorithms in positive characteristic. *Appl. Algebra Engrg. Comm. Comput.*, vol. 7, n°1, 1996, p. 1–14.

99 M. GIUSTI, K. HÄGELE, J. HEINTZ, J. L. MONTAÑA, J. E. MORAIS, L. M. PARDO. Lower bounds for Diophantine approximations. *J. Pure Appl. Algebra*, vol. 117/118, 1997, p. 277–317.

100 M. GIUSTI, K. HÄGELE, G. LECERF, J. MARCHAND, B. SALVY. The projective Noether Maple package: computing the dimension of a projective variety. *J. Symbolic Comput.*, vol. 30, n°3, 2000, p. 291–307.

101 M. GIUSTI, J. HEINTZ. La détermination des points isolés et de la dimension d'une variété algébrique peut se faire en temps polynomial. In D. EISENBUD, L. ROBBIANO (éd.), *Computational algebraic geometry and commutative algebra (Cortona, 1991). Sympos. Math., XXXIV*, p. 216–256. Cambridge Univ. Press, 1993.

102 M. GIUSTI, J. HEINTZ. Kronecker's smart, little black boxes. In R. A. DEVORE, A. ISERLES, E. SÜLI (éd.), *Foundations of computational mathematics (Oxford, 1999). London Math. Soc. Lecture Note Ser.*, vol. 284, p. 69–104. Cambridge Univ. Press, 2001.

103 M. GIUSTI, J. HEINTZ, J. E. MORAIS, J. MORGENSTERN, L. M. PARDO. Straight-line programs in geometric elimination theory. *J. Pure Appl. Algebra*, vol. 124, n°1-3, 1998, p. 101–146.

104 M. GIUSTI, J. HEINTZ, J. E. MORAIS, L. M. PARDO. When polynomial equation systems can be "solved" fast? In G. COHEN, M. GIUSTI, T. MORA (éd.), *Proc. AAECC-11. Lect. Notes Comput. Sci.*, vol. 948, p. 205–231. Springer-Verlag, 1995.

105 M. GIUSTI, J. HEINTZ, J. E. MORAIS, L. M. PARDO. Le rôle des structures de données dans les problèmes d'élimination. *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 325, n°11, 1997, p. 1223–1228.

106 M. GIUSTI, J. HEINTZ, J. SABIA. On the efficiency of effective Nullstellensätze. *Comput. Complexity*, vol. 3, n°1, 1993, p. 56–95.

107 M. GIUSTI, G. LECERF, B. SALVY. A Gröbner free alternative for polynomial system solving. *J. Complexity*, vol. 17, n°1, 2001, p. 154–211.

108 M. GIUSTI, G. LECERF, B. SALVY, J.-C. YAKOUBSOHN. On location and approximation of clusters of zeros of analytic functions. *Found. Comput. Math.*, vol. 5, n°3, 2005, p. 257–311.

109 M. GIUSTI, G. LECERF, B. SALVY, J.-C. YAKOUBSOHN. On location and approximation of clusters of zeros: Case of embedding dimension one. *Found. Comput. Math.*, vol. 7, n°1, 2007, p. 1–49.

110 M. GIUSTI, É. SCHOST. Solving some overdetermined polynomial systems. In J. JOHNSON, H. PARK, E. KALTOFEN (éd.), *Proc. ISSAC '99*. p. 1–8. ACM, 1999.

111 W. GOVAERTS. Computation of singularities in large nonlinear systems. *SIAM J. Numer. Anal.*, vol. 34, n°3, 1997, p. 867–880.

112 T. GRANLUND, *et al.* GMP, the GNU multiple precision arithmetic library, 1991. http://gmplib.org.

113 G.-M. GREUEL, G. PFISTER. *A Singular introduction to commutative algebra*. Springer-Verlag, 2002.

114 A. GRIEWANK. Starlike domains of convergence for Newton's method at singularities. *Numer. Math.*, vol. 35, n°1, 1980, p. 95–111.

115 A. GRIEWANK. On solving nonlinear equations with simple singularities or nearly singular solutions. *SIAM Rev.*, vol. 27, n°4, 1985, p. 537–563.

116 A. GRIEWANK, M. R. OSBORNE. Analysis of Newton's method at irregular singularities. *SIAM J. Numer. Anal.*, vol. 20, n°4, 1983, p. 747–773.

117 A. GRIEWANK, G. W. REDDIEN. Characterization and computation of generalized turning points. *SIAM J. Numer. Anal.*, vol. 21, n°1, 1984, p. 176–185.

118 A. GRIEWANK, G. W. REDDIEN. The approximate solution of defining equations for generalized turning points. *SIAM J. Numer. Anal.*, vol. 33, n°5, 1996, p. 1912–1920.

119 D. GRIGORIEV, M. KARPINSKI, M. F. SINGER. Computational complexity of sparse rational interpolation. *SIAM J. Comput.*, vol. 23, n°1, 1994, p. 1–11.

120 D. Y. GRIGORIEV, M. KARPINSKI, M. F. SINGER. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput.*, vol. 19, n°6, 1990, p. 1059–1063.

121 D. Y. GRIGORIEV, A. L. CHISTOV. Fast factorization of polynomials into irreducible ones and the solution of systems of algebraic equations. *Dokl. Akad. Nauk SSSR*, vol. 275, n°6, 1984, p. 1302–1306.

122 V. GURUSWAMI, M. SUDAN. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Trans. Inform. Theory*, vol. 45, 1998, p. 1757–1767.

123 G. HANROT, M. QUERCIA, P. ZIMMERMANN. *The middle product algorithm, I. Speeding up the division and square root of power series*. Technical report, RR INRIA 4664, 2002.

124 G. HANROT, P. ZIMMERMANN. A long note on Mulders' short product. *J. Symbolic Comput.*, vol. 37, n°3, 2004, p. 391–401.

125 J. HEINTZ, T. KRICK, S. PUDDU, J. SABIA, A. WAISSBEIN. Deformation techniques for efficient polynomial equation solving. *J. Complexity*, vol. 16, n°1, 2000, p. 70–109.

126 J. HEINTZ, G. MATERA, L. M. PARDO, R. WACHENCHAUZER. The intrinsic complexity of parametric elimination methods. *Electronic J. of SADIO*, vol. 1, n°1, 1998, p. 37–51.

127 J. Heintz, G. Matera, A. Waissbein. On the time-space complexity of geometric elimination procedures. *Appl. Algebra Engrg. Comm. Comput.*, vol. 11, n°4, 2001, p. 239–296.

128 J. Heintz, M. Sieveking. Absolute primality of polynomials is decidable in random polynomial time in the number of variables. In S. Even, O. Kariv (éd.), *Automata, languages and programming (Akko, 1981). Lect. Notes Comput. Sci.*, vol. 115, p. 16–28. Springer-Verlag, 1981.

129 G. Hermann. Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. *Math. Ann.*, vol. 95, n°1, 1926, p. 736–788.

130 M. Hermann, P. Kunkel, W. Middelmann. Augmented systems for the computation of singular points in Banach space problems. *ZAMM Z. Angew. Math. Mech.*, vol. 78, n°1, 1998, p. 39–50.

131 D. Hilbert. Ueber die Irreducibilität ganzer rationaler Functionen mit ganzzahligen Coefficienten. *J. Reine Angew. Math.*, vol. 110, 1892.

132 M. Hoeij, J.-F. Ragot, F. Ulmer, J.-A. Weil. Liouvillian solutions of linear differential equations of order three and higher. *J. Symbolic Comput.*, vol. 28, n°4-5, 1999, p. 589–609.

133 J. van der Hoeven. Lazy multiplication of formal power series. In W. W. Küchlin (éd.), *Proc. ISSAC '97*, p. 17–20. 1997.

134 J. van der Hoeven. Relax, but don't be too lazy. *J. Symbolic Comput.*, vol. 34, n°6, 2002, p. 479–542.

135 J. van der Hoeven. The truncated Fourier transform and applications. In J. Schicho (éd.), *Proc. ISSAC '04*. p. 290–296. ACM, 2004.

136 J. van der Hoeven. Effective analytic functions. *J. Symbolic Comput.*, vol. 39, n°3–4, 2005, p. 433–449.

137 J. van der Hoeven. New algorithms for relaxed multiplication. *J. Symbolic Comput.*, vol. 42, n°8, 2007, p. 792–802.

138 J. van der Hoeven. *Relaxed resolution of implicit equations*. Technical report, HAL, 2009. http://hal.archives-ouvertes.fr/hal-00441977/fr/.

139 J. van der Hoeven. Newton's method and FFT trading. *J. Symbolic Comput.*, vol. 45, n°8, 2010.

140 J. van der Hoeven. Overview of the Mathemagix type system. In *Electronic proc. ASCM '12*. Beijing, China, 2012. Available from http://hal.archives-ouvertes.fr/hal-00702634.

141 J. van der Hoeven, A. Grozin, M. Gubinelli, G. Lecerf, F. Poulain, D. Raux. GNU TEXmacs: a scientific editing platform. *ACM SIGSAM Communications in Computer Algebra*, vol. 47, n°2, 2013, p. 59–62.

142 J. van der Hoeven, R. Lebreton, É. Schost. Structured FFT and TFT: symmetric and lattice polynomials. In M. Monagan, G. Cooperman, M. Giesbrecht (éd.), *Proc. ISSAC '13*. p. 355–362. ACM, 2013.

143 J. van der Hoeven, G. Lecerf. On the complexity of multivariate blockwise polynomial multiplication. In J. van der Hoeven, M. van Hoeij (éd.), *Proc. ISSAC '12*. p. 211–218. ACM, 2012.

144 J. van der Hoeven, G. Lecerf. Interfacing Mathemagix with C++. In M. Monagan, G. Cooperman, M. Giesbrecht (éd.), *Proc. ISSAC '13*. p. 363–370. ACM, 2013.

145 J. van der Hoeven, G. Lecerf. *Mathemagix User Guide*. HAL, 2013. http://hal.archives-ouvertes.fr/hal-00785549.

146 J. van der Hoeven, G. Lecerf. On the bit-complexity of sparse polynomial and series multiplication. *J. Symbolic Comput.*, vol. 50, 2013, p. 227–254.

147 J. van der Hoeven, G. Lecerf, B. Mourrain, *et al.* Mathemagix, 2002–2013. http://www.mathemagix.org.

148 J. van der Hoeven, É. Schost. Multi-point evaluation in higher dimensions. *Appl. Alg. Eng. Comm. Comp.*, vol. 24, n°1, 2013, p. 37–52.

149 A. Hoy. A relation between Newton and Gauss-Newton steps for singular nonlinear equations. *Computing*, vol. 40, n°1, 1988, p. 19–27.

150 A. Hoy. Analysis of a bordering approach for solving singular nonlinear equations. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, vol. 38, n°4, 1989, p. 115–121.

151 R. D. Jenks, R. Sutor. *AXIOM: the scientific computation system*. Springer-Verlag, 1992.

152 G. Jeronimo, T. Krick, J. Sabia, M. Sombra. The computational complexity of the Chow form. *Found. Comput. Math.*, vol. 4, n°1, 2004, p. 41–117.

153 G. Jeronimo, G. Matera, P. Solernó, A. Waissbein. Deformation techniques for sparse systems. *Found. Comput. Math.*, vol. 9, n°1, 2009, p. 1–50.

154 G. Jeronimo, S. Puddu, J. Sabia. Computing Chow forms and some applications. *J. Algorithms*, vol. 41, n°1, 2001, p. 52–68.

155 G. Jeronimo, J. Sabia. Probabilistic equidimensional decomposition. *C. R. Acad. Sci. Paris Sér. I Math.*, vol. 331, n°6, 2000, p. 485–490.

156 G. Jeronimo, J. Sabia. Effective equidimensional decomposition of affine varieties. *J. Pure Appl. Algebra*, vol. 169, n°2-3, 2002, p. 229–248.

157 S. C. Johnson. Sparse polynomial arithmetic. *SIGSAM Bull.*, vol. 8, n°3, 1974, p. 63–71.

158 S. G. Johnson, M. Frigo. Implementing FFTs in practice. In C. S. Burrus (éd.), *Fast Fourier Transforms*. Connexions, Rice University, Houston TX, 2008.

159 J.-P. Jouanolou. *Théorèmes de Bertini et applications*. Birkhäuser Boston, 1983, *Progress in Mathematics*, vol. 42.

160 E. Kaltofen. Polynomial factorization. In B. Buchberger, G. Collins, R. Loos (éd.), *Computer algebra*. p. 95–113. Springer-Verlag, 1982.

161 E. Kaltofen. A polynomial reduction from multivariate to bivariate integral polynomial factorization. In H. R. Lewis, *et al.* (éd.), *Proc. STOC '82*. p. 261–266. ACM, 1982.

162 E. Kaltofen. Effective Hilbert irreducibility. *Inform. and Control*, vol. 66, n°3, 1985, p. 123–137.

163 E. Kaltofen. Fast parallel absolute irreducibility testing. *J. Symbolic Comput.*, vol. 1, n°1, 1985, p. 57–67.

164 E. Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput.*, vol. 14, n°2, 1985, p. 469–489.

165 E. Kaltofen. Sparse Hensel lifting. In B. F. Caviness (éd.), *Proc. EUROCAL '85 (Vol. 2). Lect. Notes Comput. Sci.*, vol. 204, p. 4–17. Springer-Verlag, 1985.

166 E. Kaltofen. Polynomial factorization 1982–1986. In D. Chudnovsky (éd.), *Computers in mathematics (Stanford, CA, 1986). Lecture Notes in Pure and Appl. Math.*, vol. 125, p. 285–309. Dekker, 1990.

167 E. Kaltofen. Polynomial factorization 1987–1991. In I. Simon (éd.), *LATIN '92 (São Paulo, 1992). Lect. Notes Comput. Sci.*, vol. 583, p. 294–313. Springer-Verlag, 1992.

168 E. Kaltofen. Effective Noether irreducibility forms and applications. *J. Comput. System Sci.*, vol. 50, n°2, 1995, p. 274–295.

169 E. Kaltofen. Challenges of symbolic computation: my favorite open problems. *J. Symbolic Comput.*, vol. 29, n°6, 2000, p. 891–919.

170 E. Kaltofen. Polynomial factorization: a success story. In H. Hong (éd.), *Proc. ISSAC '03*. p. 3–4. ACM, 2003.

171 E. Kaltofen, Y. N. Lakshman. Improved sparse multivariate polynomial interpolation algorithms. In P. Gianni (éd.), *Proc. ISSAC '88*. p. 467–474. ACM, 1988.

172 E. Kaltofen, Y. N. Lakshman, J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In S. Watanabe, M. Nagata (éd.), *Proc. ISSAC '90*. p. 135–139. ACM, 1990.

173 E. Kaltofen, W. Lee, A. A. Lobo. Early termination in Ben-Or/Tiwari sparse interpolation and a hybrid of Zippel's algorithm. In C. Traverso (éd.), *Proc. ISSAC '00*. p. 192–201. ACM, 2000.

174 A. KARATSUBA, J. OFMAN. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, vol. 7, 1963, p. 595–596.

175 P. KIRRINNIS. Newton iteration towards a cluster of polynomial zeros. In F. CUCKER, M. SHUB (éd.), *Foundations of computational mathematics (Rio de Janeiro, 1997)*. p. 193–215. Springer-Verlag, 1997.

176 S. L. KLEIMAN. Bertini and his two fundamental theorems. *Rend. Circ. Mat. Palermo (2) Suppl.*, vol. 55, 1998, p. 9–37. Studies in the history of modern mathematics, III.

177 D. E. KNUTH. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*. Addison Wesley Longman, 1998, 3^e édition.

178 P. KRAVANJA, A. HAEGEMANS. A modification of Newton's method for analytic mappings having multiple zeros. *Computing*, vol. 62, n°2, 1999, p. 129–145.

179 P. KRAVANJA, T. SAKURAI, M. VAN BAREL. On locating clusters of zeros of analytic functions. *BIT*, vol. 39, n°4, 1999, p. 646–682.

180 P. KRAVANJA, M. VAN BAREL. *Computing the zeros of analytic functions*. Springer-Verlag, 2000, *Lect. Notes Math.*, vol. 1727.

181 P. KRAVANJA, M. VAN BAREL, T. SAKURAI. Error analysis of a derivative-free algorithm for computing zeros of holomorphic functions. *Computing*, vol. 70, 2003, p. 335–347.

182 T. KRICK, L. M. PARDO. A computational method for Diophantine approximation. In T. R. L. GONZÁLEZ-VEGA (éd.), *Algorithms in algebraic geometry and applications (Santander, 1994)*, *Progr. Math.*, vol. 143, p. 193–253. 1996.

183 T. KRICK, L. M. PARDO, M. SOMBRA. Sharp estimates for the arithmetic Nullstellensatz. *Duke Math. J.*, vol. 109, n°3, 2001.

184 L. KRONECKER. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *J. reine angew. Math.*, vol. 92, 1882, p. 1–122.

185 P. KUNKEL. A tree-based analysis of a family of augmented systems for the computation of singular points. *IMA J. Numer. Anal.*, vol. 16, n°4, 1996, p. 501–527.

186 R. LEBRETON. Relaxed hensel lifting of triangular sets. 2013. Accepté pour présentation à la conférence MEGA 2013.

187 G. LECERF. Computing an equidimensional decomposition of an algebraic variety by means of geometric resolutions. In C. TRAVERSO (éd.), *Proc. ISSAC '00*. p. 209–216. ACM, 2000.

188 G. LECERF. Quadratic Newton iteration for systems with multiplicity. *Found. Comput. Math.*, vol. 2, n°3, 2002, p. 247–293.

189 G. LECERF. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *J. Complexity*, vol. 19, n°4, 2003, p. 564–596.

190 G. LECERF. Sharp precision in Hensel lifting for bivariate polynomial factorization. *Math. Comp.*, vol. 75, 2006, p. 921–933.

191 G. LECERF. Improved dense multivariate polynomial factorization algorithms. *J. Symbolic Comput.*, vol. 42, n°4, 2007, p. 477–494.

192 G. LECERF. Fast separable factorization and applications. *Appl. Alg. Eng. Comm. Comp.*, vol. 19, n°2, 2008, p. 135–160.

193 G. LECERF. New recombination algorithms for bivariate polynomial factorization based on Hensel lifting. *Appl. Alg. Eng. Comm. Comp.*, vol. 21, n°2, 2010, p. 151–176.

194 G. LECERF. Factorization des polynômes à plusieurs variables (Cours no. II). In G. CHÈZE, P. BOITO, C. PERNET, M. S. EL DIN (éd.), *Journées nationales de calcul formel. Les cours du CIRM*, vol. 3, p. 1–85. Cedram, 2013.

195 G. LECERF, É. SCHOST. Fast multivariate power series multiplication in characteristic zero. *SADIO Electronic Journal on Informatics and Operations Research*, vol. 5, n°1, 2003, p. 1–10.

196 L. LEHMANN. *Wavelet-Konstruktion als Anwendung der algorithmischen reellen algebraischen Geometrie*. Thèse de doctorat, Humboldt-Universität, Berlin, Allemagne, 2007.

197 A. K. LENSTRA, H. W. LENSTRA, JR., L. LOVÁSZ. Factoring polynomials with rational coefficients. *Math. Ann.*, vol. 261, n°4, 1982, p. 515–534.

198 A. LEYKIN. Numerical primary decomposition. In J. R. SENDRA, L. GONZALEZ-VEGA (éd.), *Proc. ISSAC '08*. p. 165–172. ACM, 2008.

199 A. LEYKIN, J. VERSCHELDE, A. ZHAO. Newton's method with deflation for isolated singularities of polynomial systems. *Theoret. Comput. Sci.*, vol. 359, n°1–3, 2006, p. 111 – 122.

200 A. LEYKIN, J. VERSCHELDE, A. ZHAO. Evaluation of jacobian matrices for newton's method with deflation to approximate isolated singular solutions of polynomial systems. In D. WANG, L. ZHI (éd.), *Symbolic-Numeric Computation. Trends in Mathematics*, p. 269–278. Birkhäuser Basel, 2007.

201 A. LEYKIN, J. VERSCHELDE, A. ZHAO. Higher-order deflation for polynomial systems with isolated singular solutions. In A. DICKENSTEIN, F.-O. SCHREYER, A. SOMMESE (éd.), *Algorithms in Algebraic Geometry. The IMA Volumes in Mathematics and its Applications*, vol. 146, p. 79–97. Springer-Verlag, 2008.

202 N. LI, L. ZHI. Computing the multiplicity structure of an isolated singular solution: Case of breadth one. *J. Symbolic Comput.*, vol. 47, n°6, 2012, p. 700–710.

203 K. MAKINO, M. BERZ. Remainder differential algebras and their applications. In M. BERZ (éd.), *Computational differentiation: techniques, applications and tools*. p. 63–74. SIAM, 1996.

204 K. MAKINO, M. BERZ. Suppression of the wrapping effect by Taylor model-based verified integrators: long-term stabilization by preconditioning. *Int. J. Differ. Equ. Appl.*, vol. 10, n°4, 2005, p. 353–384.

205 S. MALLAT. Foveal detection and approximation for singularities. *Appl. Comput. Harmon. Anal.*, vol. 14, n°2, 2003, p. 133–180.

206 M. G. MARINARI, H. M. MÖLLER, T. MORA. On multiplicities in polynomial system solving. *Trans. Amer. Math. Soc.*, vol. 348, n°8, 1996, p. 3283–3321.

207 G. MATERA. Probabilistic algorithms for geometric elimination. *Appl. Algebra Engrg. Comm. Comput.*, vol. 9, n°6, 1999, p. 463–520.

208 H. MATSUMURA. *Commutative ring theory*. Cambridge Univ. Press, 1986, *Cambridge Studies in Advanced Mathematics*, vol. 8. Translated from the Japanese by M. Reid.

209 E. W. MAYR, A. R. MEYER. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, vol. 46, n°3, 1982, p. 305–329.

210 D. MCKINNON. An arithmetic analogue of Bezout's theorem. *Compositio Math.*, vol. 126, n°2, 2001, p. 147–155.

211 Z. MEI. A special extended system and a Newton-like method for simple singular nonlinear equations. *Computing*, vol. 45, n°2, 1990, p. 157–167.

212 R. MINES, F. RICHMAN. Separability and factoring polynomials. *Rocky Mountain J. Math.*, vol. 12, n°1, 1982, p. 43–54.

213 R. MINES, F. RICHMAN, W. RUITENBURG. *A course in constructive algebra*. Springer-Verlag, 1988, *Universitext*.

214 R. T. MOENCK. Practical fast polynomial multiplication. In R. D. JENKS (éd.), *Proc. SYMSAC '76*. p. 136–148. ACM, 1976.

215 M. MONAGAN, R. PEARCE. Polynomial division using dynamic arrays, heaps, and packed exponent vectors. In V. G. GANZHA, E. W. MAYR, E. V. VOROZHTSOV (éd.), *Computer Algebra in Scientific Computing (CASC 2007). Lect. Notes Comput. Sci.*, vol. 4770, p. 295–315. Springer-Verlag, 2007.

216 M. MONAGAN, R. PEARCE. Parallel sparse polynomial multiplication using heaps. In J. JOHNSON, H. PARK, E. KALTOFEN (éd.), *Proc. ISSAC '09*. p. 263–270. ACM, 2009.

217 M. Monagan, R. Pearce. Polynomial multiplication and division in Maple 14. *ACM Commun. Comput. Algebra*, vol. 44, n°4, 2010, p. 205–209.

218 T. Mora. *Solving polynomial equation systems. I The Kronecker-Duval philosophy*. Cambridge Univ. Press, 2003, *Encyclopedia of Mathematics and its Applications*, vol. 88.

219 T. Mora. La queste del saint Gra(AL): A computational approach to local algebra. *Discrete Applied Mathematics*, vol. 33, n°1–3, 1991, p. 161–190.

220 J. E. Morais. *Resolución eficaz de sistemas de ecuaciones polinomiales*. Thèse de doctorat, Universidad de Cantabria, Spain, 1997.

221 A. P. Morgan, A. J. Sommese, C. W. Wampler. Computing singular solutions to polynomial systems. *Adv. in Appl. Math.*, vol. 13, n°3, 1992, p. 305–327.

222 B. Mourrain. Isolated points, duality and residues. *Journal of Pure and Applied Algebra*, vol. 117–118, 1997, p. 469–493.

223 B. Mourrain, *et al.* Axel, algebraic geometric modeler, 2007–2013. http://axel.inria.fr.

224 G. L. Mullen, D. Panario. *Handbook of Finite Fields*. Chapman and Hall/CRC, 2013, *Discrete Mathematics and Its Applications*.

225 D. Mumford. *Algebraic geometry. I Complex projective varieties*. Springer-Verlag, 1995, *Classics in Mathematics*.

226 D. R. Musser. *Algorithms for Polynomial Factorization*. Thèse de doctorat, Univ. of Wisconsin, USA, 1971.

227 D. R. Musser. Multivariate polynomial factorization. *J. Assoc. Comput. Mach.*, vol. 22, 1975, p. 291–308.

228 E. Noether. Ein algebraisches Kriterium für absolute Irreduzibilität. *Math. Ann.*, vol. 85, n°1, 1922, p. 26–40.

229 T. Ojika. Modified deflation algorithm for the solution of singular problems. I. a system of nonlinear algebraic equations. *J. Math. Anal. Appl.*, vol. 123, 1987, p. 199–221.

230 T. Ojika, S. Watanabe, T. Mitsui. Deflation algorithm for the multiple roots of a system of nonlinear equations. *J. Math. Anal. Appl.*, vol. 96, n°2, 1983, p. 463–479.

231 A. M. Ostrowski. *Solution of equations and systems of equations*. Academic Press, 1966.

232 V. Y. Pan. Approximating complex polynomial zeros: modified Weyl's quadtree construction and improved Newton's iteration. *J. Complexity*, vol. 16, n°1, 2000, p. 213–264. Real computation and complexity (Schloss Dagstuhl, 1998).

233 L. M. Pardo. How lower and upper complexity bounds meet in elimination theory. In G. Cohen, M. Giusti, T. Mora (éd.), *Proc. AAECC-11*. *Lect. Notes Comput. Sci.*, vol. 948, p. 33–69. Springer-Verlag, 1995.

234 L. M. Pardo, J. San Martin. Deformation techniques to solve generalised Pham systems. *Theoret. Comput. Sci.*, vol. 315, n°2-3, 2004, p. 593–625.

235 The PARI Group, Bordeaux. *PARI/GP*, 2012. http://pari.math.u-bordeaux.fr.

236 A.-E. Pellet. Sur un mode de séparation des racines des équations et la formule de Lagrange. *Bulletin des Sciences Mathématiques et Astronomiques*, vol. 5, 1881, p. 393–395. Deuxième Série, Tome 16 de la Collection.

237 P. Penfield, Jr., R. Spencer, S. Duinker. *Tellegen's theorem and electrical networks*. The M.I.T. Press, Cambridge, Mass.-London, 1970.

238 C. G. Ponder. Parallel multiplication and powering of polynomials. *J. Symbolic Comput.*, vol. 11, n°4, 1991, p. 307–320.

239 P. J. Rabier, G. W. Reddien. Characterization and computation of singular points with maximum rank deficiency. *SIAM J. Numer. Anal.*, vol. 23, n°5, 1986, p. 1040–1051.

240 L. Rall. Convergence of the Newton process to multiple solutions. *Num. Math.*, vol. 9, 1966, p. 23–37.

241 G. W. Reddien. On Newton's method for singular problems. *SIAM J. Numer. Anal.*, vol. 15, n°5, 1978, p. 993–996.

242 G. W. Reddien. Newton's method and high order singularities. *Comput. Math. Appl.*, vol. 5, n°2, 1979, p. 79–86.

243 J. Renegar. On the worst-case arithmetic complexity of approximating zeros of polynomials. *J. Complexity*, vol. 3, n°2, 1987, p. 90–113.

244 F. Richman. Seidenberg's condition *P*. In F. Richman (éd.), *Constructive mathematics (Las Cruces, N.M., 1980)*. *Lect. Notes Math.*, vol. 873, p. 1–11. Springer-Verlag, 1981.

245 D. S. Roche. Chunky and equal-spaced polynomial multiplication. *J. Symbolic Comput.*, vol. 46, n°7, 2011, p. 791–806.

246 M. Safey El Din, É. Schost. Polar varieties and computation of one point in each connected component of a smooth algebraic set. In H. Hong (éd.), *Proc. ISSAC '03*. p. 224–231. ACM, 2003.

247 M. Safey El Din, É. Schost. Properness defects of projections and computation of at least one point in each connected component of a real algebraic set. *Discrete Comput. Geom.*, vol. 32, n°3, 2004, p. 417–430.

248 T. Sasaki, T. Saito, T. Hilano. Analysis of approximate factorization algorithm. I. *Japan J. Indust. Appl. Math.*, vol. 9, n°3, 1992, p. 351–368.

249 T. Sasaki, M. Sasaki. A unified method for multivariate polynomial factorizations. *Japan J. Indust. Appl. Math.*, vol. 10, n°1, 1993, p. 21–39.

250 T. Sasaki, M. Suzuki, M. Kolář, M. Sasaki. Approximate factorization of multivariate polynomials and absolute irreducibility testing. *Japan J. Indust. Appl. Math.*, vol. 8, n°3, 1991, p. 357–375.

251 A. Schönhage, V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, vol. 7, 1971, p. 281–292.

252 É. Schost. Computing parametric geometric resolutions. *Appl. Algebra Engrg. Comm. Comput.*, vol. 13, n°5, 2003, p. 349–393.

253 É. Schost. Multivariate power series multiplication. In X.-S. Gao, G. Labahn (éd.), *Proc. ISSAC '05*. p. 293–300. ACM, 2005.

254 E. Schröder. Über unendlich viele Algorithmen zur Auflösung der Gleichungen. *Math. Annalen*, vol. 2, 1870, p. 317–365.

255 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, vol. 27, n°4, 1980, p. 701–717.

256 A. Seidenberg. Construction of the integral closure of a finite integral domain. *Rend. Sem. Mat. Fis. Milano*, vol. 40, 1970, p. 100–120.

257 A. Seidenberg. Constructions in algebra. *Trans. Amer. Math. Soc.*, vol. 197, 1974, p. 273–313.

258 A. Seidenberg. Constructions in a polynomial ring over the ring of integers. *Amer. J. Math.*, vol. 100, n°4, 1978, p. 685–703.

259 I. R. Shafarevich. *Basic algebraic geometry. 1 Varieties in projective space*. Springer-Verlag, 1994, 2^e édition.

260 V. Shoup. A fast deterministic algorithm for factoring polynomials over finite fields of small characteristic. In S. M. Watt (éd.), *Proc. ISSAC'91*. p. 14–21. ACM, 1991.

261 V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, vol. 20, n°4, 1995, p. 363–397.

262 V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In J. Johnson, H. Park, E. Kaltofen (éd.), *Proc. ISSAC '99*. p. 53–58. ACM, 1999.

263 V. Shoup. *NTL: A Library for doing Number Theory*, 2013. Software, version 6.0, http://www.shoup.net/ntl/.

264 M. SHUB, S. SMALE. Computational complexity. On the geometry of polynomials and a theory of cost. I. *Ann. Sci. École Norm. Sup. (4)*, vol. 18, n°1, 1985, p. 107–142.

265 M. SHUB, S. SMALE. Computational complexity: on the geometry of polynomials and a theory of cost. II. *SIAM J. Comput.*, vol. 15, n°1, 1986, p. 145–161.

266 M. F. SINGER, F. ULMER. Linear differential equations and products of linear forms. *J. Pure Appl. Algebra*, vol. 117/118, 1997, p. 549–563.

267 S. SMALE. The fundamental theorem of algebra and complexity theory. *Bull. Amer. Math. Soc. (N.S.)*, vol. 4, n°1, 1981, p. 1–36.

268 S. SMALE. Newton method estimates from data at one point. In R. E. EWING, K. I. GROSS, C. F. MARTIN (éd.), *In The Merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics*. p. 185–196. Springer-Verlag, 1986.

269 A. J. SOMMESE, J. VERSCHELDE. Numerical homotopies to compute generic points on positive dimensional algebraic sets. *J. Complexity*, vol. 16, n°3, 2000, p. 572–602.

270 A. J. SOMMESE, J. VERSCHELDE, C. W. WAMPLER. A method for tracking singular paths with application to the numerical irreducible decomposition. In P. FRANCIA, F. CATANESE, C. CILIBERTO, A. LANTERI, C. PEDRINI, M. BELTRAMETTI (éd.), *Algebraic geometry*. p. 329–345. de Gruyter, Berlin, 2002.

271 A. J. SOMMESE, J. VERSCHELDE, C. W. WAMPLER. Advances in polynomial continuation for solving problems in kinematics. *ASME J. Mech. Design*, vol. 126, n°2, 2004, p. 262–268.

272 A. J. SOMMESE, J. VERSCHELDE, C. W. WAMPLER. Solving polynomial systems equation by equation. In A. DICKENSTEIN, F.-O. SCHREYER, A. SOMMESE (éd.), *Algorithms in Algebraic Geometry. The IMA Volumes in Mathematics and its Applications*, vol. 146, p. 133–152. Springer-Verlag, 2008.

273 A. J. SOMMESE, C. W. WAMPLER. Numerical algebraic geometry. In *The mathematics of numerical analysis (Park City, UT, 1995). Lectures in Appl. Math.*, vol. 32, p. 749–763. Amer. Math. Soc., Providence, RI, 1996.

274 A. STEEL. Conquering inseparability: primary decomposition and multivariate factorization over algebraic function fields of positive characteristic. *J. Symbolic Comput.*, vol. 40, n°3, 2005, p. 1053–1075.

275 D. STEHLE, *et al.* fplll: software for LLL-reduction of euclidean lattices and for solving the shortest vector problem, 2009–2013. http://perso.ens-lyon.fr/damien.stehle/fplll.

276 W. A. STEIN, *et al. Sage Mathematics Software*. The Sage Development Team, 2004–2013. http://www.sagemath.org.

277 A. STORJOHANN. *Algorithms for matrix canonical forms*. Thèse de doctorat, ETH, Zürich, Switzerland, 2000.

278 D. R. STOUTEMYER. Which polynomial representation is best? In *Proc. 1984 MACSYMA Users' Conference: Schenectady, New York*. p. 221–243. General Electric, 1984.

279 B. TELLEGEN. A general network theorem, with applications. *Philips Research Reports*, vol. 7, 1952, p. 259–269.

280 A. TERUI, T. SASAKI. "Approximate zero-points" of real univariate polynomial with large error terms. *IPSJ Journal*, vol. 41, n°4, avril 2000, p. 974–989.

281 A. L. TOOM. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics*, vol. 4, n°2, 1963, p. 714–716.

282 B. M. TRAGER. *Integration of algebraic functions*. Thèse de doctorat, M.I.T., USA, 1984.

283 T. TSUCHIYA. Enlargement procedure for resolution of singularities at simple singular solutions of nonlinear equations. *Numer. Math.*, vol. 52, n°4, 1988, p. 401–411.

284 H. VAN DE VEL. A method for computing a root of a single nonlinear equation, including its multiplicity. *Computing*, vol. 14, n°1-2, 1975, p. 167–171.

285 M. VANDER STRAETEN, H. VAN DE VEL. Multiple root-finding methods. *J. Comput. Appl. Math.*, vol. 40, n°1, 1992, p. 105–114.

286 H. D. VICTORY, JR., B. NETA. A higher order method for multiple zeros of nonlinear functions. *Internat. J. Comput. Math.*, vol. 12, n°3-4, 1982/83, p. 329–335.

287 B. L. VAN DER WAERDEN. Eine Bemerkung über die Unzerlegbarkeit von Polynomen. *Math. Ann.*, vol. 102, n°1, 1930, p. 738–739.

288 B. L. VAN DER WAERDEN. *Modern Algebra. Vol. I*. Frederick Ungar Publishing Co., New York, N. Y., 1949.

289 D. WANG. *Elimination practice. Software tools and applications*. Imperial College Press, London, 2004.

290 P. S. WANG. An improved multivariate polynomial factoring algorithm. *Math. Comp.*, vol. 32, n°144, 1978, p. 1215–1231.

291 P. S. WANG, L. P. ROTHSCHILD. Factoring multivariate polynomials over the integers. *Math. Comp.*, vol. 29, 1975, p. 935–950.

292 X. H. WANG, D. F. HAN. On dominating sequence method in the point estimate and Smale theorem. *Sci. China, Series A*, vol. 33, n°2, 1990, p. 135–144.

293 H. WEBER, W. WERNER. On the accurate determination of nonisolated solutions of nonlinear equations. *Computing*, vol. 26, n°4, 1981, p. 315–326.

294 M. WEIMANN. A lifting and recombination algorithm for rational factorization of sparse polynomials. *J. Complexity*, vol. 26, n°6, 2010, p. 608–628.

295 M. WEIMANN. Algebraic osculation and application to factorization of sparse polynomials. *Found. Comput. Math.*, vol. 12, n°2, 2012, p. 173–201.

296 K. WERTHER. The complexity of sparse polynomial interpolation over finite fields. *Appl. Algebra Engrg. Comm. Comput.*, vol. 5, n°2, 1994, p. 91–103.

297 WOLFRAM RESEARCH. Mathematica, 1988. http://www.wolfram.com/mathematica.

298 X. WU, L. ZHI. Computing the multiplicity structure from geometric involutive form. In L. G.-V. J. RAFAEL SENDRA (éd.), *Proc. ISSAC '08*. p. 325–332. ACM, 2008.

299 J.-C. YAKOUBSOHN. Finding a cluster of zeros of univariate polynomials. *J. Complexity*, vol. 16, n°3, 2000, p. 603–638. Complexity theory, real machines, and homotopy (Oxford, 1999).

300 J.-C. YAKOUBSOHN. Simultaneous computation of all the zero-clusters of a univariate polynomial. In F. CUCKER, M. ROJAS (éd.), *Foundations of computational mathematics (Hong Kong, 2000)*. p. 433–455. World Sci. Publishing, River Edge, NJ, 2002.

301 N. YAMAMOTO. Regularization of solutions of nonlinear equations with singular Jacobian matrices. *J. Inform. Process.*, vol. 7, n°1, 1984, p. 16–21.

302 T. YAN. The geobucket data structure for polynomials. *J. Symbolic Comput.*, vol. 25, n°3, 1998, p. 285–293.

303 T. J. YPMA. Finding a multiple zero by transformations and Newton-like methods. *SIAM Rev.*, vol. 25, n°3, 1983, p. 365–378.

304 H. ZASSENHAUS. On Hensel factorization I. *J. Number Theory*, vol. 1, n°1, 1969, p. 291–311.

305 Z. ZENG, B. H. DAYTON. The approximate GCD of inexact polynomials. In J. SCHICHO (éd.), *Proc. ISSAC '04*. p. 320–327. ACM, 2004.

306 R. ZIPPEL. Probabilistic algorithms for sparse polynomials. In E. W. NG (éd.), *Proc. EUROSAM' 79*. *Lect. Notes Comput. Sci.*, p. 216–226. Springer-Verlag, 1979.

307 R. ZIPPEL. Newton's iteration and the sparse Hensel algorithm (Extended Abstract). In P. S. WANG (éd.), *Proc. SYMSAC '81*. p. 68–72. ACM, 1981.

308 R. ZIPPEL. Interpolating polynomials from their values. *J. Symbolic Comput.*, vol. 9, n°3, 1990, p. 375–403.

309 R. ZIPPEL. *Effective Polynomial Computation*. Kluwer Academic Publishers, 1993.