8. une présentation des premiers algorithmes quasi-linéaires qui calculent une représentation à une variable de l'algèbre de décomposition universelle sur les corps finis d'une part et le polynôme caractéristique de ses éléments d'autre part ;
9. enfin, une preuve qu'il suffit, pour calculer des invariants fondamentaux sur les rationnels, de les calculer modulo p.

Publications Les contributions 3, 4 et 6 ont été publiées dans les actes de la conférence *ISSAC'12* avec J. Berthomieu [BL12]. Leur présentation dans ce mémoire contient des détails, preuves et exemples additionnels. De plus, l'application du solveur détendu de systèmes linéaires du point 4 aux matrices structurées est un travail en cours, fait en commun avec É. Schost. La contribution du point 5 provient d'un article écrit avec A. Bostan, M. Chowdhurry, B. Salvy et É. Schost qui est publié dans les actes de *ISSAC'12* [BCL+12]. Dernièrement, les contributions de l'item 8 viennent d'un travail avec É. Schost et publié dans les actes de *ISSAC'12* [LS12].

Prix Le prix du meilleur article étudiant m'a été décerné à la conférence *ISSAC'12* pour l'article [LS12]. J'ai également reçu à cette conférence le prix du meilleur poster décerné par le « Fachgruppe Computer Algebra » pour le poster [LMS12] en collaboration avec E. Mehrabi et É. Schost.

Algorithmes détendus pour la multiplication

Cette section présente la notion d'algorithmes en-ligne et détendus sur des anneaux p-adiques généraux. Soit R un anneau commutatif avec unité. Étant donné un idéal principal propre (p) avec $p \in R$, nous notons R_p la complétion de l'anneau R pour la valuation p-adique. Les éléments $a \in R_p$ sont appelés des p-adiques. Pour avoir l'unicité de la décomposition des éléments de R_p, fixons un sous-ensemble M de R tel que la projection $\pi : M \to R/(p)$ soit une bijection. Alors, tout p-adique $a \in R_p$ s'écrit de manière unique $a = \sum_{i \in \mathbb{N}} a_i p^i$ avec $a_i \in M$.

Deux exemples classiques sont l'anneau des séries formelles $\Bbbk[[X]]$ qui est le complété de l'anneau des polynômes $\Bbbk[X]$ pour l'idéal (X), et l'anneau des entiers p-adiques $\mathbb{Z}_p$ qui est le complété de l'anneau des entiers $\mathbb{Z}$ pour l'idéal (p). Dans le cas, $R = \mathbb{Z}$, nous prendrons $M = \{-(p-1)/2, \ldots, (p-1)/2\}$ si $p \neq 2$ et $M = \{0, 1\}$ si $p = 2$. Pour $R = \Bbbk[X]$, nous prendrons $M = \Bbbk$.

Nous sommes maintenant en mesure de donner la définition d'un algorithme détendu telle qu'elle fut introduite par [Hen66].

Définition 1. ([Hen66, FS74]) *Soit une machine de Turing qui calcule une fonction f sur des suites, avec $f : \Sigma^* \times \Sigma^* \to \Delta^*$ et Σ et Δ des ensembles. On dit que la machine calcule f* en-ligne *si, pour toutes suites $a = a_0 a_1 \ldots a_n$, $b = b_0 b_1 \ldots b_n$ en entrée et pour des sorties correspondantes $f(a, b) = c_0 c_1 \ldots c_n$, avec $a_i, b_j \in \Sigma$, $c_k \in \Delta$, la machine écrit la sortie c_k avant de lire soit a_j, soit b_j, avec $0 \leqslant k < j \leqslant n$.*

La machine calcule f semi-en-ligne *(par rapport au premier argument) si elle écrit la sortie c_k avant de lire a_j pour $0 \leqslant k < j \leqslant n$. L'entrée a est appelée l'entrée* en-ligne *et b l'entrée* hors-ligne.

Le reste de ce chapitre traite des algorithmes en-ligne rapides pour la multiplication de p-adiques. Ces algorithmes sont faits d'appels à des multiplications hors-ligne de p-adiques de précision finie. Nous rappelons d'abord l'état de l'art des algorithmes de multiplication de polynômes et d'entiers qui sont respectivement les séries formelles et les entiers p-adiques de précision finie. Nous regarderons aussi les algorithmes existants pour les produits court et médian.

Avec ces notions à portée de main, nous faisons un rappel des algorithmes détendus suivants de multiplication de p-adiques. Le premier algorithme en-ligne quasi-optimal de multiplication fut présenté dans [FS74] pour les entiers, puis dans [Sch97] pour les nombres réels et finalement dans [Hoe97] pour les séries formelles. Ce dernier algorithme fut adapté en un algorithme semi-détendu (ou semi-en-ligne) dans [FS74, Hoe03]. Un première amélioration d'un facteur constant du produit semi-détendu est présenté dans [Hoe03].

Notre contribution consiste en la présentation d'un nouvel algorithme détendu utilisant des produits *médians* et *courts* qui améliore d'un facteur constant les algorithmes précédents. De plus, nous analysons pour la première fois précisément le nombre de multiplications de base que font ces algorithmes détendus. Pour finir, nous donnons des temps de calcul qui confirment le bon comportement des algorithmes détendus qui utilisent le produit médian.

À partir de maintenant, nous utiliserons les notations suivantes. Pour tout p-adique $a = \sum_{i\in\mathbb{N}} a_i\, p^i$, la longueur $\lambda(a)$ de a est définie par $\lambda(a) := 1 + \sup(i \in \mathbb{N} \mid a_i \neq 0)$ si $a \neq 0$ et $\lambda(0) = 0$. Le coût de la multiplication de deux p-adiques de longueur N par un algorithme hors-ligne (resp. en-ligne) est noté $\mathsf{I}(N)$ (resp. $\mathsf{R}(N)$) dans notre modèle de complexité précisé en Section 1.1.2. Aussi nous noterons $\mathsf{M}(N)$ le nombre d'opérations arithmétiques que nécessite la multiplication de deux polynômes de longueur N.

2 Nombres p-adiques récursifs

L'étude des algorithmes en-ligne est motivée par l'implémentation pratique des p-adiques récursifs à l'aide de ces algorithmes. Il fut pointé pour la première fois dans [Wat89] que le calcul de séries formelles récursives est bien adapté à l'algorithmique paresseuse. Cela donna lieu à l'époque à un cadre très pratique pour calculer avec les séries formelles récursives, mais pas encore très efficace.

Ce fait fut redécouvert par [Hoe02] plus généralement pour l'algorithmique en-ligne. Mis bout à bout avec l'algorithme en-ligne rapide de multiplication de [Hoe97], van der Hoeven obtint un cadre simple et efficace pour calculer avec les séries formelles récursives.

Un p-adique récursif y est un p-adique qui satisfait $y = \Phi(y)$ pour un opérateur Φ qui vérifie l'égalité entre les n-ièmes coefficients p-adiques $\Phi(y)_n = \Phi(y + a\, p^n)_n$ pour tout $a \in R_p$. Par conséquent, y est uniquement déterminé par la donnée de Φ et de son premier coefficient y_0.

Dans ce chapitre, nous rappelons la méthode de [Hoe02] qui, à partir d'un algorithme en-ligne qui évalue la fonction Φ, calcule les coefficients du p-adique récursif y l'un après l'autre. Cependant cette méthode ne marche pas toujours tel quel.

Notre contribution est d'identifier le problème sous-jacent et de donner une condition suffisante pour que la méthode précédente marche. Nous n'avons pas connaissance de traces de ce problème dans la littérature.

Nous introduisons la nouvelle notion *d'algorithmes décalés* dans l'optique de résoudre ce problème. Un entier, appelé le décalage, est associé à toute entrée p-adique d'un algorithme donné et mesure quels coefficients de ces entrées sont lus au moment où l'algorithme produit le n-ième coefficient de la sortie. À titre d'exemple, un algorithme est en-ligne si et seulement si son décalage est positif.

Finalement un algorithme est un *algorithme décalé* si son décalage est positif. Nous avons alors à notre disposition les outils nécessaires pour énoncer la proposition fondamentale suivante.

Proposition. *Soient y un p-adique récursif et Ψ un algorithme décalé tels que $y = \Psi(y)$. Alors le p-adique y peut être calculé à précision N dans le temps nécessaire pour évaluer Ψ en y à précision N.*

Ainsi, le coût du calcul d'un p-adique récursif est le même que le coût de sa vérification. La proposition précédente est la pierre angulaire des futures estimations de complexité liées à des p-adiques récursifs. Elle sera utilisée dans les chapitres 3, 4, 5 et 6.

3 Algèbre linéaire sur les p-adiques

Dans ce chapitre, nous présentons un algorithme basé sur le cadre détendu pour les p-adiques récursifs du chapitre 2 qui peut en principe être appliqué à n'importe quel choix de représentation de matrices (dense, creuse, structurée, *etc.*). Nous nous concentrons sur les deux cas particuliers importants que sont les matrices *denses* et *structurées* et nous montrons comment notre algorithme peut améliorer les techniques existantes.

Considérons un système linéaire de la forme $A = B \cdot C$, avec A et B connues et C inconnue. La matrice A appartient à $\mathcal{M}_{r\times s}(R_p)$ et la matrice $B \in \mathcal{M}_{r\times r}(R_p)$ est inversible ; nous résolvons le système linéaire $A = B \cdot C$ en $C \in \mathcal{M}_{r\times s}(R_p)$. Les cas les plus intéressants sont le cas $s = 1$, qui revient à résoudre un système linéaire, et $s = r$, qui contient le problème de l'inversion de B. Une application importante de la résolution linéaire de systèmes à coefficients p-adiques est la résolution de systèmes sur R (c'est-à-dire sur les entiers ou les polynômes par exemple) à l'aide de techniques de remontée.

Nous noterons $d := \max(\lambda(A), \lambda(B))$ la longueur maximale des coefficients des matrices A et B. Soit N la précision à laquelle nous voulons connaître C. Ainsi, nous pourrons toujours supposer que $d \leqslant N$. Le cas particulier $N = d$ correspond à la résolution de systèmes linéaires p-adiques en propre, tandis que la résolution de systèmes linéaires sur R nécessite souvent une précision $N \gg d$. En effet, dans ce cas et pour les séries formelles par exemple, les formules de Cramer indiquent que les numérateurs et les dénominateurs de C ont une longueur $\mathcal{O}(r\,d)$, de telle sorte que l'on doive prendre N de l'ordre de $\mathcal{O}(r\ d)$ pour permettre la reconstruction rationnelle.

Parmi les méthodes préexistantes, un premier algorithme dû à Dixon [Dix82] calcule un à un les coefficients p-adiques de la solution C puis met à jour la matrice A. De l'autre côté de l'étendue des techniques, on trouve l'itération de Newton qui double la précision de la solution à chaque étape. L'algorithme de Moenck-Carter [MC79] est une variante de l'algorithme de Dixon qui travaille avec des p^d-adiques au lieu de p-adiques. Finalement, l'algorithme de Storjohann de remontée à grande précision [Sto03] peut être vu comme une version rapide de l'algorithme de Moenck-Carter, adapté aux cas $N \gg d$.

Nous contribuons avec un algorithme qui résout des systèmes linéaires par des techniques détendues. Cette algorithme est obtenu en prouvant que les coefficients de la solution $C = B^{-1} \cdot A$ sont des p-adiques récursifs. En d'autres termes, nous montrons que C est le point fixe d'un certain opérateur décalé.

Si l'on prend par exemple $s = 1$, le calcul de C à précision N coûte avec notre algorithme à peu près :

1. une inversion $\Gamma := B^{-1}$ modulo (p) ;
2. $\mathcal{O}(N)$ produits matrice-vecteur utilisant l'inverse Γ où chacune des entrées du vecteur est aussi de longueur 1 ;
3. $\mathcal{O}(1)$ produits matrice-vecteur utilisant B, avec un vecteur dont les entrées sont des p-adiques détendus.

Nous verrons que notre algorithme est un intermédiaire entre les algorithmes de Dixon et de Moenck-Carter puisque que nous faisons la même peu coûteuse initialisation modulo (p) que Dixon (*cf.* item 1) tout en gardant la même bonne complexité asymptotique que Moenck-Carter (*cf.* item 3).

Le tableau suivant donne les complexités des algorithmes présentés pour le cas des matrices denses, avec $R_p = \Bbbk[[X]]$, $s = 1$ et les deux valeurs de N importantes en pratique, c'est-à-dire $N = d$ et $N = r\,d$. Il en ressort que pour la résolution à précision $N = d$, notre algorithme est le plus rapide de ceux présentés. En précision $N = r\,d$, la remontée à grande précision de Storjohann est meilleure (car elle est conçue pour de telles précisions). Soit $\omega \in \mathbb{R}_{>0}$ tel que l'on peut multiplier et inverser toute matrice $r \times r$ en $\mathcal{O}(r^\omega)$ opérations arithmétiques sur tout corps.

Algorithme	$N = d$	$N = r\,d$
Dixon	$\tilde{\mathcal{O}}(r^\omega + r^2\,d^2)$	$\tilde{\mathcal{O}}(r^3\,d^2)$
Moenck-Carter	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^3\,d)$
Itération de Newton	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^{\omega+1}\,d)$
Storjohann	$\tilde{\mathcal{O}}(r^\omega\,d)$	$\tilde{\mathcal{O}}(r^\omega\,d)$
Notre algorithme	$\tilde{\mathcal{O}}(r^\omega + r^2\,d)$	$\tilde{\mathcal{O}}(r^3\,d)$

Tableau. Coût simplifié de la résolution de $A = B \cdot C$ pour des matrices denses sur $\Bbbk[[X]]$ avec $s = 1$.

Ensuite, nous analysons la situation pour les matrices structurées. Brièvement, dans la représentation des matrices structurées, un entier α, nommé le rang (de déplacement), est associé à toute matrice $A \in \mathcal{M}_{r\times r}(R_p)$. Les deux principales caractéristiques de la matrice structurée A est que A peut être stockée en mémoire par une structure de donnée compacte en taille $\mathcal{O}(\alpha\,r)$ et que le produit matrice-

vecteur $A \cdot V$ coûte $\mathcal{O}(\alpha\,\mathsf{M}(r))$ pour tout vecteur $V \in \mathcal{M}_{r\times 1}(R_p)$. Nous renvoyons à [Pan01] pour une présentation complète.

Le tableau qui suit rappelle les résultats de complexité connus pour la résolution de systèmes linéaires structurés et donne le temps de calcul de notre algorithme. Ici, d' désigne la longueur des entrées p-adiques de la structure de donnée compacte qui sert à stocker A. La précision voulue est toujours notée N. Comme précédemment, nous donnons des complexités simplifiées pour les séries formelles dans le cas $s=1$ et $N=d'$ ou $N=r\,d'$.

Algorithme	$N=d'$	$N=r\,d'$
Dixon	$\tilde{\mathcal{O}}\big(\alpha^2 r+\alpha\,r\,d'^2\big)$	$\tilde{\mathcal{O}}\big(\alpha\,r^2\,d'^2\big)$
Moenck-Carter	$\tilde{\mathcal{O}}(\alpha^2\,r\,d')$	$\tilde{\mathcal{O}}(\alpha\,r^2\,d')$
Itération de Newton	$\tilde{\mathcal{O}}(\alpha^2\,r\,d')$	$\tilde{\mathcal{O}}(\alpha^2\,r^2\,d')$
Notre algorithme	$\tilde{\mathcal{O}}(\alpha^2\,r+\alpha\,r\,d')$	$\tilde{\mathcal{O}}(\alpha\,r^2\,d')$

Tableau. Coût simplifié de la résolution de $A=B\cdot C$ pour des matrices structurées sur $\Bbbk[[X]]$ avec $s=1$.

Notre algorithme est le plus rapide pour des matrices structurées dans les deux cas $N=d$ et $N=r\,d$. Remarquons que l'algorithme de Moenck-Carter est aussi rapide dans le second cas.

Pour finir, nous implémentons ces algorithmes et comparons les temps de calcul pour la représentation dense. Notre implémentation est disponible dans la bibliothèque C++ nommée ALGEBRAMIX incluse dans MATHEMAGIX [HLM+02]. Comme application, nous résolvons des systèmes linéaires sur les entiers et nous nous comparons aux logiciels LINBOX et IML. Nous montrons que nous améliorons les temps pour les petites matrices à coefficients de grands entiers.

4 Séries solutions d'équations (q)-différentielles

Le but de ce chapitre est de fournir des algorithmes qui calculent les séries solutions d'une large famille d'équations, ou de systèmes, différentiels ou aux q-différences. Le nombre d'opérations arithmétiques est linéaire en la précision, à des facteurs logarithmiques près.

Nous nous concentrons sur le cas particulier des équations linéaires, puisque dans de nombreux cas une linéarisation est possible [BCO+07]. Quand l'ordre n de l'équation est strictement plus grand que 1, nous utilisons la technique classique qui transforme ces équations en des équations du premier ordre sur des vecteurs, et nous nous intéresserons ainsi à des équations de la forme

$$x^k\,\delta(F)=A\,F+C, \tag{1}$$

où A est une matrice $n\times n$ sur l'anneau des séries formelles $\Bbbk[[x]]$ ($\Bbbk$ étant le corps des coefficients), C et l'inconnue F sont des vecteurs de taille n sur $\Bbbk[[x]]$ et δ désigne temporairement l'opérateur différentiel d/dx. L'exposant k de (1) est un entier positif qui joue un rôle clé dans cette équation.

Par *résoudre* une telle équation, nous entendons calculer un vecteur F de séries formelles tel que (1) soit vrai modulo x^N. À cet effet, il est nécessaire de calculer F que comme un polynôme de degré au plus N (quand $k=0$) ou $N-1$ (autrement). Réciproquement, quand (1) a une solution série, ses premiers N coefficients peuvent être calculés en résolvant (1) modulo x^N (quand $k\neq 0$) ou x^{N-1} (autrement).

Si $k=0$ et le corps $\Bbbk$ a caractéristique 0, alors un théorème de Cauchy formel s'applique et (1) a un unique vecteur de solutions séries pour une condition initiale donnée. Dans ce cas, des algorithmes existent pour calculer les N premiers termes de la solution en complexité quasi-linéaire : [BK78] pour les équations scalaires d'ordre 1 ou 2, adapté dans [BCO+07] pour les systèmes d'équations. Les algorithmes détendus de [Hoe02] s'appliquent aussi à ce cadre.

Dans ce chapitre, nous étendons les algorithmes précédents dans trois directions.

Singularités Nous traitons le cas où k est strictement positif. Le théorème de Cauchy et les techniques de [BCO+07] ne sont pas applicables. Nous montrons dans ce chapitre comment dépasser ce comportement singulier et obtenir une complexité quasi-linéaire.

Caractéristique positive Même dans le cas $k=0$, le théorème de Cauchy ne s'applique pas en caractéristique positive et l'équation (1) peut ne pas avoir de solutions séries (un exemple simple est $F'=F$). Cependant, une telle équation peut tout de même avoir une solution modulo x^N. Nos objectifs à cet égard sont de surpasser le manque de théorème de Cauchy, ou d'une théorie formelle des équations singulières, en donnant des conditions suffisantes pour assurer l'existence de solutions à la précision demandée.

Équations fonctionnelles Les équations linéaires différentielles ou aux (q)-différences se résolvent par des algorithmes similaires. Pour ceci, introduisons un morphisme d'anneau unitaire $\sigma\colon \Bbbk[[x]]\to\Bbbk[[x]]$ et une σ-dérivation $\delta\colon \Bbbk[[x]]\to\Bbbk[[x]]$, en ce sens que δ est $\Bbbk$-linéaire et que pour tout f,g dans $\Bbbk[[x]]$, on a

$$\delta(fg)=f\delta(g)+\delta(f)\,\sigma(g).$$

Ces définitions, et l'égalité ci-dessus, s'étendent aux matrices sur $\Bbbk[[x]]$. Ainsi, notre objectif est de résoudre la généralisation suivante de (1) :

$$x^k\,\delta(F)=A\,\sigma(F)+C. \tag{2}$$

Comme auparavant, nous sommes intéressés par le fait de calculer un vecteur F de séries tel que (2) soit vrai modulo x^N.

À propos des algorithmes détendus, les techniques de [Hoe02] s'appliquent déjà en caractéristique positive. Au commencement de ma thèse, les outils pour adapter les algorithmes détendus au cas des équations singulières n'existaient pas. Notre méthode pour traiter le cas des équations singulières a été découverte indépendamment à la même époque par [Hoe11]. Ce dernier article traite d'équations récursives plus générales comme les équations algébriques, différentielles ou une combinaison des deux. Cependant, ce même article ne traite pas le cas des équations (q)-différentielles et travaille sous des hypothèses plus restrictives.

Nous nous limitons à la situation suivante :

$$\delta(x)=1,\qquad \sigma\colon x\mapsto q\,x,$$

pour un élément $q \in \Bbbk \setminus \{0\}$. Il n'y a alors que deux possibilités :

- $q = 1$ et $\delta\colon f \mapsto f'$ (cas *différentiel*) ;
- $q \neq 1$ et $\delta\colon f \mapsto \frac{f(qx) - f(x)}{x(q-1)}$ (cas (q)-*différentiel*).

En voyant l'équation (2) comme un système linéaire, on peut résoudre l'équation en utilisant des méthodes d'algèbre linéaire en dimension $n\,N$. Bien que cette solution soit toujours valable, nous donnons des algorithmes de bien meilleure complexité, sous des hypothèses de *bon spectre* liées au spectre $\operatorname{Spec} A_0$ du coefficient p-adique constant de A.

Similairement à l'article [BCO+07] pour le cas non-singulier, nous développons deux approches. La première est une méthode diviser-pour-régner. Le problème est d'abord résolu à précision $N/2$ puis le calcul à précision N est complété en résolvant un problème du même type à précision $N/2$. Cela nous donne le résultat suivant.

Théorème. *Il est possible de calculer des générateurs de l'espace des solutions de l'équation (2) à précision N par une approche diviser-pour-régner. En supposant que A_0 a un bon spectre à précision N, cela peut être fait en temps $\mathcal{O}(n^\omega\,\mathsf{M}(N)\log(N))$. Si l'on a soit $k>1$ soit $k=1$ et $q^i\,A_0 - \gamma_i\,\mathrm{Id}$ inversible pour tout $0 \leq i < N$, le temps de calcul tombe à $\mathcal{O}(n^2\,\mathsf{M}(N)\log(N) + n^\omega\,N)$.*

Cette approche diviser-pour-régner coïncide avec le calcul détendu d'un p-adique récursif sous le deuxième ensemble d'hypothèses, c'est-à-dire soit $k>1$ soit $k=1$ et $q^i\,A_0 - \gamma_i\,\mathrm{Id}$ inversible pour $0 \leq i < N$. Nous prenons le parti de le présenter comme un algorithme diviser-pour-régner car cela permettra de traiter plus facilement le problème sous des hypothèses plus générales.

Notre deuxième algorithme se comporte mieux par rapport à N, avec un coût de seulement $\mathcal{O}(\mathsf{M}(N))$, mais il nécessite des multiplications de matrices polynomiales. Comme dans de nombreux cas, l'approche diviser-pour-régner évite de telles multiplications, le deuxième algorithme est à préférer pour des précisions relativement grandes.

Dans le cas différentiel, quand $k=0$ et que la caractéristique est 0, les algorithmes de [BCO+07, BK78] calculent une matrice inversible de solutions séries de l'équation homogène par une itération de Newton puis en déduisent une solution du problème initial par la méthode de la variation de la constante. Dans le contexte plus général que nous considérons ici, une telle matrice n'existe pas. Cependant, il se trouve qu'une équation associée dérivée de (2) admet une telle solution. Nous utilisons alors une variante de l'itération de Newton pour résoudre cette équation et obtenons le résultat suivant.

Théorème. *En supposant que A_0 a un bon spectre à précision N, il est possible de calculer des générateurs de l'espace des solutions de l'équation (2) à précision N par une itération semblable à celle de Newton en temps $\mathcal{O}(n^\omega\,\mathsf{M}(N) + n^\omega \log(n)\,N)$.*

À notre connaissance, c'est la première fois qu'une complexité aussi basse est atteinte pour ce problème. Cependant, si l'on retire l'hypothèse de bon spectre, nous ne pouvons plus garantir que l'algorithme est correct, et encore moins contrôler sa complexité.

5 Remontée détendue pour les systèmes algébriques

Ce chapitre peut être vu comme un cas particulier de la remontée d'ensemble triangulaire faite dans le chapitre 6.

Supposons qu'il nous est donné un système polynomial $\boldsymbol{P} = (P_1, ..., P_r)$ dans $R[Y_1, ..., Y_r]$ ainsi que $\boldsymbol{y}_0 \in (R/(p))^r$ tel que $\boldsymbol{P}(\boldsymbol{y}_0) = 0$ dans $(R/(p))^r$. Nous travaillerons sous l'hypothèse du Lemme de Hensel qui requiert que la matrice jacobienne $\mathrm{Jac}_{\boldsymbol{P}}(\boldsymbol{y}_0)$ soit inversible. En conséquence, il existe une unique racine $\boldsymbol{y} \in R_p^r$ de $\boldsymbol{P}$ qui se réduise à $\boldsymbol{y}_0$ modulo p.

Notre travail consiste à transformer ces équations implicites en des équations récursives. Alors, nous pourrons utiliser le cadre des p-adiques récursifs détendus pour remonter une racine résiduelle $\boldsymbol{y}_0 \in (R/(p))^r$ en la racine $\boldsymbol{y} \in (R_p)^r$ de $\boldsymbol{P}$ sur l'anneau des p-adiques. Nos résultats sur la transformation d'équations implicites en équations récursives furent découverts en même temps par [Hoe11].

Par souci de clarté, nous ne donnons ici que les complexités asymptotiques quand la précision N des p-adiques tend vers l'infini. Par conséquent, nous noterons $f(n, L, d, N) = \mathcal{O}_{N\to\infty}(g(n, L, d, N))$ s'il existe $K_{n,L,d} \in \mathbb{R}_{\geqslant 0}$ tel que pour tout $N \in \mathbb{N}$, on ait $f(n, L, d, N) \leqslant K_{n,L,d}\, g(n, L, d, N)$.

Commençons par énoncer le résultat pour les polynômes denses à une variable.

Proposition. *Étant donné un polynôme P de degré d en représentation dense et une racine simple modulaire y_0 de P, la remontée de la racine y_0 à précision N dans R_p coûte $(d-1)\,\mathsf{R}(N) + \mathcal{O}_{N\to\infty}(N)$.*

En comparaison, l'itération de Newton remonte y à précision N en temps $(3\,d + 4)\,\mathsf{I}(N) + \mathcal{O}_{N\to\infty}(N)$ (*cf.* [GG03, Théorème 9.25]). Le premier avantage de notre algorithme en-ligne est qu'il fait moins de multiplications en-ligne que l'algorithme hors-ligne ne fait de multiplications hors-ligne. Du coup, nous pouvons espérer des temps de calcul meilleurs pour l'algorithme en-ligne quand la précision N est telle que $\mathsf{R}(N) \leqslant 3\,\mathsf{I}(N)$.

Traitons maintenant le cas des systèmes de polynômes à plusieurs variables.

Théorème. *Soient $\boldsymbol{P} = (P_1, ..., P_r)$ un système polynomial dans $R[Y_1, ..., Y_r]^r$ donné en représentation dense, vérifiant $d \geqslant 2$ avec $d := \max_{1\leqslant i,j\leqslant r}(\deg_{X_j}(P_i)) + 1$, et $\boldsymbol{y}_0$ un zéro résiduel de $\boldsymbol{P}$ sur $R/(p)$.*

Alors on peut calculer $\boldsymbol{y}$ à précision N en temps $d^r\,\mathsf{R}(N) + \mathcal{O}_{N\to\infty}(N)$.

Comparons-nous à nouveau avec l'itération de Newton qui nécessite à chaque étape une évaluation des équations polynomiales et de leur matrice jacobienne, et une inversion de la matrice jacobienne évaluée. Cela coûte $C\,(r\,d^r + r^\omega)\,\mathsf{I}(N) + \mathcal{O}_{N\to\infty}(N)$ où C est une constante réelle strictement positive. Le théorème précédent montre que l'on peut économiser le coût de l'inversion de la matrice jacobienne à précision N avec les algorithmes en-ligne.

Cet avantage est d'autant plus significatif que le coût de l'évaluation du système est inférieur au coût de l'inversion de la matrice. Pour mieux quantifier ce phénomène, nous adaptons notre approche détendue aux polynômes donnés par des calculs d'évaluation directs (« straight-line programs » en anglais), c'est-à-dire donnés comme une suite d'opérations arithmétiques sans branchement.

Théorème. *Soit $\boldsymbol{P}$ un système polynomial de r polynômes à r variables sur R, donné par un calcul d'évaluation direct contenant L^* multiplications. Soit $\boldsymbol{y}_0 \in (R/(p))^r$ tel que $\boldsymbol{P}(\boldsymbol{y}_0) = 0 \bmod p$ et $\det(\mathrm{Jac}_{\boldsymbol{P}}(\boldsymbol{y}_0)) \neq 0 \bmod p$.*

Alors, le calcul de $\boldsymbol{y}$ à précision N coûte $3\,L^*\,\mathsf{R}(N) + \mathcal{O}_{N\to\infty}(N)$.

Dans ce cas, l'itération de Newton coûte $C'\,(L^* + r^\omega)\,\mathsf{I}(N) + \mathcal{O}_{N\to\infty}(N)$, où C' est une autre constante réelle positive. Ainsi, notre approche détendue est particulièrement adaptée aux systèmes polynomiaux qui s'évaluent bien, comme par exemple les systèmes creux. Notons que malgré ces avantages, nos algorithmes sont plus coûteux d'un facteur logarithmique en la précision N par rapport à l'itération de Newton.

Finalement, nous implantons ces algorithmes et obtenons des temps de calculs compétitifs avec Newton, et même meilleurs sur une grande plage de paramètres. Notre implémentation est disponible dans la bibliothèque C++ ALGEBRAMIX de MATHEMAGIX [HLM+02].

6 Remontée détendue d'ensembles triangulaires

De la même manière que l'opérateur de Newton-Hensel a été adapté pour remonter des représentations à une variable dans [GLS01, HMW01] puis des représentations triangulaires dans [Sch02], nous adaptons notre approche détendue du chapitre 5 pour remonter de tels objets. Aussi, nous avions remarqué en section 5 que les algorithmes détendus peuvent économiser le coût de l'algèbre linéaire pour la remontée de racines de systèmes polynomiaux, *a contrario* des méthodes hors-ligne. Nous souhaitons aussi économiser ce coût dans le cadre présent.

Introduisons la notion de représentation à une variable d'un idéal zéro-dimensionnel $\mathcal{I} \subseteq R[X_1, \ldots, X_n]$. Soient A l'algèbre quotient $R[X_1, \ldots, X_n]/\mathcal{I}$ et $\Lambda \in A$ tels que la R-algèbre $R[\Lambda]$ engendrée par Λ est égale à A. Une *représentation à une variable* de A est une famille de polynômes $\mathfrak{P} = (Q, S_1, \ldots, S_n)$ de $R[T]$ avec $\deg(S_i) < \deg(Q)$ telle que l'on ait l'isomorphisme de R-algèbre suivant

$$\begin{array}{ccc} A = R[X_1, \ldots, X_n]/\mathcal{I} & \to & R[T]/(Q) \\ X_1, \ldots, X_n & \mapsto & S_1, \ldots, S_n \\ \Lambda & \mapsto & T. \end{array}$$

La trace la plus ancienne de cette représentation se trouve dans [Kro82] et quelques années plus tard dans [Kön03]. L'article [Mac16] contient un bon résumé de leur travaux. Le « shape lemma » de [GM89] énonce l'existence d'une telle représentation pour une forme linéaire générique Λ d'un idéal zéro-dimensionnel. Il existe tout une famille d'algorithmes pour calculer cette représentation, utilisant une résolution géométrique [GHMP97, GHH+97, GLS01, HMW01] ou des bases de Gröbner [Rou99].

Un *ensemble triangulaire* est une famille de n polynômes $\boldsymbol{t} = (t_1, \ldots, t_n)$ de $R[X_1, \ldots, X_n]$ telle que t_i est dans $R[X_1, \ldots, X_i]$, unitaire en X_i et réduit par rapport à $(t_1, \ldots, t_{i-1})$. La notion d'ensemble triangulaire est née avec l'article [Rit66] dans le contexte des algèbres différentielles. Plusieurs notions similaires ont été introduites après coup dans les articles [Wu84, Laz91, Kal93, ALMM99]. Bien que ces notions ne coïncident pas en général, elles définissent le même objet dans le cas des idéaux zéro-dimensionnels.

Les représentations à une variable peuvent être vues comme un cas particulier d'ensemble triangulaire. En effet, avec les notations précédentes, la famille $(Q(T), X_1 - S_1(T), ..., X_n - S_n(T))$ est un ensemble triangulaire de l'algèbre $R[T, X_1, ..., X_n]$. À partir de maintenant, nous considérerons les représentations à une variable comme un cas particulier d'ensembles triangulaires.

Définissons $\mathsf{Rem}(d_1, \ldots, d_n)$ comme étant le coût d'une opération arithmétique dans l'algèbre quotient $R[X_1, ..., X_n]/(t_1, ..., t_n)$ où $d_i := \deg_{X_i}(t_i)$. Quand on utilise une représentation à une variable, les éléments de $A \simeq R[T]/(Q)$ sont représentés comme des polynômes à une variable de degré strictement inférieur à $d := \deg(Q)$. Du coup, la multiplication dans A coûte quelques multiplications polynomiales.

La remontée d'ensembles triangulaires (ou de représentation à une variable) est une opération cruciale. Plusieurs implémentations d'algorithmes qui calculent des ensembles triangulaires sur les rationnels se ramènent à calculer ces objets modulo un nombre premier, puis à faire une remontée p-adique et enfin une reconstruction rationnelle. Par exemple, le logiciel Kronecker [L+02] qui calcule des représentations à une variable, et la bibliothèque RegularChains [LMX05] de Maple qui calcule des ensembles triangulaires, utilisent des remontées p-adiques. Mieux encore, une autre remontée est utilisée au cœur de l'algorithme de résolution géométrique [GLS01, HMW01] qui sous-tend Kronecker. En effet, cet algorithme manipule des représentations à une variable de courbes et nécessite donc des remontées sur les séries formelles.

En pratique, la majorité du temps de calcul d'un ensemble triangulaire est passé dans la remontée. Donc toute amélioration de la complexité de la remontée impactera directement l'algorithme complet.

Soit $\boldsymbol{f} = (f_1, \ldots, f_n) \in R[X_1, \ldots, X_n]$ un système polynomial donné par un calcul d'évaluation direct avec L opérations arithmétiques. Si L_{f_i} est le nombre d'opérations nécessaires juste pour la sortie f_i, alors nous notons $L^\perp := L_{f_1} + \cdots + L_{f_n}$ la complexité des calculs indépendants de $f_1, \ldots, f_n$, c'est-à-dire sans partage de calculs entre eux. Puisque $L_{f_i} \leqslant L$, nous avons

$$L \leqslant L^\perp \leqslant n\, L.$$

Un algorithme dû à Baur et Strassen [BS83] permet d'évaluer la matrice jacobienne de $\boldsymbol{f}$ par un calcul direct en $5\, L^\perp$ opérations.

Soit $\boldsymbol{t}_0$ un ensemble triangulaire de $R/(p)[X_1, ..., X_n]$ tel que $\boldsymbol{f} = 0$ dans l'algèbre $R/(p)[X_1, ..., X_n]/\langle \boldsymbol{t}_0 \rangle$. Nous travaillerons sous l'hypothèse que le déterminant de la matrice jacobienne $\mathrm{Jac}_{\boldsymbol{f}}$ dans $\mathcal{M}_n(R/(p)[X_1, ..., X_n])$ est inversible modulo $\boldsymbol{t}_0$. Cette condition est suffisante pour avoir l'existence et l'unicité d'un ensemble triangulaire $\boldsymbol{t}$ de $R_p[X_1, ..., X_n]$ qui se réduit à $\boldsymbol{t}_0$ modulo p et satisfait $\boldsymbol{f} = 0$ dans $R_p[X_1, ..., X_n]/\langle \boldsymbol{t} \rangle$.

L'algorithme de remontée calcule, à partir des entrées $\boldsymbol{f}$ et $\boldsymbol{t}_0$, cet unique ensemble triangulaire $\boldsymbol{t}$ à précision N.

Notre contribution consiste à donner, pour tout ensemble triangulaire p-adique, un algorithme décalé dont il est point fixe. Nous appliquons alors le cadre détendu pour la remontée de p-adiques récursifs et en déduisons un algorithme pour calculer cet ensemble triangulaire.

Par souci de clarté, nous ne donnerons que les complexités asymptotiques en la précision N des p-adiques. Énonçons les résultats de complexité.

Théorème. *Notre algorithme détendu peut effectuer la remontée de l'ensemble triangulaire* $\boldsymbol{t}$ *à précision* N *en temps*

$$C\,n\,L\,\mathsf{R}(N)\,\mathsf{Rem}(d_1,...,d_n)+\mathcal{O}_{N\to\infty}(N),$$

avec C *une constante réelle positive.*

Le précédent algorithme hors-ligne de [Sch02] fait la remontée de $\boldsymbol{t}$ à précision N en temps $C'\,(L^{\perp}+n^{\omega})\,\mathsf{I}(N)\,\mathsf{Rem}(d_1,...,d_n)+\mathcal{O}_{N\to\infty}(N)$ avec C' une autre constante réelle positive. Notre algorithme détendu n'améliore donc la complexité par rapport au précédent algorithme que dans le cas $n\,L\leqslant n^{\omega}$, c'est-à-dire pour les systèmes $\boldsymbol{f}$ qui s'évaluent en moins de n^2 opérations, en prenant $\omega=3$.

La situation est plus à notre avantage pour la remontée de représentations à une variable. Supposons que $\boldsymbol{t}$ est une représentation à une variable de degré d.

Théorème. *Notre algorithme détendu peut remonter la représentation à une variable* $\boldsymbol{t}$ *à précision* N *en temps*

$$C''\,L\,\mathsf{R}(N)\,\mathsf{M}(d)+\mathcal{O}_{N\to\infty}(N),$$

avec C'' *une constante réelle positive.*

Comparons notre algorithme avec l'algorithme hors-ligne existant de [GLS01, HMW01]. Cet algorithme hors-ligne remonte une représentation à une variable $\boldsymbol{t}$ à précision N en temps $C'''\ (L^{\perp}+n^{\omega})\ \mathsf{I}(N)\ \mathsf{M}(d)+\mathcal{O}_{N\to\infty}(N)$, où C''' est une autre constante strictement positive. Par conséquent, notre algorithme détendu fait toujours asymptotiquement moins de multiplications en-ligne que l'autre algorithme ne fait de multiplications hors-ligne. De plus, pour les systèmes polynomiaux $\boldsymbol{f}$ qui s'évaluent en moins de n^{ω} opérations, nous pouvons nous attendre à un gain de performance significatif de la part de notre algorithme.

Pour finir, nous présentons une implémentation de cet algorithme au sein de la bibliothèque C++ ALGEBRAMIX de MATHEMAGIX [HLM+02] dans le cas particulier des représentations à une variable. Notre algorithme se compare favorablement sur les exemples présentés. Mentionnons aussi que notre algorithme détendu est en cours de branchement à KRONECKER dans MATHEMAGIX avec l'aide de G. Lecerf.

7 Algorithmique de l'algèbre de décomposition universelle

Soient $\Bbbk$ un corps et $f\in\Bbbk[X]$ un polynôme séparable de degré n. Notons $\mathcal{R}:=\{\alpha_1,...,\alpha_n\}$ l'ensemble des racines de f dans une clôture algébrique de $\Bbbk$. *L'idéal des relations symétriques* $\mathcal{I}_s$ est l'idéal

$$\{P\in\Bbbk[X_1,...,X_n]\,|\,\forall\sigma\in\mathfrak{S}_n,P(\alpha_{\sigma(1)},...,\alpha_{\sigma(n)})=0\}.$$

L'algèbre de décomposition universelle est l'algèbre quotient $\mathbb{A}:=\Bbbk[X_1,...,X_n]/\mathcal{I}_s$, qui est zéro-dimensionnelle de degré $\delta:=n!$.

Le but de ce chapitre est de calculer efficacement dans $\mathbb{A}$. Nous utilisons une représentation à une variable de $\mathbb{A}$, c'est-à-dire un isomorphisme de la forme $\mathbb{A} \simeq \Bbbk[T]/Q(T)$, puisque les opérations arithmétiques dans $\mathbb{A}$ ont un coût quasi-optimal dans cette représentation. Nous détaillerons deux algorithmes liés, l'un pour calculer l'isomorphisme précédent et l'autre pour calculer le polynôme caractéristique d'un élément de $\mathbb{A}$. Ces algorithmes sont les premiers à être quasi-optimaux pour ces tâches.

Nous mesurerons le coût de nos algorithmes par le nombre d'opérations arithmétiques dans $\Bbbk$ qu'ils font. En pratique, ce modèle de complexité est bien adapté au cas où $\Bbbk$ est un corps fini ; sur $\Bbbk = \mathbb{Q}$, il conviendrait d'utiliser une remontée p-adique comme celle du chapitre 6.

La question de quelle représentation doit être utilisée pour $\mathbb{A}$ est le cœur de l'article ainsi que la clé pour obtenir de meilleurs algorithmes. Une représentation répandue est celle par *ensembles triangulaires*. Les *différences divisées*, aussi connues sous le nom de *modules de Cauchy* [Che50, RV99], sont définies par $C_1(X_1) := f(X_1)$ et récursivement

$$C_{i+1} := \frac{C_i(X_1, ..., X_i) - C_i(X_1, ..., X_{i-1}, X_{i+1})}{X_i - X_{i+1}}$$

pour $1 \leqslant i < n$. Elles forment une *base triangulaire* de $\mathcal{I}_s$. Les différences divisées sont peu coûteuses à calculer à partir de leur formule récursive mais il est difficile de rendre les calculs dans $\mathbb{A}$ efficace dans cette représentation. L'article [BCHS11] donne un algorithme de multiplication qui coûte $\tilde{\mathcal{O}}(\delta)$, mais cet algorithme cache des facteurs logarithmiques de hauts degrés dans le grand-O. Il n'y a pas d'algorithme connu pour l'inversion d'éléments de $\mathbb{A}$ qui soit quasi-linéaire.

Le seconde représentation que nous considérerons est celle à une variable. Dans cette représentation, les éléments de $\mathbb{A} \simeq \Bbbk[T]/(Q)$ sont représentés comme des polynômes à une variable de degré strictement inférieur à δ. La multiplication et l'inversion (si possible) dans $\mathbb{A}$ coûte alors respectivement $\mathcal{O}(\mathsf{M}(\delta))$ et $\mathcal{O}(\mathsf{M}(\delta)\log(\delta))$. Pour le polynôme caractéristique, la situation n'est pas aussi favorable puisqu'aucun algorithme quasi-linéaire n'est connu : le meilleur résultat, dû à [Sho94], est $\mathcal{O}(\mathsf{M}(\delta)\ \delta^{1/2} + \delta^{(\omega+1)/2})$, ce qui donne un algorithme en $\mathcal{O}(\delta^{1.69})$ opérations pour le polynôme caractéristique.

Le calcul d'une représentation à une variable de $\mathbb{A}$ est coûteux : le meilleur algorithme existant, dû à [PS11], prend en entrée un ensemble triangulaire (tel que les différences divisées) et le convertit en une représentation à une variable en temps $\tilde{\mathcal{O}}(\delta^{1.69})$.

Pour résumer, une représentation triangulaire de $\mathbb{A}$ est facile à calculer mais implique une algorithmique plutôt inefficace pour $\mathbb{A}$. D'un autre côté, le calcul d'une représentation à une variable n'est pas chose aisée, mais une fois qu'il est réalisé, certains calculs dans $\mathbb{A}$ deviennent plus rapides. La principale contribution de ce chapitre est de montrer comment contourner les inconvénients des représentations à une variable, en donnant des algorithmes rapides pour leur construction. Nous expliquons aussi comment utiliser l'arithmétique rapide à une variable dans $\mathbb{A}$ pour calculer efficacement des polynômes caractéristiques.

Dans le cas des représentations à une variable, nos algorithmes sont Las Vegas et nous donnons alors la complexité moyenne des algorithmes.

Théorème. *Supposons que la caractéristique de* $\Bbbk$ *est zéro, ou supérieure à* $2\,\delta^2$. *Alors nous pouvons calculer une représentation à une variable de* $\mathbb{A}$ *et des polynômes caractéristiques dans les temps indiqués dans le tableau suivant.*

$\mathcal{X}_{P,\mathbb{A}}$	*représentation à une variable (temps moyen)*
$\mathcal{O}(n^{(\omega+3)/2}\,\mathsf{M}(\delta)) = \tilde{\mathcal{O}}(\delta)$	$\mathcal{O}(n^{(\omega+3)/2}\,\mathsf{M}(\delta)) = \tilde{\mathcal{O}}(\delta)$

Nous proposons deux approches qui sont toutes les deux basées sur des idées classiques. La première approche calcule des polynômes caractéristiques grâce à leurs sommes de Newton, à la suite des travaux [Val89, AV94, CM94], mais en se limitant à des polynômes simples, comme par exemple les formes linéaires. Cela produira le meilleur algorithme en pratique. La deuxième approche se base sur des résultants itérés [Lag70, Soi81, Leh97, RV99] et fournit les estimations de complexité du théorème.

Finalement, nous implémentons nos algorithmes dans MAGMA 2.17.1 [BCP97] et donnons des résultats expérimentaux. Nous observons des améliorations pratiques pour le calcul de représentation à une variable de $\mathbb{A}$. Notre algorithme de changement de base entre des représentations à une variable et triangulaires est efficace ; pour une opération telle que l'inversion, et malgré le surcoût dû au changement de représentation, il est plus efficace de passer par une représentation à une variable.

8 Remontée d'invariants fondamentaux

Ce court appendice est dédié à la preuve d'un résultat utile en théorie des invariants que nous avons obtenu lors de la rédaction du chapitre 7 : nous montrons que les invariants fondamentaux de l'action d'un groupe fini se spécialisent toujours bien modulo tout nombre premier, sauf un petit nombre connu à l'avance. Ce phénomène est rare en Calcul Formel puisque empiriquement, pour des systèmes non-linéaires, les nombres premiers donnant lieu à une mauvaise réduction ne peuvent pas être déterminés directement.

Ce résultat a des implications pratiques : pour calculer les invariants fondamentaux sur les rationnels, il suffit de les calculer modulo p. La remontée vers les rationnels est alors triviale.

Une correspondance privée avec H. E. A. Campbell, D. Wehlau et M. Roth nous a appris que ce résultat leur était déjà connu mais, à notre connaissance, jamais publié. Nous décidons de l'inclure dans cette thèse pour qu'il puisse être utilisé en pratique : à notre connaissance, un logiciel tel que MAGMA [BCP97] n'utilise pas ce résultat pour calculer des anneaux d'invariants.

Partie I

On-line algorithms