Réécriture algébrique dans les systèmes d'équations différentiellespolynomiales en vue d'applications dans les Sciences du Vivant

François Boulier

Université des Sciences et Technologies de Lille
Laboratoire d'Informatique Fondamentale de Lille
Équipe Calcul Formel
59655 Villeneuve d'Ascq, France

Mémoire d'Habilitation à Diriger des Recherches
Numéro d'ordre : H497

Réécriture algébrique dans les systèmes d'équations différentielles polynomiales en vue d'applications dans les Sciences du Vivant

François Boulier

Soutenance le 4 mai 2006 devant le jury

Max Dauchet	LIFL	Président
Daniel Lazard	LIP6	Rapporteur
Claude Gomez	INRIA	Rapporteur
Greg Reid	ORCCA	Rapporteur
Gérard Jacob	LIFL	Directeur
Michel Petitot	LIFL	Examinateur
Bruno Salvy	INRIA	Examinateur
Bernard Vandenbunder	IBL	Examinateur

Remerciements

« *Une journée existe et la précédente existe et celle qui précède la précédente, et celle d'avant ... et elles sont bien agglutinées, des dizaines ensemble, des trentaines, des années entières, et on n'arrive pas à vivre,* soi, *mais seulement à vivre* la vie, *et l'on est tout étonné.*

L'homme du pays de la Magie sait bien cela. Il sait que la journée existe et très forte, très soudée, et qu'il doit faire ce que la journée ne tient pas à faire.

Il cherche donc à sortir sa journée du mois. C'est l'attraper qui est difficile. Et ce n'est pas le matin qu'on y arriverait. Mais vers deux heures de l'après–midi, il commence à la faire bouger, vers deux heures, elle bascule, elle bascule ; là, il faut être tout à son affaire, peser, tenir, lâcher, décharger, convoyer par–dessus.

Enfin il la détourne, *la* chevauche. *Il s'en rend maître. Et vite à l'important, vite, obligé qu'il sera — hélas ! — à abandonner la journée à l'enclenchement des suivantes au plus tard vers minuit. Mais que faire ? C'est là le tribut à l'existence animale.* »

Au pays de la Magie
Henri Michaux

Mes premiers remerciements vont à Isabelle, Hugo et Sophie qui me demandent bien sûr beaucoup de temps mais me donnent la force et l'équilibre nécessaires pour mener ce travail.

Je remercie tous les membres du jury et en particulier les trois rapporteurs de ce mémoire. Daniel Lazard était déjà un des rapporteurs de ma thèse de troisième cycle. Une partie importante des résultats que j'expose lui est due soit directement soit par l'intermédiaire de ses étudiants. J'ai rencontré Greg Reid pour la première fois en 1995. Depuis dix ans, nous avons eu de nombreux échanges très fructueux. Je connais beaucoup moins bien Claude Gomez. En acceptant de rédiger un rapport sur ce mémoire, dont le contenu est éloigné de son domaine de recherche, Claude Gomez m'apporte un avis très important pour le futur.

Je remercie tous les membres de l'équipe de calcul formel du LIFL, en particulier Gérard Jacob qui m'a encadré dans la préparation de cette habilitation à diriger des recherches, François Lemaire et Alexandre Sedoglavic qui ont relu ce document, Michel Petitot qui dirige l'équipe, Nour-Eddine Oussous, Leopold Weinberg, Hoang Ngoc Minh, Raouf Dridi, Natacha Skrzypczak et Asli Ürgüplü. Sans oublier ni Sylvain Neut ni Marc Moreno Maza.

Je remercie enfin Lilianne Denis–Vidal, Ghislaine Joly–Blanchard, Marc Lefranc, François–Yves Bouget et tous les membres du groupe de travail consacré à la modélisation du cyle cellulaire d'*ostreococcus tauri*. Ils m'apportent les applications qui aiguillent mon travail.

Le 13 mars 2006,
François Boulier

Table des matières

Introduction

Fil directeur. Dans son livre « Physique quantique et représentation du monde », Erwin Schrödinger écrit (les expressions en italique sont soulignées par lui) : « tout scientifique devrait *à long terme* être capable d'expliquer *à n'importe qui* ce qu'il a fait et la raison pour laquelle il l'a fait ».

Je suis un informaticien, spécialisé dans le domaine mathématique de la simplification des systèmes d'équations différentielles ordinaires ou aux dérivées partielles polynomiales. J'ai inventé pendant ma thèse [7, 9] un bel algorithme théorique sur ce sujet, nommé *Rosenfeld–Gröbner*. Les années qui ont suivi, j'en ai beaucoup travaillé la théorie [10, 8, 12, 11]. Aujourd'hui, j'essaie de le « rendre utile ».

Je m'explique : l'algorithme *Rosenfeld–Gröbner* a beaucoup d'applications théoriques que je ne trouve pas très ... satisfaisantes. C'est subjectif bien sûr. Voici un exemple un peu caricatural : *Rosenfeld–Gröbner* permet de décider si un système quelconque d'équations différentielles polynomiales admet au moins une solution analytique. C'est une application très importante d'un point de vue théorique mais sans véritable intérêt pratique. J'ai étudié de nombreux systèmes différentiels. Ils ont tous des solutions, pour des raisons évidentes qui tiennent à leur conception.

Je suis plutôt à la recherche d'applications convaincantes en physique, chimie ou biologie. Trois difficultés apparaissent. La première : il faut « créer la demande ». Les praticiens des domaines scientifiques visés ne sont pas habitués à utiliser les méthodes que nous proposons. *Rosenfeld–Gröbner* résout un problème théorique qui est resté quasiment ouvert jusqu'à son invention. Que font alors les praticiens confrontés à de tels problèmes ? Ils résolvent « à la main » leurs cas particuliers [103] ou les reformulent.

Deuxième difficulté : la théorie mathématique sur laquelle *Rosenfeld–Gröbner* s'appuie n'est connue que d'une petite minorité de scientifiques. Il s'agit de « l'algèbre différentielle ». Elle a été inventée au cours de la première moitié du vingtième siècle à la grande époque de la géométrie algébrique où ont été mises au point la théorie des anneaux Nœthériens, la théorie des idéaux ... Joseph Fels Ritt écrit dans l'introduction de son livre [86] qu'il a été très impressionné par le livre « Die Moderne Algebra » de Bruno Louis Van Der Waerden et qu'il a souhaité développer une théorie équivalente pour les équations différentielles. L'algèbre différentielle est une très belle théorie algébrique des équations différentielles mais il reste à la populariser auprès du grand public scientifique.

Troisième difficulté : l'algorithmique de *Rosenfeld–Gröbner* est difficile. D'une part, même pour qui connaît l'algèbre différentielle, l'algorithmique s'appuie sur des arguments d'algèbre

commutative non élémentaires (le « *unmixedness theorem* » de Francis Sowerby Macaulay pour ne citer que lui) qui avaient échappé aux spécialistes jusqu'aux années 1995–2000. D'autre part, il est très difficile d'implanter l'algorithme en dehors d'un système de calcul formel. L'implantation que j'ai réalisée dans les bibliothèques *BLAD* en langage C est assez minimale. Elle comporte quarante mille lignes de code dont vingt–cinq mille pour le seul calcul du pgcd de deux polynômes en plusieurs indéterminées et à coefficients entiers. Le public de physiciens, chimistes ou biologistes que j'aimerais séduire programme la plupart du temps en FORTRAN ou en C. Pour les convaincre d'utiliser *Rosenfeld–Gröbner*, il est donc inutile d'écrire des articles de recherche dans des journaux de calcul formel : il faut fournir un composant logiciel (par exemple une bibliothèque) prêt à l'emploi.

Applications. Je ne connais que deux applications (au sens expliqué plus haut) d'un algorithme d'élimination en algèbre différentielle tel que *Rosenfeld–Gröbner*. La première concerne le problème de l'estimation des valeurs des paramètres d'un système dynamique à partir d'observations (de mesures), lorsque toutes les variables du système dynamique ne sont pas observées. Cette application a été inventée par Ghislaine Joly–Blanchard, Lilianne Denis–Vidal et Céline Noiret qui ont travaillé en collaboration avec Michel Petitot. Elle est exposée dans la thèse [75] de Céline Noiret. L'étude de l'identifiabilité locale ou globale d'un système par des méthodes d'élimination différentielle est un problème ancien [105, 42, 78, 32, 34, 33, 64] et toujours actuel [3, 93]. L'originalité de la thèse de Céline Noiret réside d'une part dans l'utilisation d'algorithmes d'élimination différentielle rigoureux, d'autre part dans l'étude complète d'exemples, c'est–à–dire jusqu'à l'estimation finale des paramètres. C'est ce problème qui a motivé l'écriture des bibliothèques *BLAD* dès les années 2000. J'ai monté avec Lilianne Denis–Vidal le projet *LÉPISME* (pour « *logiciel dédié à l'estimation de paramètres et à l'identification systématique de modèles* ») qui a reçu un financement MathSTIC en 2003. La première version d'un logiciel dédié à l'estimation de paramètres a été réalisée en août 2004 en même temps que la première version des bibliothèques *BLAD*. Ce logiciel a été conçu en vue d'une application en biologie. Il a été programmé par François Lemaire, Thibaut Henin et moi en juillet–août 2004 et a donné lieu à l'article [6]. Nous avons répondu sans succès courant 2005 à un appel à projets INSERM–INRIA–CNRS pour financer la continuation de notre logiciel. Ce retour d'informations nous conduit à reprendre la réflexion sur le futur de notre projet, en collaboration avec Alexandre Sedoglavic, Philippe Durif et d'autres. Parallèlement, Leopold Weinberg a commencé en 2003 une thèse coencadrée par Nour–Eddine Oussous et moi sur l'aspect « identification de modèles » de *LÉPISME* à partir des idées exposées dans [76].

La seconde application que je connaisse vient d'un problème d'étude d'un modèle d'horloge « circadienne » (c'est–à–dire d'horloge de période approximativement égale à vingt quatre heures) dans un organisme unicellulaire [103]. Dans ce contexte, un algorithme tel que *Rosenfeld–Gröbner* permet d'automatiser une partie d'un processus de réduction de modèles, ce qui permet de déterminer des intervalles de valeurs pour les paramètres du modèle pour lesquelles les trajectoires des variables du système oscillent, c'est–à–dire se comportent comme des horloges. Cette perspective d'application est récente : elle est apparue au cours

du stage de Master 2 Recherche de Natacha Skrzypczak en 2005 qui était coencadré par Michel Petitot et moi. Nous sommes entrés récemment en contact avec l'un des auteurs de l'article [103] sur ce sujet. Les discussions qui ont suivi nous ont amenés à amorcer l'étude d'un modèle beaucoup plus gros (plus de trois cents variables et deux cents réactions), en interaction avec l'un des auteurs de l'article [77] qui le présente.

Activité de l'équipe. Bien que l'équipe « calcul formel » continue son activité de recherche dans son cœur de métier, on voit qu'un effort très important est effectué depuis quelques années pour trouver des applications dans les sciences du vivant. L'équipe fait partie depuis 2004 de l'Institut de Recherches Interdisciplinaires créé à l'initiative de Bernard Vandenbunder. Le cours de Master 2 Recherche de l'équipe s'est spécialisé sur la modélisation des phénomènes biologiques. Nous nous intéressons de près au langage SBML (pour « *Systems Biology Markup Language* »), une variante de XML qui constitue un standard de représentation des modèles biologiques : nous prévoyons de l'adopter dans le cadre de notre projet *LÉPISME* et le mémoire de Master 2 Recherche de Natacha Skrzypczak en 2005 lui était à moitié consacré. Michel Petitot, François Lemaire et moi participons depuis 2004 à un groupe de travail consacré à la modélisation du rythme circadien d'une algue verte. Ce groupe de travail est composé de membres de plusieurs disciplines différentes dont le biologiste François–Yves Bouget, spécialiste de l'algue verte et le physicien Marc Lefranc.

Logiciels de calcul formel. Pour populariser l'algèbre différentielle et en faciliter l'emploi, j'ai développé deux logiciels importants de calcul formel. En 1995–1996, j'ai réalisé le paquetage *diffalg* en MAPLE lors d'un stage post–doctoral au *Symbolic Computation Group* de l'université de Waterloo, Ontario, Canada. L'algorithme *Rosenfeld–Gröbner* en constitue le cœur. Ce paquetage est depuis distribué avec la bibliothèque standard du logiciel MAPLE. Plusieurs personnes l'ont amélioré au fil des ans, et interfacé avec d'autres solveurs de MAPLE. Je pense en particulier à Évelyne Hubert, Allan D. Wittkopf et à François Lemaire.

À partir de 2000, j'ai commencé à réaliser les bibliothèques *BLAD* en langage C. La première version a été terminée en août 2004. Ces bibliothèques constituent une sorte d'analogue de *diffalg* mais il y a deux différences importantes : d'abord les bibliothèques *BLAD* sont « open source » et protégées par la licence LGPL alors que *diffalg* est un paquetage d'un logiciel propriétaire ; ensuite, *BLAD* est un ensemble de bibliothèques destinées à être appelées par un autre programme alors que *diffalg* est conçu pour être utilisé interactivement. Les deux logiciels correspondent à des usages différents et sont donc complémentaires. L'idée de développer *BLAD* m'est venue au cours de mes participations au groupe de travail du projet INRIA SPACES, suite à plusieurs discussions avec Jean–Charles Faugère et Fabrice Rouillier qui m'ont convaincu de l'importance de réaliser des chaînes logicielles les plus complètes possibles mettant en œuvre de l'élimination différentielle. C'est en réalisant de telles chaînes qu'on peut déterminer si une théorie s'applique ou pas et, si c'est le cas, où sont les goulots d'étranglement. Jean–Charles Faugère et Fabrice Rouillier m'ont aussi

fourni le mécanisme de gestion de la mémoire [38] qui est employé dans leurs logiciels.

Était–ce une bonne idée de consacrer quatre ans au développement des bibliothèques *BLAD*? Je pense que oui mais un critère de réussite convaincant serait l'intégration de *BLAD* dans une grande bibliothèque de calcul numérique telle que la *Gnu Scientific Library* ou dans un logiciel tel que *scilab*. Ce qui manque considérablement aux versions actuelles des bibliothèques *BLAD* et qui freine peut–être cette intégration, c'est l'absence d'intégrateurs numériques applicables aux « chaînes différentielles régulières » produites par *Rosenfeld–Gröbner*. Ce sujet m'intéresse de très près en ce moment. Une difficulté vient du fait que les chaînes différentielles régulières définissent souvent des systèmes implicites et raides.

Est–ce une bonne idée d'essayer d'associer des méthodes symboliques exactes telles que celles fournies par *BLAD* et des méthodes numériques inexactes ? Là encore, je pense que oui. La question a été posée lors de mes visites au séminaire de SPACES. Je pense que la donne est différente pour les systèmes différentiels et pour les systèmes polynomiaux (spécialité de SPACES). D'un point de vue théorique et algorithmique, les deux domaines sont très proches mais pas du point de vue des applications. Les méthodes (bases de Gröbner, chaînes régulières) disponibles pour les systèmes polynomiaux peuvent se présenter comme des alternatives aux méthodes numériques (méthode de Newton). C'est tout–à–fait sensé, par exemple dans le domaine des systèmes polynomiaux qui n'ont qu'un nombre fini de solutions complexes, parce qu'on dispose d'algorithmes exacts et efficaces d'isolation de racines. Sous un angle beaucoup plus théorique, ce point de vue est conforté par les résultats de l'article [18]. Pour les équations différentielles par contre, je ne connais pas d'algorithmes pouvant jouer un rôle similaire à l'exception peut–être de méthodes par intervalles [47] mentionnées dans [45, page 49]. La coopération symbolique–numérique me semble donc indispensable ... pour les méthodes symboliques.

Le mémoire. J'ai rédigé les cent quatre vingts pages et plus qui suivent en cherchant à expliquer la théorie et l'algorithmique sur lesquelles j'ai travaillé comme je pourrais essayer de la présenter à un scientifique non spécialiste du sujet. J'ai essayé d'expliquer plutôt que de prouver et je me suis efforcé de présenter en même temps les bibliothèques *BLAD*.

Le mémoire comporte quatre parties. Les trois premières commencent par une sorte de tutoriel. Je me suis servi pour les réaliser d'un support que j'ai rédigé pour l'école d'été *Open Software for Algebraic and Geometric Computations* tenue en septembre 2005 à Sophia Antipolis où j'ai présenté *BLAD*. Chaque tutoriel est conçu pour présenter à un néophyte (parfois éclairé) les notions qui sont étudiées plus théoriquement dans les chapitres suivants.

La première partie correspond à une prise de contact avec l'algèbre différentielle. Le tutoriel introductif est une introduction à l'élimination différentielle par analogie avec le pivot de Gauss. Le chapitre suivant présente des notions de base un peu « calculatoires ». Le dernier chapitre tente une présentation originale des notions de solution qui sont compatibles avec les algorithmes d'élimination différentielle qui nous concernent. Ce dernier chapitre résume de nombreuses années de réflexion menées dans l'équipe. Il expose aussi l'un des résultats les plus importants de la thèse [62] de François Lemaire.

La seconde partie présente les objets produits par l'élimination différentielle : les « chaînes

différentielles régulières ». Le tutoriel introductif est directement inspiré du projet *LÉPISME*. L'algorithme utilisé est une variante spécialisée de *Rosenfeld–Gröbner* nommée *PARDI*. Trois chapitres suivent. Le premier présente les chaînes régulières algébriques puis différentielles au travers de l'algorithme d'Euclide. Il reprend la progression que j'avais suivie lorsque j'assurais un demi–cours dans la filière « calcul formel » du DEA Algorithmique à Paris VI en 2000–2002. Le chapitre suivant présente des applications algorithmiques immédiates des chaînes régulières. L'existence d'un algorithme de forme normale me permet de donner une présentation nouvelle de quelques constructions connues (développement en série de solutions de systèmes différentiels). Le troisième chapitre est dense : j'y ai refait les preuves des théorèmes clefs qui justifient la construction des chaînes régulières. La preuve unifiée de l'équidimensionnalité des idéaux de la forme $(A) : I_A^\infty$ et de ceux de la forme $(A) : S_A^\infty$ qui s'y trouve est originale.

La troisième partie est consacrée au problème du calcul de la représentation d'un système différentiel quelconque par une famille finie de chaînes différentielles régulières. Le tutoriel introductif consiste en une analyse de l'article [103]. On y explique les motivations et la démarche des auteurs et on montre à quelle étape un algorithme tel que *Rosenfeld–Gröbner* aurait pu être utilisé. Le chapitre suivant présente une version de *Rosenfeld–Gröbner* modernisée, proche de l'implantation en *BLAD* et des dernières améliorations apportées à *diffalg* par François Lemaire. On y donne aussi des résultats récents, permettant d'optimiser les mécanismes de complétion pour les systèmes aux dérivées partielles.

La quatrième partie ne comporte qu'un chapitre qui décrit les principaux choix de conception que j'ai faits pour les bibliothèques *BLAD*.

On ne trouve pas dans ce mémoire de comparaisons de performances entre *BLAD* et d'autres logiciels. Ce serait prématuré : les effectuer me pousserait naturellement à optimiser *BLAD* or on n'optimise pas un logiciel en cours de développement. C'est un principe de génie logiciel. Il faut d'abord réaliser des programmes d'application qui utilisent *BLAD* et ensuite seulement déterminer ce qu'il convient d'optimiser. Voici deux anecdotes pour étayer mes affirmations.

Dans le cadre du projet *LÉPISME*, on s'est aperçu sur des exemples que les limitations de la méthode étaient dues à des problèmes numériques (évaluation numérique imprécise des dérivées d'ordre deux et plus) et pas à l'élimination différentielle. Optimiser *BLAD* est donc sans intérêt dans ce contexte.

Lors de la mise au point de l'algorithme *PARDI* en 2001, François Lemaire et moi comparions les performances en temps de calcul d'une implantation MAPLE avec une implantation en une vieille version de *BLAD* en C++ sur l'exemple des équations d'Euler pour un fluide incompressible en deux dimensions. Chaque fois qu'une implantation avait de meilleures performances que l'autre, nous en cherchions la raison et nous améliorions l'autre en conséquence. Il s'agissait à chaque fois d'un choix heuristique plus ou moins judicieux. Au bout du compte, *PARDI* en *BLAD* a largement surpassé *PARDI* en MAPLE mais le code de l'implantation de *PARDI* en *BLAD* était devenue difficile à lire et donc à faire évoluer. J'ai finalement supprimé toutes ces optimisations lorsque j'ai refondu *BLAD* en langage C.

Enseignement. En annexe du mémoire, on trouve une copie de la plupart des articles que j'ai écrits, la documentation technique des bibliothèques *BLAD* mais aussi trois supports de cours que j'ai rédigés. Les enseignements que j'assure sont liés à mon activité de recherche.

L'unité d'enseignement « calcul formel et sciences de la matières » est un cours de programmation destiné à de futurs physiciens et chimistes. Il est conçu avec des membres de l'UFR de physique et s'adresse à des étudiants de deuxième année de licence. Les étudiants y revoient pour commencer des notions de programmation qu'ils ont apprises en première année. On les révise et on montre comment elles se traduisent dans le langage de programmation de MAPLE. Le cours évolue ensuite vers l'étude des équations différentielles : méthodes numériques (Euler et Runge–Kutta) puis analyse qualitative des systèmes différentiels linéaires en deux variables et à coefficients constants (rôle des valeurs propres de la matrice des coefficients du système). L'emploi d'un logiciel de calcul formel confère une originalité certaine à cet enseignement : il permet aux étudiants de mener des « démonstrations assistées par ordinateur » dont tous les calculs symboliques sont menés par logiciel (preuve que la méthode d'Euler est d'ordre 1, calcul des équations définissant les coefficients de Runge–Kutta).

L'unité d'enseignement « systèmes polynomiaux, que signifie : résoudre ? » est une unité optionnelle de la licence informatique. Elle présente une chaîne logicielle dédiée à la résolution réelle exacte de systèmes polynomiaux ayant un nombre fini de solutions complexes (un calcul de base de Gröbner pour un ordre d'élimination suivi d'un algorithme d'isolation de racines réelles). Ce cours a été conçu avec Fabrice Rouillier qui y présente une application à la commande de robots parallèles. De nombreux spécialistes de calcul formel ont contribué à son amélioration.

L'unité d'enseignement « algorithmique » est une unité de la deuxième année de la licence mention informatique. J'y présente la programmation linéaire, l'algorithme du simplexe et des algorithmes de base (preuves et implantation) de la théorie des graphes. Le logiciel AMPL est utilisé pour modéliser par programmes linéaires (en variables réelles et entières) des problèmes d'optimisation énoncés en Français. J'essaie à la fois de fournir des bases solides aux étudiants qui poursuivront en Master et d'apprendre un outil immédiatement utilisable en entreprise aux étudiants qui quitteront le cursus universitaire avec une licence. En enseignement aussi, il me paraît important de lier les théories à leurs applications.

Chronologie.

1990. Je m'inscris en thèse sous la direction de Gérard Jacob.

1994. Je soutiens ma thèse [7].

1995. Première publication [9] sur *Rosenfeld–Gröbner* avec Daniel Lazard, François Ollivier et Michel Petitot. Je pars pour un stage post–doctoral d'un an au *Symbolic Computation Group* de l'université de Waterloo (Ontario). Je réalise la première version du paquetage *diffalg* pour MAPLE 5.

1996. Je suis élu Maître de Conférences. Mon travail se concentre sur la théorie et l'algorithmique de *Rosenfeld–Gröbner*.

1997. Deuxième article [10] en commun avec Daniel Lazard, François Ollivier et Michel Petitot. Je me marie.

1998. François Lemaire s'inscrit en thèse. Je coencadre cette thèse avec Gérard Jacob.

1999. Article [8]. Naissance de Hugo. Achat d'une maison avec beaucoup de travaux.

2000. Article [12] commun avec François Lemaire. J'obtiens une délégation CNRS de deux ans. Mon projet consistait à améliorer le paquetage *diffalg*. Je profite de ma délégation pour participer au séminaire du projet INRIA SPACES. J'assure pendant deux ans un demi–cours dans la filière « calcul formel » du DEA Algorithmique à Paris. Début d'implantation des bibliothèques *BLAD*. Marc Moreno Maza est élu Maître de Conférences.

2001. Article [11] en commun avec Marc Moreno Maza et François Lemaire. Article [15] en commun avec Sylvain Neut. Premiers contacts avec Bernard Vandenbunder pour la mise en place de l'Institut de Recherche Interdisciplinaire. Naissance de Sophie.

2002. François Lemaire soutient sa thèse et part en stage post–doctoral d'un an et demi au centre *ORCCA* de l'université *Western Ontario* de London (Ontario). Il apporte au paquetage *diffalg* les améliorations décrites dans [12, 11] (j'aurais eu les plus grandes difficultés à effectuer ces modifications depuis Lille). Marc Moreno Maza quitte son poste de Maître de Conférences pour un poste permanent au centre *ORCCA*. Je m'inscris en HDR à l'occasion de la soutenance de François Lemaire. Ma délégation CNRS se termine. Je prends la responsabilité de la licence de l'IUP GMI et je m'investis dans la préparation de la réforme LMD. Je suis co-organisateur local de la conférence internationale de calcul formel ISSAC 2002.

2003. Alexandre Sedoglavic est élu Maître de Conférences. Un financement MathSTIC est accordé pour le projet *LÉPISME*. Gérard Jacob prend sa retraite. Il est remplacé à la direction de l'équipe par Michel Petitot. Je reprends le cours d'algorithmique qu'il assurait en licence informatique. Leopold Weinberg est recruté sur un poste de PRAG et s'inscrit en thèse. Je coencadre sa thèse avec Nour–Eddine Oussous. Je crée avec Fabrice Rouillier et Éric Wegrzynowski un cours en deuxième année de DEUG intitulé « systèmes polynomiaux, que signifie : résoudre ? » consacré à la résolution réelle de systèmes polynomiaux n'ayant qu'un nombre fini de solutions complexes.

2004. Achèvement de la première version des bibliothèques *BLAD* et du logiciel *LÉPISME*. Article [6] en commun avec Lilianne Denis–Vidal, Thibaut Henin et François Lemaire. Début du groupe de travail sur la modélisation du rythme circadien chez l'algue verte.

2005. François Lemaire est élu Maître de Conférences. Stage de Master 2 Recherche de Natacha Skrzypczak coencadré par Michel Petitot et moi.

2006. Stage de Master 2 Recherche d'Asli Ürgüplü coencadré par Alexandre Sedoglavic, François Lemaire et moi. Article [13] en commun avec François Lemaire et Marc Moreno Maza.

Première partie

Premier contact

Chapitre 1

Introduction à l'élimination différentielle et aux logiciels

Ce chapitre constitue une introduction aussi élémentaire que possible à l'élimination différentielle et à l'utilisation des deux outils logiciels que sont le paquetage *diffalg* de MAPLE et les bibliothèques *BLAD*. On présente l'élimination différentielle par analogie avec le pivot de Gauss. Essentiellement, on montre dans ce chapitre le rôle que jouent les ordres sur les indéterminées dans les algorithmes d'élimination et la façon dont on spécifie ces ordres dans quelques paquetages MAPLE et dans *BLAD*. On en profite pour introduire par la pratique plusieurs notions qui sont approfondies dans les chapitres suivants. Les bibliothèques *BLAD* sont accessibles à l'URL http://www.lifl.fr/~boulier/BLAD.

1.1 L'élimination de Gauss

L'élimination différentielle est un procédé qui transforme un système d'équations différentielles en un autre système équivalent (en un sens à préciser) mais plus facile à résoudre. C'est là, mais dans un cadre plus compliqué, le principe même de l'élimination de Gauss qu'on rappelle sur un exemple, en utilisant le paquetage *LinearAlgebra* de MAPLE 9.

```
> with (LinearAlgebra):
> S1 := [x - 3*y - z = 4, 2*x - y + 3*z = 5];
                  S1 := [x - 3 y - z = 4, 2 x - y + 3 z = 5]

> A, b := GenerateMatrix (S1, [x, y, z]):
> Ab1 := < A | b >;
                                    [1    -3    -1    4]
                           Ab1 := [                    ]
                                    [2    -1     3    5]

> Ab2 := GaussianElimination (Ab1);
                                  [1    -3    -1     4]
```

```
                    Ab2 := [                    ]
                           [0     5     5    -3]

> S2 := GenerateEquations (Ab2, [x, y, z]);
                  S2 := [x - 3 y - z = 4, 5 y + 5 z = -3]
```

L'application du pivot de Gauss sur le système S_1 produit un système équivalent (c'est–à–dire ayant mêmes solutions) S_2 dont la dernière équation ne comporte pas l'indéterminée x. Le pivot de Gauss a « éliminé » x. Pourquoi l'indéterminée éliminée est–elle x ? Parce que, dans la matrice $\langle A \mid b \rangle$, les coefficients de x apparaissent sur la première colonne et donc parce que la liste d'indéterminées fournie en deuxième paramètre à la fonction *GenerateEquations* commence par x. Dans la théorie de la simplication algébrique, il est classique de dire qu'une telle liste fournit un « ordre » sur l'ensemble des variables. Les plus grandes variables vis–à–vis de cet ordre sont celles qui figurent le plus à gauche dans la liste :

$$[x,\, y,\, z] \equiv x > y > z.$$

Résumons–nous : l'élimination de Gauss est un procédé qui prend deux paramètres en entrée : un système linéaire et un ordre sur les indéterminées. Elle transforme le système en un système équivalent dans lequel certaines indéterminées sont éliminées. Plus l'indéterminée est grande vis–à–vis de l'ordre, plus elle est susceptible d'être éliminée.

1.2 L'élimination algébrique

Le principe de l'élimination de Gauss se généralise aux systèmes d'équations polynomiales. On obtient ce qu'on appelle « l'élimination algébrique ». Il y a au moins deux généralisations possibles qui conduisent soient à la théorie des « bases de Gröbner » soit à la théorie des « chaînes régulières ». On se concentre ici uniquement sur la seconde. Un algorithme de décomposition en chaînes régulières est un procédé qui prend en entrée un système d'équations polynomiales et un ordre sur les indéterminées. Il transforme le système d'entrée en une famille de « chaînes régulières ». Une chaîne régulière est un système d'équations polynomiales qui satisfait entre autres, la propriété suivante : chaque équation introduit au moins une indéterminée en suivant l'ordre fourni en paramètre (on dit aussi que le système est « triangulaire »). Tout système triangulaire n'est pas une chaîne régulière mais les autres propriétés importantes sont trop techniques pour être décrites dans cette section. Prenons un exemple. Avec les paramètres suivants

$$\begin{aligned} &\textit{système :} \quad z^2 - 1 = 0, \quad (z+1)(x-y-1) = 0, \\ &\qquad\qquad\qquad\quad (y+1)(x-y-z) = 0, \quad x^2 - y^2 - 2\,z\,y - 1 = 0, \\ &\textit{ordre :} \qquad x > y > z, \end{aligned}$$

un algorithme de décomposition en chaînes régulières pourrait produire la famille suivante de trois chaînes régulières (le résultat n'est pas défini de façon unique) :

$$\begin{array}{lll} \textit{première chaîne :} & z-1=0, & x-y-1=0\ ; \\ \textit{deuxième chaîne :} & z+1=0, & x-y+1=0\ ; \\ \textit{troisième chaîne :} & z+1=0, & y+1=0, \quad x^2-4=0. \end{array}$$

Le système initial est équivalent à la famille de chaînes produite : toute solution du système initial est solution d'au moins une chaîne régulière et réciproquement. Dans chaque chaîne, la première équation introduit l'indéterminée z (x et y sont éliminés). Dans les deux premières chaînes, la deuxième équation introduit x et y. Dans la troisième chaîne, la deuxième équation introduit y (x est éliminé) alors que la dernière équation introduit x. Le fait que z n'apparaisse que dans la première équation est un hasard dû à l'exemple.

Pourquoi la sortie de l'algorithme est–elle une famille de systèmes et pas un seul système comme dans le cas de l'élimination de Gauss ? Dit autrement, pourquoi y a–t–il des « scindages » ou des « discussions de cas » ? C'est parce que, dans la théorie des chaînes régulières, une équation telle que

$$(z+1)(x-y-1)=0$$

est vue comme l'équation

$$x=\frac{(z+1)(y+1)}{z+1}$$

qui se simplifie en

$$x=y+1$$

sous réserve que $z+1\neq 0$. Le cas $z+1=0$ ne doit pas être oublié et doit être traité séparément : un scindage survient. L'indéterminée en partie gauche de l'équation est la plus grande indéterminée qui figure dans l'équation vis–à–vis de l'ordre. On l'appelle « l'indéterminée principale » de l'équation.

1.2.1 Élimination en MAPLE

Le paquetage *Triade* de MAPLE a été conçu et réalisé par François Lemaire, Yuzhen Xie et Marc Moreno Maza. Il est incorporé dans MAPLE 10 sous le nom *RegularChains* [61]. Il fournit un algorithme de décomposition en chaînes régulières, du nom de *Triangularize*. Le voici, appliqué sur l'exemple. La sortie est un peu différente de celle présentée ci–dessus. L'ordre sur les indéterminées est défini lors de la création du *polynomial_ring*.

```
> with(Triade);
[ChainTools, DisplayPolynomialRing, Equations, ExtendedRegularGcd, Inequations,

    Initial, Inverse, IsRegular, MainDegree, MainVariable, MatrixCombine,

    MatrixTools, NormalForm, PolynomialRing, Rank, RegularGcd,
```

```
    RegularizeInitial, Separant, SparsePseudoRemainder, Tail, Triangularize]

> R := PolynomialRing([x,y,z]);
                              R := polynomial_ring

> sys := [z^2 - 1, (z+1)*(x-y-1), (y+1)*(x-y-z), x^2 - y^2 - 2*z*y -1]:

> result := Triangularize (sys, R);
           result := [regular_chain, regular_chain, regular_chain]

> Equations(result[1], R);
                               [-y + x + 1, z + 1]

> Equations(result[2], R);
                                [x - y - 1, z - 1]

> Equations(result[3], R);
                              [x - 2, y + 1, z + 1]
```

1.2.2 Élimination avec BLAD

On peut également calculer des décompositions en chaînes régulières grâce à l'algorithme *Rosenfeld–Gröbner* bien que cet algorithme ait été conçu pour traiter des systèmes différentiels. Voici un programme en langage C effectuant la décomposition grâce aux bibliothèques *BLAD*.

```
/* File algebraic.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bad_intersectof_regchain ideal;
    struct bap_tableof_polynom_mpz eqns, ineqns;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);
    ba0_sscanf2 ("ordering (derivations = [], blocks = [x, y, z])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bad_init_intersectof_regchain (&ideal);
    ba0_sscanf2 ("intersectof_regchain ([], \
                          [autoreduced, primitive, normalized])",
                 "%intersectof_regchain", &ideal);
```

```
    ba0_init_table ((ba0_table)&eqns);
    ba0_init_table ((ba0_table)&ineqns);
    ba0_sscanf2 ("[z^2 - 1, (z + 1)*(x - y - 1), (y + 1)*(x - y - z), \
                                x^2 - y^2 - 2*z*y - 1]", "%t[%Az]", &eqns);
    bad_Rosenfeld_Groebner (&ideal, &eqns, &ineqns, (bav_tableof_variable)0);
    ba0_printf ("%intersectof_regchain\n", &ideal);
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Ce programme C peut être compilé pour produire un exécutable. On donne ci–dessous une commande de compilation utilisant *gcc* pour les machines fonctionnant sous UNIX. On a supposé que la *Gnu Multiple Precision Library* (*GMP*) était installée dans */usr/local/gmp* (sous–répertoires *include* et *lib*). On a supposé que les bibliothèques *BLAD* étaient installées dans */usr/local/blad* (sous–répertoires *include* et *lib*).

```
$ gcc -I/usr/local/gmp/include -I/usr/local/blad/include algebraic.c
$ gcc -L/usr/local/gmp/lib -L/usr/local/blad/lib \
      -Wl,-rpath /usr/local/gmp/lib -Wl,-rpath /usr/local/blad/lib \
      algebraic.o -lbad -lbap -lbav -lba0 -lgmp
```

Il est bien sûr recommandé d'utiliser un « makefile ». Voici le mien.

```
GMPROOT  = /usr/local/gmp
BLADROOT = /usr/local/blad
CFLAGS   = -g -Wall
LDFLAGS  = -g -L${GMPROOT}/lib -L${BLADROOT}/lib \
           -Wl,-rpath ${GMPROOT}/lib -Wl,-rpath ${BLADROOT}/lib
CPPFLAGS = -I${GMPROOT}/include -I${BLADROOT}/include
CC       = gcc
LIBS     = -lbad -lbap -lbav -lba0 -lgmp
%: %.c
         $(CC) $(CPPFLAGS) $(CFLAGS) $(LDFLAGS) $< $(LIBS) -o $@
```

Voici le résultat obtenu à l'exécution. Le système initial est représenté sous la forme d'une « intersection » de trois chaînes régulières. Le terme *regchain* est la contraction de *regular chain*. Pourquoi « intersection » ? L'ensemble des solutions du système initial est l'union des ensembles de solutions des trois chaînes. Par conséquent, l'idéal défini par le système initial est l'intersection des idéaux définis par les trois chaînes. Les chaînes régulières ainsi que leur intersection sont affublées de qualificatifs qui décrivent des propriétés particulières des chaînes. Ces propriétés sont trop techniques pour être décrites dans cette section. Voir la section 5.6, page 84.

```
$ ./algebraic
```

```
intersectof_regchain ([regchain ([z - 1, x - y - 1], [autoreduced,
primitive, normalized]), regchain ([z + 1, x - y + 1], [autoreduced,
primitive, normalized ]), regchain ([z + 1, y + 1, x^2 - 4], [autoreduced,
primitive, normalized])], [autoreduced, primitive, normalized])
```

Commentons un peu le programme C. Les fonctions, les types et les structures de données des bibliothèques *BLAD* sont préfixées par *ba0*, *bav*, *bap* ou *bad* suivant la bibliothèque à laquelle elles appartiennent. La bibliothèque *ba0* gère la mémoire, les exceptions, les entrées–sorties et quelques structures de données génériques. La bibliothèque *bav* gère les variables (les indéterminées) et en particulier les ordres sur les variables. La bibliothèque *bap* contient tous les algorithmes qui sont censés s'appliquer à des polynômes (mais pas des systèmes de polynômes). La plus importante fonctionnalité qu'elle offre est le calcul du plus grand diviseur commun de deux polynômes en plusieurs indéterminées et à coefficients entiers. La bibliothèque *bad* contient les algorithmes qui s'appliquent à des systèmes de polynômes et en particulier, une implantation de *Rosenfeld–Gröbner*.

Tout appel aux bibliothèques *BLAD* doit être inclus dans ce qu'on appelle une « suite d'appels aux bibliothèques *BLAD* », qui commence par un appel à *bad_restart* et se termine par un appel à *bad_terminate*. Les paramètres fournis à *bad_restart* sont des limites en temps et en espace mémoire données pour les calculs (un zéro indique qu'on ne donne pas de limite). La variable M a un rôle lié à la gestion de la mémoire que je ne commente pas davantage ici.

La fonction *ba0_sscanf2* est une variante de la fonction *sscanf* offerte par la bibliothèque standard du langage C. Voici un petit rappel pour ceux qui auraient oublié leurs bases de C. En C, l'instruction suivante lit un entier sur l'entrée standard et affecte le résultat à la variable x. La chaîne de caractères `"%d"` qui constitue le premier paramètre de l'appel à *scanf* est un « format » qui décrit la nature de ce qui doit être lu. Le dernier paramètre est l'adresse de la variable destinée à recevoir le résultat.

```
{   int x;
    scanf ("%d", &x);
}
```

La fonction *scanf* admet plusieurs variantes. L'une d'elles est la fonction *sscanf* qui lit des données dans une chaîne de caractères plutôt que sur l'entrée standard. Ce premier paramètre est en première place. Dans l'exemple suivant, l'entier 421 est lu et affecté à x.

```
{   int x;
    sscanf ("421", "%d", &x);
}
```

La fonction *ba0_sscanf2* est construite sur le même principe que *sscanf* mais elle offre une variété de formats plus importante. L'extrait de code suivant lit l'ordre $x > y > z$ et l'affecte à la variable r

```
{   bav_Iordering r;
    ba0_sscanf2 ("ordering (derivations = [], blocks = [x, y, z])",
```

```
                "%ordering", &r);
}
```

Dans le programme, cet ordre est ensuite « empilé » (appel à *bav_R_push_ordering*) sur une pile prédéfinie d'ordres et devient l'ordre courant, c'est–à–dire celui qui est implicitement utilisé dans la suite des opérations.

L'extrait de code suivant lit un tableau de polynômes en plusieurs indéterminées et à coefficients entiers. Le format `"%Az"` désigne les polynômes en plusieurs indéterminées et à coefficients entiers. Le format `"%t[%Az]"` désigne les tableaux de tels polynômes. Le tableau *eqns* doit être initialisé avant de pouvoir être utilisé. Le schéma de programmation est calqué sur celui de la *GMP*.

```
{   struct bap_tableof_polynom_mpz eqns;

    ba0_init_table ((ba0_table)&eqns);
    ba0_sscanf2 ("[z^2 - 1, (z + 1)*(x - y - 1), (y + 1)*(x - y - z), \
                                x^2 - y^2 - 2*z*y - 1]", "%t[%Az]", &eqns);
}
```

L'extrait suivant effectue l'élimination en appelant *Rosenfeld–Gröbner*. Tout d'abord, la variable *ideal*, destinée à recevoir le résultat de l'élimination est initialisée. Puis on lui affecte une intersection « vide » de chaînes régulières. Le but est ici uniquement de positionner à vrai les indicateurs signifiant qu'on désire des chaînes « autoréduites », « primitives » et « fortement normalisées ». Enfin, *Rosenfeld–Gröbner* est appelé. Je ne commente pas les deux derniers paramètres de l'appel de fonction, qui ne sont pas utilisés.

```
{   struct bad_intersectof_regchain ideal;

    bad_init_intersectof_regchain (&ideal);
    ba0_sscanf2 ("intersectof_regchain ([], \
                        [autoreduced, primitive, normalized])",
                 "%intersectof_regchain", &ideal);
    bad_Rosenfeld_Groebner (&ideal, &eqns, &ineqns, (bav_tableof_variable)0);
}
```

1.3 L'élimination différentielle

Le principe de l'élimination algébrique et le concept de chaînes régulières se généralise au cas différentiel. Les indéterminées ne représentent plus des nombres mais des fonctions analytiques. Le système suivant est un exemple de système de deux équations différentielles polynomiales. Le « point » au–dessus d'une indéterminée désigne sa dérivée première (notation des « fluxions »). Une solution du système suivant est un couple $(u(t), v(t))$ de fonctions du temps.

$$\dot{u}^2 - 4\,\dot{v} = 0, \quad \dot{v} - u + 1 = 0.$$

L'élimination différentielle est un procédé qui prend en entrée un système d'équations polynomiales différentielles et un ordre sur l'ensemble infini de toutes les dérivées de u et de v. De tels ordres, lorsqu'ils satisfont certaines conditions techniques, sont appelés des « classements » (« rankings » en Anglais). Elle transforme le système fourni en entrée en une famille équivalente de « chaînes différentielles régulières ». Fixons par exemple l'ordre

$$\cdots > \ddot{u} > \dot{u} > u > \cdots > \ddot{v} > \dot{v} > v.$$

L'algorithme *Rosenfeld–Gröbner*, appliqué au système et à l'ordre ci–dessus produit la famille suivante de deux chaînes différentielles régulières :

$$\begin{array}{ll} \textit{première chaîne :} & \ddot{v}^2 - 4\,\dot{v} = 0, \quad u - \dot{v} - 1 = 0\ ; \\ \textit{seconde chaîne :} & \dot{v} = 0, \quad u - 1 = 0\ ; \end{array}$$

Dans la première équation de chaque chaîne l'indéterminée différentielle u et toutes ses dérivées sont éliminées. La famille est équivalente au système initial : toute solution analytique du système initial est une solution analytique de l'une des deux chaînes et réciproquement. Les solutions analytiques de la première chaîne sont

$$v(t) = c_1 + \frac{(t+c_2)^3}{3}, \quad u(t) = 1 + (t+c_2)^2, \quad c_1,\, c_2 \in \mathbb{C}.$$

Les solutions analytiques de la seconde chaîne sont

$$v(t) = c, \quad u(t) = 1, \quad c \in \mathbb{C}.$$

1.3.1 Élimination en MAPLE

J'ai réalisé le paquetage *diffalg* 1995–1996. Le paquetage a ensuite été fortement amélioré par plusieurs personnes et en particulier par Évelyne Hubert et (plus récemment) par François Lemaire. L'implantation de *Rosenfeld–Gröbner* est plus ancienne que celle de *BLAD*. Elle produit aussi des chaînes différentielles régulières bien que l'appellation « chaîne différentielle régulière » n'ai pas encore été inventée lors de la réalisation du paquetage (elle date de la thèse de François Lemaire en 2002).

L'ordre est défini au moment de la création de l'*ODE_ ring*. Les dérivées des indéterminées différentielles sont notées au moyen de listes de dérivations : par exemple, `u[t]` désigne $\dot{u}$ et `u[t,t]` désigne $\ddot{u}$. Pour contourner une limitation de MAPLE, on note `u[]` pour u.

```
> with (diffalg);
[Rosenfeld_Groebner, belongs_to, delta_leader, delta_polynomial, denote,

    derivatives, differential_ring, differential_sprem, differentiate,

    equations, essential_components, field_extension, greater, inequations,

    initial, initial_conditions, is_orthonomic, leader, power_series_solution,
```

```
    preparation_polynomial, print_ranking, rank, reduced, reduced_form,

    rewrite_rules, separant]

> R := differential_ring (derivations = [t], ranking = [u, v]);
                                  R := ODE_ring

> ideal := Rosenfeld_Groebner ([u[t]^2 - 4*v, v[t] - u + 1], R);
                 ideal := [characterizable, characterizable]

> equations (ideal [1]);
                                                    2
                      [-v[t] + u[] - 1, v[t, t]  - 4 v[]]

> equations (ideal [2]);
                                 [-1 + u[], v[]]
```

1.3.2 Élimination différentielle avec BLAD

Le programme C ci–dessous applique *Rosenfeld–Gröbner* sur l'exemple. Il est très proche du programme donné dans la section précédente. La principale différence est le qualificatif *differential* utilisé pour initialiser la variable *ideal*.

La bibliothèque *BLAD* offre en fait un concept unifié de *chaîne régulière* qui peut être ou non différentielle. Cette unification de concept, suggerée dans la thèse de François Lemaire, n'est pas encore réalisée dans les paquetages MAPLE.

```
/* File differential.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bad_intersectof_regchain ideal;
    struct bap_tableof_polynom_mpz eqns, ineqns;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);
    ba0_sscanf2 ("ordering (derivations = [t], blocks = [u, v])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bad_init_intersectof_regchain (&ideal);
    ba0_sscanf2 ("intersectof_regchain ([], \
                        [differential, autoreduced, primitive, normalized])",
```

```
                "%intersectof_regchain", &ideal);
    ba0_init_table ((ba0_table)&eqns);
    ba0_init_table ((ba0_table)&ineqns);
    ba0_sscanf2 ("[u[t]^2 - 4*v[t], v[t] - u + 1]", "%t[%Az]", &eqns);
    bad_Rosenfeld_Groebner (&ideal, &eqns, &ineqns, (bav_tableof_variable)0);
    ba0_printf ("%intersectof_regchain\n", &ideal);
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Le programme ci–dessus peut être compilé (voir une commande dans la section précédente). On obtient à l'exécution :

```
$ ./differential
intersectof_regchain ([regchain ([v[t,t]^2 - 4*v[t], u - v[t] - 1], [
differential, autoreduced, primitive, squarefree, coherent, normalized]),
regchain ([v[t], u - 1], [differential, autoreduced, primitive, squarefree,
coherent , normalized])], [differential, autoreduced, primitive, squarefree,
coherent, normalized])
```

1.3.3 Autres classements

Le classement donné ci–dessus est un « classement d'élimination ». Un autre type très important de classement est constitué des classements « compatibles avec l'ordre total » (« *orderly* » en Anglais). Vis–à–vis de ces classements, plus une indéterminée est dérivée (plus elle est d'ordre élevé) plus elle est grande. Voici un exemple de tel classement

$$\cdots > \ddot{u} > \ddot{v} > \dot{u} > \dot{v} > u > v.$$

En *BLAD*, il est noté (remarquer les deux niveaux de crochets) :

```
ordering (derivations = [t], blocks = [[u, v]])
```

Avec trois indéterminées différentielles ou plus, il est possible de définir des classements encore plus généraux, très utiles aussi. Par exemple,

$$\cdots > \ddot{w} > \dot{w} > w > \cdots > \ddot{u} > \ddot{v} > \dot{u} > \dot{v} > u > v.$$

Vis–à–vis de ce classement, toute dérivée de w est plus grande que toute dérivée de u ou de v alors que les dérivées de u et de v sont ordonnées entre elles suivant un classement compatible avec l'ordre total. On appelle cette sorte de classement un « classement d'élimination par blocs ». On le note souvent de façon abrégée :

$$w \gg (u, v).$$

On le note en *BLAD* :

```
ordering (derivations = [t], blocks = [w, [u, v]])
```

Chapitre 2

Arithmétique des polynômes différentiels

Ce chapitre présente des notions sans difficulté que je supposerai connues dans les chapitres suivants. Je l'ai émaillé de petits exercices (corrigés) et en particulier de petits exercices de programmation utilisant les bibliothèques *BLAD*.

2.1 Polynômes non différentiels

Soit K un corps commutatif et de caractéristique nulle. Soit X un alphabet, éventuellement infini, d'indéterminées sur K. On note $K[X]$ l'anneau des polynômes construits sur l'alphabet X et à coefficients dans K. On suppose X totalement ordonné et on considère un polynôme $f \in K[X] \setminus K$. On appelle « indéterminée principale » de f la plus grande des indéterminées $x \in X$ qui figure dans f c'est–à–dire telle que $\deg(f, x) \neq 0$. On la note $\operatorname{ld} f$. Notons $d = \deg(f, x)$. Le monôme x^d est appelé le « rang » de f. Le coefficient de x^d dans f est l' « initial » de f. Le polynôme $\partial f/\partial x$ est le « séparant » de f. Ce sont tous les deux des polynômes de $K[X]$.

Exercice 1 *Quel est l'initial du polynôme* $3\,x^2\,y - x^2 + y + 5\,x - y^3$ *(en supposant* $x > y$*) ? Écrire un programme C utilisant* BLAD *qui calcule et affiche l'initial du polynôme.*

Exercice 2 *Quel est le séparant du polynôme précédent ?*

Exercice 3 *Dans quel cas l'initial d'un polynôme est–il égal à son séparant ?*

Un ensemble fini de polynômes est dit « triangulaire » si les indéterminées principales de ses éléments sont toutes distinctes.

Soit $g \in K[X]$ un autre polynôme. Supposons que l'initial de f appartienne à K. On peut alors calculer le quotient q et le reste r de la division euclidienne de g par f, vus comme des polynômes en l'indéterminée x et à coefficients dans l'anneau $K[X \setminus \{x\}]$.

$$\begin{array}{l|l} g & f \\ \hline r & q \end{array}$$

Le couple (q, r) est défini de façon unique. Il satisfait

$$\deg(r, x) < \deg(f, x), \qquad g = f\,q + r.$$

On note $\mathrm{quo}(g, f, x)$ le quotient et $\mathrm{rem}(g, f, x)$ le reste ou parfois $\mathrm{quo}(g, f)$ et $\mathrm{rem}(g, f)$ lorsqu'aucune ambiguité n'est à craindre. Si l'initial de f n'appartient pas à K, la division euclidienne de g par f n'est plus possible : il n'y a aucune raison que l'initial de f divise exactement le coefficient principal de g par rapport à x. On dispose toutefois de l'algorithme de « pseudo–division » qui consiste à multiplier g par une puissance de l'initial de f suffisante pour que les divisions par l'initial de f tombent juste. En notant i_f l'initial de f et $a = \deg(f, x) - \deg(g, x) + 1$,

$$\begin{array}{r|l} i_f^a\, g & f \\ \hline r & q \end{array}$$

Exercice 4 *Effectuer la pseudo–division de $a\,x^2 + b\,x + c$ par $d\,x + e$. On suppose que l'indéterminée principale des deux polynômes est x. Écrire un programme C utilisant* BLAD *qui effectue la pseudo–division et affiche le résultat.*

Le couple (q, r) est défini de façon unique. Il satisfait

$$\deg(r, x) < \deg(f, x), \qquad i_f^a\, g = f\,q + r.$$

On note $\mathrm{pquo}(g, f, x)$ le pseudo–quotient et $\mathrm{prem}(g, f, x)$ le pseudo–reste de f par g ou parfois $\mathrm{pquo}(g, f)$ et $\mathrm{prem}(g, f)$ lorsqu'aucune ambiguité n'est à craindre. En fait, plusieurs variantes de l'algorithme de pseudo–division sont disponibles (on peut en particulier prendre parfois $a < \deg(f, x) - \deg(g, x) + 1$). Il peut être utile d'utiliser plusieurs variantes de l'algorithme de pseudo–division dans un logiciel : dans certains cas, on peut vouloir chercher l'exposant a le plus petit possible pour éviter le grossissement des données ; dans le contexte de l'algorithme des sous–résultants, on est obligé de prendre $a = \deg(f, x) - \deg(g, x) + 1$ pour que les divisions exactes mises en œuvre dans cet algorithme tombent juste. Pour couvrir tous les cas, nous dirons que le couple (q, r) *n'est pas* défini de façon unique mais qu'il satisfait dans tous les cas

$$\deg(r, x) < \deg(f, x), \qquad \exists\ a \geq 0,\ i_f^a\, g = f\,q + r.$$

Soit $F \subset K[X] \setminus K$ un ensemble ni nécessairement triangulaire ni même nécessairement fini. On définit $\mathrm{prem}(g, F)$ comme l'un des résultats possibles de l'algorithme suivant.

```
fonction prem(g, F)
début
  r := g
  tant que ∃ f ∈ F, deg(r, ld f) ≥ deg(f, ld f) faire
    r := prem(r, f, ld f)
  fait
  retourner r
fin
```

Sans précision supplémentaire, rien ne garantit que la fonction termine. Ces précisions (utilisation de « classements ») sont données dans les sections qui suivent. On suppose donc que le calcul termine. Le pseudo–reste $r = \mathrm{prem}(g, F)$ n'est pas nécessairement défini de façon unique mais il satisfait dans tous les cas les propriétés suivantes :

1. r est « algébriquement réduit » par rapport à F, c'est–à–dire que pour tous $f \in F$ on a $\deg(r, \mathrm{ld}\, f) < \deg(f, \mathrm{ld}\, f)$;
2. il existe un produit de puissances h d'initiaux d'éléments de F tel que $h\, g = r \mod (F)$ où (F) désigne l'idéal engendré par les éléments de F.

Un ensemble F dont tous les éléments sont algébriquement réduits les uns par rapport aux autres est dit « algébriquement autoréduit ».

Exercice 5 *Trouver un ordre sur l'ensemble* $\{x, y\}$ *tel que* $F = \{x^2, x\,y - 1\}$ *soit algébriquement autoréduit.*

Soit E un ensemble de polynômes différentiels n'appartenant pas à K (sauf peut–être zéro). Un sous–ensemble algébriquement autoréduit F de E est un « ensemble caractéristique algébrique » de E si E ne contient aucun élément non nul algébriquement réduit par rapport à F.

Exercice 6 *Montrer que si* F *est un ensemble caractéristique algébrique d'un idéal* E *alors, quel que soit* $f \in E$ *on a* $\mathrm{prem}(f, F) = 0$.

2.1.1 Saturations

Si $\mathfrak{A}$ est un idéal d'un anneau R et $M \subset R$ est une famille multiplicative, on définit la saturation de $\mathfrak{A}$ par M comme l'ensemble

$$\mathfrak{A} : M = \{f \in R \mid \exists\, m \in M,\ m\, f \in \mathfrak{A}\}.$$

Si $S = \{s_1, \ldots, s_t\}$ on note $S^\infty = \{s_1^{\ell_1} \cdots s_t^{\ell_t} \mid \ell_1, \ldots, \ell_t \geq 0\}$ la famille multiplicative engendrée par S.

Exercice 7 *Soit* $F = \{(x+1)\, y, x^2 - 1\}$ *un système. On prend l'ordre* $y > x$. *Décrire l'ensemble des solutions des idéaux* (F) *et* $(F) : I_F^\infty$. *Montrer que* F *n'est pas un ensemble caractéristique algébrique de l'idéal* $(F) : I_F^\infty$.

Exercice 8 *Montrer que si* $\mathrm{prem}(f, F) = 0$ *alors* $f \in (F) : I_F^\infty$ *où* I_F *désigne l'ensemble des initiaux des éléments de* F.

Exercice 9 *Montrer que* $\mathfrak{A} : M$ *est un idéal contenant* $\mathfrak{A}$.

Exercice 10 *Dans un anneau nœthérien, tout idéal* $\mathfrak{A}$ *est une intersection d'idéaux « primaires », c'est–à–dire d'idéaux* $\mathfrak{q}_1, \ldots, \mathfrak{q}_t$ *tels que*

$$a\, b \in \mathfrak{q}_i \Rightarrow [a \in \mathfrak{q}_i \quad ou \quad \exists\, n \geq 0,\ b^n \in \mathfrak{q}_i]\,.$$

Montrer que $\mathfrak{A} : M$ *est l'intersection des idéaux primaires* $\mathfrak{q}_i$ *tels que* $\mathfrak{q}_i \cap M = \emptyset$.

2.2 Polynômes différentiels

On appelle « dérivation » sur un anneau R toute application $\delta : R \to R$ satisfaisant les axiomes des dérivations :

$$\delta(a+b) = \delta(a) + \delta(b), \quad \delta(a\,b) = \delta(a)\,b + a\delta(b), \qquad (\forall\ a,\, b \in R)$$

Un « anneau différentiel » est un anneau muni d'un nombre fini de dérivations supposées commuter entre elles :

$$\delta_i(\delta_j(a)) = \delta_j(\delta_i(a)) \qquad (\forall\ a,\, b \in R).$$

De la même façon, un « corps différentiel » est un anneau différentiel qui est aussi un corps.

Exercice 11 *Montrer que les axiomes des dérivations impliquent que* $\delta(0) = 0$, $\delta(1) = 0$ *et plus généralement que* $\delta(z) = 0$ *pour tout* $z \in \mathbb{Z}$.

Exercice 12 *Soient* $a, b \in R$ *tels que* $1/b \in R$. *Montrer que les axiomes des dérivations impliquent :*

$$\delta\left(\frac{a}{b}\right) = \frac{\delta(a)\,b + a\,\delta(b)}{b^2}.$$

En algèbre différentielle, on manipule des systèmes finis d'un anneau de polynômes différentiels $R = K\{U\}$ muni d'un ensemble de dérivations $\{\delta_1, \ldots, \delta_m\}$. L'ensemble $U = \{u_1, \ldots, u_n\}$ est l'ensemble des « indéterminées différentielles ». L'anneau des coefficients K est un corps différentiel commutatif de caractéristique nulle. L'ensemble $\{\delta_1, \ldots, \delta_m\}$ engendre un monoïde commutatif pour l'opération de composition. On le note

$$\Theta = \{\delta_1^{a_1} \cdots \delta_m^{a_m} \mid a_1, \ldots, a_m \in \mathbb{N}\}.$$

Les éléments de Θ sont les « opérateurs de dérivation ». Dans cette théorie, on se restreint donc à des opérateurs de dérivation commutatifs. Si $\theta = \delta_1^{a_1} \cdots \delta_m^{a_m}$ est un opérateur de dérivation, on définit son « ordre » comme

$$\operatorname{ord}\theta = a_1 + \cdots + a_m.$$

Si $\theta = \delta_1^{a_1} \cdots \delta_m^{a_m}$ et $\varphi = \delta_1^{b_1} \cdots \delta_m^{b_m}$ sont deux opérateurs de dérivation, on note $\theta\varphi = \delta_1^{a_1+b_1} \cdots \delta_m^{a_m+b_m}$. Si de plus $a_i \geq b_i$ pour tous $1 \leq i \leq m$, on note $\theta/\varphi = \delta_1^{a_1-b_1} \cdots \delta_m^{a_m-b_m}$. Le monoïde Θ agit sur l'ensemble des indéterminées différentielles U, engendrant l'ensemble infini ΘU des « dérivées ». Les polynômes différentiels de R peuvent ainsi être vus comme des polynômes au sens habituel sur l'alphabet infini des dérivées : $R = K[\Theta U]$.

Un « classement » est une relation d'ordre total sur ΘU compatible avec l'action des dérivations sur ΘU, c'est–à–dire qui satisfait les deux axiomes suivants :

1. $v \leq \theta v \qquad (\forall\ v \in \Theta U,\ \theta \in \Theta)$
2. $v < w \Rightarrow \theta v < \theta w \qquad (\forall\ v,\, w \in \Theta U,\ \theta \in \Theta)$

On distingue plusieurs types de classements. Les trois qui suivent sont importants bien que nous ne devions pas nous en servir dans ce chapitre. Les définitions sont données pour deux indéterminées différentielles. Elles se généralisent aisément à un nombre quelconque d'indéterminées différentielles.

1. Les classements d'élimination.
   Soient u, $v \in U$ deux indéterminées différentielles. On dit qu'un classement « élimine » u, ce qu'on note
   $$u \gg v \quad \text{si} \quad \theta u > \phi v \qquad (\forall\ \theta,\ \phi \in \Theta).$$
2. Les classements compatibles avec l'ordre total.
   Soient u, $v \in U$ deux indéterminées différentielles. On dit qu'un classement est « compatible avec l'ordre total » pour u et v si
   $$\operatorname{ord}\theta < \operatorname{ord}\phi \Rightarrow \theta u < \phi v \qquad (\forall\ \theta,\ \phi \in \Theta).$$
3. Les classements de Riquier.
   Soient u, $v \in U$ deux indéterminées différentielles. On dit qu'un classement est « de Riquier » pour u et v si
   $$\theta u < \phi u \Leftrightarrow \theta v < \phi v \qquad (\forall\ \theta,\ \phi \in \Theta).$$

Exercice 13 *Pour chacun des extraits de classements suivants, indiquer s'il relève de l'un des types précédents. Les indéterminées différentielles sont u et v. les dérivations δ_x et δ_y sont notées en indice.*

1. $\cdots > u_{xy} > u_{yy} > u_x > u_y > u > \cdots > v_{xx} > v_{xy} > v_{yy} > v_x > v_y > v.$
2. $\cdots > u_{xx} > u_{xy} > u_{yy} > v_{xx} > v_{xy} > v_{yy} > u_x > u_y > v_x > v_y > u > v.$
3. $\cdots > u_{xx} > u_{xy} > u_{yy} > v_{yy} > v_{xy} > v_{xx} > u_x > u_y > v_y > v_x > u > v.$

Supposons ΘU ordonné suivant un classement. Toutes les définitions données dans la section précédente s'appliquent aux polynômes différentiels, si ce n'est qu'on dit « dérivée dominante » au lieu d'indéterminée principale.

Exercice 14 *Donner, vis–à–vis de chacun des classements ci–dessus, la dérivée dominante du polynôme différentiel $u_y\, v_{yy} + u_x^2 - 3$. Écrire un programme C, utilisant* BLAD*, qui calcule et affiche ces dérivées dominantes.*

Proposition 1 *Soient $f \in R \setminus K$ un polynôme différentiel et $\theta \in \Theta$ un opérateur de dérivation d'ordre non nul. Le séparant de f est l'initial de θf.*

Preuve C'est une conséquence des axiomes des classements. □

2.2.1 La réduction de Ritt

L'appellation vient du nom du véritable fondateur de l'algèbre différentielle : Joseph Fels Ritt. Soient $g \in R$ un polynôme différentiel et $F \subset R$ un ensemble fini de polynômes différentiels. On définit le « reste partiel » de g par F pour la réduction de Ritt par

$$\text{reste\_partiel}(g, F) = \text{prem}(g, \{\Theta F\} \setminus F).$$

On définit le « reste complet » de g par F pour la réduction de Ritt par

$$\text{reste\_complet}(g, F) = \text{prem}(g, \Theta F).$$

Les remarques faites au sujet de la pseudo–division s'appliquent *a fortiori* pour la réduction de Ritt : « le » reste partiel et « le » reste complet de g par F pour la réduction de Ritt ne sont pas définis de façon unique. Si on les programme naturellement, c'est–à–dire en cherchant à réduire d'abord les dérivées les plus grandes vis–à–vis du classement, ces algorithmes de réduction terminent. On le démontre en la section 2.4. Notons $[F]$ le plus petit « idéal différentiel » engendré par F dans R, c'est–à–dire le plus petit idéal de R contenant F et stable sous l'action des dérivations (c'est–à–dire tel que $p \in [F] \Rightarrow \theta p \in [F]$ pour toute dérivée θp de p). Le polynôme différentiel $r = \text{reste\_partiel}(g, F)$ satisfait les propriétés :

1. il est « partiellement réduit » par rapport à F, c'est–à–dire algébriquement réduit par rapport à $\{\Theta F\} \setminus F$;
2. il existe un produit de puissances h de séparants d'éléments de F tels que $h\, g = r \mod [F]$.

Le polynôme différentiel $r = \text{reste\_complet}(g, F)$ satisfait les propriétés :

1. il est « complètement réduit » par rapport à F, c'est–à–dire algébriquement réduit par rapport à ΘF ;
2. il existe un produit de puissances h d'initiaux et de séparants d'éléments de F tels que $h\, g = r \mod [F]$.

Exercice 15 *Calculer le reste partiel et le reste complet de* $g = u_{xx} + u_x$ *par* $F = \{u_x^2 + u_x + u\}$. *Écrire un programme C, utilisant* BLAD, *qui calcule et affiche ces deux restes.*

Exercice 16 *Montrer que* $u_x^3 \in [u\, u_x]$.

Un ensemble F dont tout élément est partiellement réduit par rapport à tous les autres est dit « partiellement autoréduit ». Un ensemble F dont tout élément est complètement réduit par rapport à tous les autres est dit « complètement autoréduit ». Soit $E \subset R$ un ensemble de polynômes différentiels n'appartenant pas à K (sauf peut–être zéro). Un sous–ensemble complètement autoréduit F de E est un « ensemble caractéristique différentiel » de E si E ne comporte aucun élément non nul complètement réduit par rapport à F.

Exercice 17 *Montrer que si* F *est un ensemble caractéristique différentiel d'un idéal différentiel* E *alors, quel que soit* $f \in E$ *on a* $\text{reste\_complet}(f, F) = 0$.

Exercice 18 *Notons* S_F *l'ensemble des séparants des éléments de* F *et* H_F *l'ensemble des initiaux et des séparants des éléments de* F. *Montrer que si* $\text{reste\_partiel}(f, F) = 0$ *alors* $f \in [F] : S_F^\infty$. *Montrer que si* $\text{reste\_complet}(f, F) = 0$ *alors* $f \in [F] : H_F^\infty$.

2.3 Correction des exercices

Corrigé de l'exercice 1. L'initial est $3\,y - 1$. Voici un programme C calculant l'initial.

La séquence d'appels aux bibliothèques *BLAD* est encadrée par des appels à *bap_ restart* et à *bap_ terminate* au lieu de *bad_ restart* et *bad_ terminate*. Cela indique que le programme utilise pas les bibliothèques *ba0*, *bav* et *bap* mais pas la bibliothèque *bad*.

Pourquoi la variable *initial* est–elle initialisée en tant que polynôme « *readonly* », c'est–à–dire en tant que polynôme « en lecture seule » ? Ce n'est pas indispensable mais c'est plus efficace. La fonction *bap_ initial_ polynom_ mpz* ne construit pas vraiment un nouveau polynôme. Elle affecte à la variable *initial* un polynôme qui partage des données avec A. Par sécurité, cette fonction positionne un indicateur dans la variable *initial* interdisant que le contenu de cette variable soit modifié (sans cela, toute modification faite sur *initial* modifierait A par « effet de bord »). Sachant cela, il est possible d'optimiser le code en initialisant *initial* de façon « incomplète ». Cette sorte d'initialisation est réalisée par *bap_ init_ readonly_ polynom_ mpz*.

```
/* File exercice_1.c */

#include "bap.h"

int main ()
{   bav_Iordering r;
    struct bap_polynom_mpz A, initial;
    struct ba0_mark M;

    bap_restart (0, 0);
    ba0_record (&M);

    ba0_sscanf2 ("ordering (derivations = [], blocks = [x, y])",
                 "%ordering", &r);
    bav_R_push_ordering (r);

    bap_init_polynom_mpz (&A);
    ba0_sscanf2 ("3*x^2*y - x^2 + y + 5*x - y^3", "%Az", &A);
    bap_init_readonly_polynom_mpz (&initial);
    bap_initial_polynom_mpz (&initial, &A);
    ba0_printf ("initial = %Az\n", &initial);

    bav_R_pull_ordering ();
    ba0_restore (&M);
    bap_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu à l'exécution.

```
$ ./exercice_1
initial = 3*y - 1
```

□

Corrigé de l'exercice 2. Le séparant est $6\,x\,y - 2\,x + 5$. □

Corrigé de l'exercice 3. Quand le polynôme est de degré 1 en son indéterminée principale. □

Corrigé de l'exercice 4. Avant d'effectuer la division, on multiplie le polynôme à diviser par d^2. Le pseudo–quotient est $d\,a\,x + (d\,b - a\,e)$; le pseudo–reste $d^2\,c - d\,b\,e + a\,e^2$. Voici comment procéder en *BLAD*.

```
/* File exercice_4.c */

#include "bap.h"

int main ()
{   bav_Iordering r;
    bav_Idegree deg;
    struct bap_polynom_mpz A, B, Q, R;
    struct ba0_mark M;

    bap_restart (0, 0);
    ba0_record (&M);

    ba0_sscanf2 ("ordering (derivations = [], blocks = [x, a, b, c, d, e])",
                 "%ordering", &r);
    bav_R_push_ordering (r);

    bap_init_polynom_mpz (&A);
    bap_init_polynom_mpz (&B);
    ba0_sscanf2 ("a*x^2 + b*x + c", "%Az", &A);
    ba0_sscanf2 ("d*x + e", "%Az", &B);
    bap_init_polynom_mpz (&Q);
    bap_init_polynom_mpz (&R);

    bap_pseudo_division_polynom_mpz (&Q, &R, &deg, &A, &B);
    ba0_printf ("Q = %Az, R = %Az, deg = %d\n", &Q, &R, deg);

    bav_R_pull_ordering ();
    ba0_restore (&M);
    bap_terminate (ba0_init_level);
    return 0;
}
```

Voici le résultat obtenu à l'exécution.

```
$ ./exercice_4
Q = x*a*d - a*e + b*d, R = a*e^2 - b*d*e + c*d^2, deg = 2
```

□

Corrigé de l'exercice 5. Prendre $y > x$. □

Corrigé de l'exercice 6. Soient $f \in E$ et $r = \mathrm{prem}(f, F)$. D'après les spécifications de prem, le pseudo–reste r est algébriquement réduit par rapport à F et appartient à E. D'après la définition d'un ensemble caractéristique, $r = 0$. □

Corrigé de l'exercice 7. L'ensemble des initiaux est $I_F = \{x+1,\, 1\}$. L'idéal (F) admet pour solutions la droite $x = -1$ et le point $(x, y) = (1, 0)$. L'idéal $(F) : I_F^\infty$ n'admet que le point pour solution. Comme $(x+1)(x-1) \in (F)$ on a $x-1 \in (F) : I_F^\infty$. Le polynôme $x-1$ est non nul et algébriquement réduit par rapport à F donc F n'est pas un ensemble caractéristique algébrique de $(F) : I_F^\infty$. □

Corrigé de l'exercice 8. Si $\mathrm{prem}(f, F) = 0$ alors, d'après les spécifications de prem, il existe un élément h de la famille multiplicative engendrée par I_F tel que $h\, f \in (F)$. Par conséquent, $f \in (F) : I_F^\infty$. □

Corrigé de l'exercice 9. Toute famille multiplicative contient 1. Par conséquent, $\mathfrak{A} \subset \mathfrak{A} : M$. Montrons que $\mathfrak{A} : M$ est un idéal c'est–à–dire un ensemble stable par addition interne et par multiplication par un élément R. Si a, $a' \in \mathfrak{A} : M$ alors il existe m, $m' \in M$ tels que $m\, a$, $m'\, a' \in \mathfrak{A}$. D'une part $m\, m'\, (a + a') \in \mathfrak{A}$, d'autre part $m\, m' \in M$ et donc $a + a' \in \mathfrak{A} : M$. Si $a \in \mathfrak{A} : M$ et $r \in R$ alors il existe $m \in M$ tel que $m\, a \in \mathfrak{A}$. Par conséquent $m\, r\, a \in \mathfrak{A}$ et $r\, a \in \mathfrak{A} : M$. □

Corrigé de l'exercice 10.

Supposons que $\mathfrak{q}_i \cap M = \emptyset$ pour $1 \leq i \leq s$ et que $\mathfrak{q}_i \cap M \neq \emptyset$ pour $s < i \leq t$. On doit montrer que

$$\mathfrak{A} : M = \mathfrak{q}_1 \cap \cdots \cap \mathfrak{q}_s.$$

L'inclusion $\subset$ de gauche à droite. Si $a \in \mathfrak{A} : M$ alors il existe $m \in M$ tel que $m\, a \in \mathfrak{A}$. Soit $1 \leq i \leq s$ un indice. D'une part $m\, a \in \mathfrak{q}_i$, d'autre part, aucune puissance de m n'appartient à $\mathfrak{q}_i$. D'après la définition des idéaux primaires, $a \in \mathfrak{q}_i$.

L'inclusion $\supset$ de droite à gauche. Soit $a \in \mathfrak{q}_i$ pour $1 \leq i \leq s$. Pour $s < j \leq t$, il existe $m_j \in M$ tel que $m_j \in \mathfrak{q}_j$. Le produit m des m_j pour $s < j \leq t$ appartient donc à tous les idéaux $\mathfrak{q}_j$ (pour $s < j \leq t$). Par conséquent, $m\, a \in \mathfrak{q}_k$ pour $1 \leq k \leq t$ c'est–à–dire à $\mathfrak{A}$ et donc $a \in \mathfrak{A} : M$. □

Corrigé de l'exercice 11. D'après le premier axiome, $\delta(0) = \delta(0 + 0) = 2\,\delta(0)$. D'après le deuxième axiome, $\delta(1) = \delta(1 \times 1) = 2\,\delta(1)$. Pour $z \notin \{0, 1\}$, on peut procéder par récurrence. □

Corrigé de l'exercice 12.

On a $(a/b) \times b = a$ et donc $\delta((a/b) \times b) = \delta(a/b) \times b + (a/b) \times \delta(b) = \delta(a)$. □

Corrigé de l'exercice 13.

1. Ce classement élimine u. C'est aussi un classement de Riquier. Les dérivées de u sont ordonnées suivant un classement compatible avec l'ordre total, de même que les dérivées de v.
2. Ce classement est à la fois compatible avec l'ordre total et de Riquier.
3. Ce classement est compatible avec l'ordre total mais n'est pas de Riquier.

□

Corrigé de l'exercice 14. Respectivement u_x, v_{yy} et v_{yy}.

La syntaxe des classements utilisée aussi bien dans le paquetage *diffalg* que dans les bibliothèques *BLAD* permet de définir les deux premiers classements mais pas le troisième. Le programme C suivant extrait la dérivée dominante (le « *leader* ») du polynôme vis–à–vis des deux premiers classements et l'affiche à l'écran.

```
/* File exercice_14.c */

#include "bap.h"

int main ()
{   bav_Iordering r;
    bav_variable v;
    struct bap_polynom_mpz A;
    struct ba0_mark M;

    bap_restart (0, 0);
    ba0_record (&M);
/* First ranking */
    ba0_sscanf2 ("ordering (derivations = [x, y], blocks = [u, v])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bap_init_polynom_mpz (&A);
    ba0_sscanf2 ("u[y]*v[y,y] + u[x]^2 - 3", "%Az", &A);
    v = bap_leader_polynom_mpz (&A);
    ba0_printf ("leader = %v\n", v);
    bav_R_pull_ordering ();
/* Second ranking */
    ba0_sscanf2 ("ordering (derivations = [x, y], blocks = [[u, v]])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bap_sort_polynom_mpz (&A, &A);
    v = bap_leader_polynom_mpz (&A);
    ba0_printf ("leader = %v\n", v);
```

```
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bap_terminate (ba0_init_level);
    return 0;
}
```

Quelle est l'utilité de l'appel à *bap_ sort_ polynom_ mpz*? La quasi totalité des fonctions de manipulation de polynômes supposent que les monômes qui constituent les polynômes sont ordonnés en fonction du classement courant. C'est le cas de *bap_ leader_ polynom_ mpz*. Les monômes qui constituent le polynôme A sont naturellement ordonnés en fonction du premier classement, qui est le classement courant au moment de la création de ce polynôme. Pour pouvoir obtenir la dérivée dominante de A vis–à–vis du deuxième classement, il suffit donc de réordonner les monômes de A avant d'appeler à nouveau *bap_ leader_ polynom_ mpz*. Ce travail est réalisé par *bap_ sort_ polynom_ mpz*.

Voici l'affichage obtenu en exécutant ce programme :

```
$ ./exercice_14
leader = u[x]
leader = v[y,y]
```

□

Corrigé de l'exercice 15.

Notons $f = u_x^2 + u_x + u$. Sa dérivée dominante est u_x quel que soit le classement. Le polynôme g n'est pas partiellement réduit par rapport à f puisqu'une dérivée propre $u_x x$ de la dérivée dominante de f y figure avec un coefficient non nul. Pour calculer le reste partiel de g par f, on commence par dériver f suivant δ_x, ce qui donne

$$f_x = (2\,u_x + 1)\,u_{xx} + u_x.$$

Le polynôme $s_f = 2\,u_x + 1$ est à la fois le séparant de f et l'initial de f_x. On effectue la pseudo–division de g par f_x, vus comme des polynômes en u_{xx} ce qui donne un premier reste

$$r = \mathrm{prem}(g, f_x) = 2\,u_x^2$$

et une relation

$$s_f\,g = r \mod (f_x).$$

Le polynôme r est partiellement réduit par rapport à f. Il s'agit donc du reste partiel de g par f. Il n'est toutefois pas complètement réduit par rapport à f. On le pseudo–réduit par f, tous deux vus comme des polynômes en u_x, ce qui donne un deuxième reste

$$r' = \mathrm{prem}(r, f) = -2\,u_x - 2\,u$$

et une relation

$$s_f\,g = r' \mod (f,\, f_x)$$

ou encore, en notant $[f]$ l'idéal différentiel engendré par f,

$$s_f\, g = r' \mod [f].$$

Le polynôme r' est complètement réduit par rapport à f. Il s'agit donc du reste complet de g par f. Voici une façon de procéder avec *BLAD*.

L'interface des bibliothèques ne fournit pas d'implantation de l'algorithme de réduction de Ritt par un ensemble quelconque. On dispose par contre dans *bad* de la réduction de Ritt par une « chaîne régulière ». Cette notion est définie dans un autre chapitre. Il suffit ici de savoir que tout polynôme (différentiel) constitue une chaîne régulière (différentielle).

La fonction *bad_reduce_polynom_by_regchain* attend six paramètres. Les deux premiers sont destinés à recevoir le reste R et le produit de puissances H d'initiaux et de séparants d'éléments de la chaîne utilisés lors de la réduction. Ce ne sont pas des polynômes mais des « produits » c'est–à–dire des polynômes sous forme factorisée : les implantations les plus efficaces de l'algorithme sont en effet conçues pour détecter certaines factorisations et pour effectuer les réductions facteur par facteur. Le troisième paramètre est le polynôme A à réduire. Le quatrième, la chaîne avec laquelle réduire. Le cinquième est un élément d'un type énuméré indiquant si la réduction doit être partielle ou complète. Le sixième est un élément d'un type énuméré indiquant si on souhaite ou non que la dérivée dominante de A soit réduite.

```
/* File exercice_15.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bad_regchain C;
    struct bap_product_mpz R;
    struct bap_polynom_mpz A;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);

    ba0_sscanf2 ("ordering (derivations = [x], blocks = [u])",
                 "%ordering", &r);
    bav_R_push_ordering (r);

    bad_init_regchain (&C);
    ba0_sscanf2 ("regchain ([u[x]^2 + u[x] + u], [differential])",
                 "%regchain", &C);

    bap_init_polynom_mpz (&A);
    ba0_sscanf2 ("u[x,x] + u[x]", "%Az", &A);
```

```
    bap_init_product_mpz (&R);

    bad_reduce_polynom_by_regchain
        (&R, (bap_product_mpz)0, &A, &C, bad_partial_reduction,
         bad_all_derivatives_to_reduce);
    ba0_printf ("reste partiel = %Pz\n", &R);

    bad_reduce_polynom_by_regchain
        (&R, (bap_product_mpz)0, &A, &C, bad_full_reduction,
         bad_all_derivatives_to_reduce);
    ba0_printf ("reste complet = %Pz\n", &R);

    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici le résultat obtenu en exécutant le programme :

```
$ ./exercice_15
reste partiel = 2*u[x]^2
reste complet = 2*(-u[x] - u)
```

□

Corrigé de l'exercice 16.

La dérivée $u\,u_{xx} + u_x^2$ de $u\,u_x$ appartient à l'idéal différentiel $[u\,u_x]$. On trouve

$$u_x\,(u\,u_{xx} + u_x^2) - u_{xx}\,(u\,u_x) = u_x^3 \in [u\,u_x].$$

□

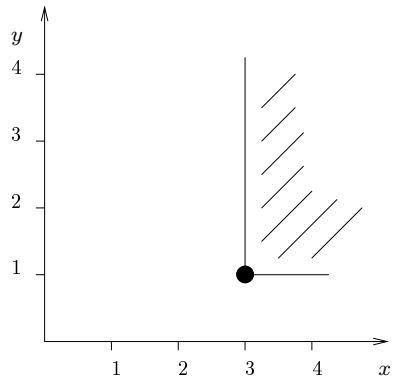
Corrigé de l'exercice 17. La démonstration est essentiellement la même que dans le cas algébrique (exercice 6). □

Corrigé de l'exercice 18 La démonstration est essentiellement la même que dans le cas algébrique (exercice 8). □

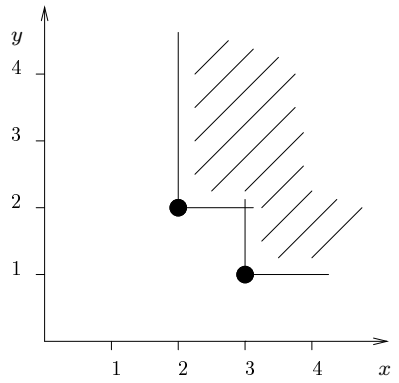
2.4 Finitude

Les démonstrations de presque tous les résultats de « finitude » en algèbre différentielle (le fait que les classements soient des bons ordres, que les ensembles caractéristiques et les chaînes différentielles régulières soient finies, que les algorithmes s'arrêtent) peuvent se réduire à la démonstration de la finitude du « mécanisme de complétion » suivant.

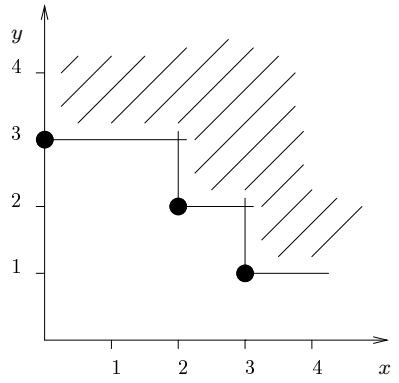
Commençons par un exemple et supposons qu'il n'y ait qu'une seule indéterminée différentielle u et deux dérivations δ_x et δ_y. On peut représenter l'ensemble des dérivées de u sur un diagramme en deux dimensions : l'axe des x pour la dérivation δ_x, l'axe des y pour la dérivation δ_y. Toute dérivée $\delta_x^{a_1}\,\delta_y^{b_1}\,u$ est représentée par le point de coordonnées (a_1, b_1). On hachure l'ensemble de ses dérivées. Avec $a_1 = 3$ et $b_1 = 1$ on obtient :



Considérons maintenant une deuxième dérivée $\delta_x^{a_2}\,\delta_y^{b_2}\,u$ dont le point associé se trouve dans la partie non hachurée du diagramme. Reportons la et hachurons l'ensemble des dérivées des deux dérivées. On peut prendre par exemple $a_2 = 2$ et $b_2 = 2$, ce qui donne



Considérons une troisième dérivée $\delta_x^{a_3}\,\delta_y^{b_3}\,u$ dont le point associé se trouve à nouveau dans la partie non hachurée du diagramme. Reportons la et hachurons l'ensemble des dérivées des trois dérivées. En prenant $a_3 = 0$ et $b_3 = 3$ on obtient :



On « voit bien » qu'on ne pourra pas continuer ainsi indéfiniment. On peut remarquer qu'à cette étape–ci, on ne peut pas borner le nombre d'étapes qu'il est encore possible d'effectuer mais, qu'après au plus cinq étapes les points situés dans la partie non hachurée du diagramme seront en nombre fini.

On peut généraliser ce mécanisme de complétion au cas d'une indéterminée différentielle et de m dérivations. Algébriquement, on considère une suite $(\theta_i u)$ de dérivées

$$\theta_1 u,\ \theta_2 u,\ \theta_3 u,\ \ldots$$

telle que pour tout indice i, la dérivée $\theta_i u$ n'est la dérivée d'aucune des dérivées qui la précèdent.

Le lemme suivant est équivalent au lemme de Dickson [31].

Lemme 1 *La suite $(\theta_i u)$ est finie.*

Preuve Par récurrence sur le nombre m de dérivations.

La base $m = 1$ est triviale.

Le cas général. Hypothèse de récurrence : toute suite vérifiant la même condition que $(\theta_i u)$ pour $m-1$ dérivations est finie. À tout opérateur $\theta = \delta_1^{a_1} \cdots \delta_m^{a_m}$ on associe l'opérateur « tronqué » $\phi = \delta_1^{a_1} \cdots \delta_{m-1}^{a_{m-1}}$.

On tient un raisonnement par l'absurde en utilisant le fait que toute suite infinie d'entiers naturels contient une sous–suite infinie croissante. Supposons que la suite $(\theta_i u)$ soit infinie. Elle contient alors une sous–suite $(\theta_j u)$ pour $j \in J$ infini, telle que l'exposant de δ_m soit croissant. La suite $(\phi_j u)$ pour $j \in J$ des opérateurs tronqués est alors infinie, vérifie la même condition que $(\theta_i u)$ mais pour $m-1$ dérivations.

Cette contradiction avec l'hypothèse de récurrence prouve le lemme. □

La preuve précédente n'est pas constructive. Une preuve constructive du lemme de Dickson a été récemment mise au point par Hervé Perdry dans sa thèse [81]. Ce résultat se généralise immédiatement au cas de n indéterminées différentielles et m dérivations. Graphiquement, il faut représenter ΘU par n diagrammes en m dimensions. Algébriquement, on considère une suite (v_i) de dérivées

$$v_1,\ v_2,\ v_3,\ \ldots$$

telle que pour tout indice i la dérivée v_i n'est la dérivée d'aucune des dérivées qui la précèdent. Parce que le nombre d'indéterminées différentielles est fini, si cette suite était infinie, elle contiendrait une sous–suite infinie composée de dérivées d'une seule indéterminée différentielle[1], ce qui est impossible d'après le lemme.

Proposition 2 *Tout classement est un bon ordre*[2].

[1] D'après le théorème de Ramsey, dont la formulation la plus élémentaire est : un ensemble infini de jetons soit verts soit jaunes contient une infinité de jetons verts ou une infinité de jetons jaunes.

[2] Un « bon ordre » sur un ensemble E est une relation d'ordre telle que toute suite strictement décroissante d'éléments de E soit finie.

Preuve C'est un corollaire immédiat du lemme. En effet, si (v_i) est une suite strictement décroissante de dérivées pour un classement alors, d'après le premier axiome des classements, toute dérivée v_i n'est la dérivée d'aucune des dérivées qui la précèdent. □

Proposition 3 *Tout ensemble complètement autoréduit de dérivées est fini.*

Preuve C'est un corollaire immédiat du lemme.

Énumérons les éléments d'un ensemble complètement autoréduit de dérivées suivant un ordre quelconque (par exemple d'après un classement). On obtient une suite (v_i) de dérivées telle que, pour tout indice i, la dérivée v_i n'est la dérivée d'aucune de celles qui la précèdent. □

Proposition 4 *Tout ensemble caractéristique différentiel est fini.*

Preuve L'ensemble des dérivées dominantes d'un ensemble caractéristique différentiel est un ensemble complètement autoréduit de dérivées. □

Proposition 5 *Pour peu qu'on les programme « naturellement », les algorithmes de réduction de Ritt terminent.*

Preuve Programmer naturellement les algorithmes de réduction de Ritt signifie qu'à chaque étape on cherche à réduire, dans le reste courant, la plus grande dérivée possible. La suite de ces dérivées est strictement décroissante et donc finie. □

Plusieurs algorithmes de complétion en algèbre différentielle sont amenés à construire une suite d'ensembles de polynômes (A_i) dont l'ensemble des dérivées dominantes est autoréduit. L'ensemble A_{i+1} s'obtient à partir de A_i et d'un polynôme différentiel p_i dont le rang est réduit par rapport à A_i par une opération du type :

$$A_{i+1} = \{p_i\} \cup \{q \in A_i \mid \mathrm{ld}\, q \text{ n'est pas une dérivée de } \mathrm{ld}\, p_i\}.$$

Proposition 6 *Toute suite (A_i) ainsi construite est finie.*

Preuve C'est un corollaire immédiat du lemme. Formons la suite $(v_i^{d_i})$ des rangs des polynômes p_i et supposons la infinie. Comme un degré ne peut pas décroître indéfiniment, elle contient une sous–suite de rangs $(v_j^{d_j})$ pour $j \in J$ infini dont les dérivées sont toutes distinctes. Pour tout $j \in J$, la dérivée v_j n'est la dérivée d'aucune des dérivées qui la précèdent. □

Chapitre 3

Solutions d'un système différentiel

Dans tous les cours que j'ai assurés, j'ai rencontré la difficulté pédagogique d'expliquer ce qu'est une solution d'un système de polynômes différentiels Σ en n indéterminées différentielles, m dérivations et à coefficients dans un corps différentiel (mettons $\mathbb{Q}$ pour fixer les idées). Cette difficulté constitue peut–être le principal obstacle à la popularisation de la théorie : l'algèbre différentielle.

Selon cette théorie en effet, une solution n'est pas la donnée de n fonctions de m variables mais de n quantités, appartenant à une extension de corps différentielle du corps des coefficients de Σ et qui annule le système (ce que j'appellerai par la suite une « solution abstraite » de Σ).

Ce point de vue est en fait l'analogue du point de vue classique pour les systèmes polynomiaux usuels (une solution d'un système de polynômes en n indéterminées et à coefficients rationnels est un n–uplet de valeurs appartenant à une extension de corps finie de $\mathbb{Q}$) mais il est beaucoup moins intuitif. Pour les systèmes polynomiaux usuels, on peut poser qu'on cherche des solutions dans le corps des nombres complexes (puisque toute extension de corps finie de $\mathbb{Q}$ s'injecte dans $\mathbb{C}$). On ne dispose pas dans le cas différentiel d'un analogue du corps des nombres complexes qui soit aussi « familier » pour la grande majorité des scientifiques que ne l'est $\mathbb{C}$.

Dans son livre fondateur [86], Joseph Ritt ne néglige pas du tout le point de vue traditionnel (à la différence d'Ellis Robert Kolchin [56]). Il présente dans l'ordre, les systèmes de polynômes différentiels, la théorie des idéaux de polynômes différentiels, les solutions abstraites et fait une correspondance rapide avec ce qu'il appelle le « cas analytique » c'est–à–dire des solutions sous la forme de fonctions méromorphes : il écrit page 23 de son livre que les définitions données et les raisonnements tenus pour les solutions abstraites restent valables dans le « cas analytique ».

Dans le chapitre qui suit, je vais tenter une présentation un peu différente de celle de Joseph Ritt. Une démarche d'informaticien, peut–être plus pédagogique : plutôt que de définir *a priori* ce qu'est une solution et d'en déduire les manipulations algorithmiques qu'elle autorise, je pars des manipulations algorithmiques qu'on souhaite pouvoir effectuer et j'en déduis quelles conditions toute définition de solution doit satisfaire. Je montre ensuite qu'au moins trois définitions conviennent : celle des solutions abstraites, celle des solutions en série

formelle et celle des solutions analytiques. Pour cette dernière, je m'appuie sur un résultat récent dû à François Lemaire (bien que la chose soit connue depuis les travaux de Ritt et de Seidenberg).

Ce chapitre doit beaucoup aux textes d'Abraham Seidenberg [94, 96, 97] qui a éprouvé lui aussi le besoin de revenir sur les explications données par Joseph Ritt.

3.1 Deux règles d'inférence et un théorème

On se donne un sous–ensemble fini Σ d'un anneau de polynômes différentiels $R = K\{u_1, \ldots, u_n\}$ muni de m dérivations abstraites $\delta_1, \ldots, \delta_m$, c'est–à–dire m opérations unaires satisfaisant les axiomes des dérivations :

$$\delta(a+b) = \delta(a) + \delta(b), \quad \delta(a\,b) = \delta(a)\,b + a\delta(b), \qquad (\forall\ a,\, b \in R)$$

supposées commuter entre elles :

$$\delta_i(\delta_j(a)) = \delta_j(\delta_i(a)) \qquad (\forall\ a,\, b \in R).$$

L'anneau des coefficients K est un corps différentiel commutatif de caractéristique nulle (on prend souvent $K = \mathbb{Q}$). Les « indéterminées différentielles » u_i sont vues comme de simples symboles. Naïvement, une « solution » de Σ, c'est un n–uplet $(\overline{u}_1, \ldots, \overline{u}_n)$ de « valeurs » qui, substituées dans les équations de Σ, donnent zéro. Dans ce chapitre, on se pose le problème de la nature des valeurs $\overline{u}_i$, c'est–à–dire du type de structure algébrique G à laquelle elles peuvent appartenir. La démarche est la suivante : on part des raisonnements tenus et on en déduit des conditions sur G. Beaucoup de manipulations algorithmiques en algèbre différentielle, font usage de deux règles d'inférence et d'un théorème. Les règles d'inférence sont, pour tous polynômes différentiels $p,\, q \in R$:

1. $p = 0 \Rightarrow \theta p = 0$ où θp désigne une dérivée de p d'ordre quelconque,
2. $p\,q = 0 \Rightarrow [p = 0$ ou $q = 0]$.

Avec une autre formulation, la première règle exprime que toute solution de $p = 0$ est solution de $\theta p = 0$; la deuxième que toute solution de $p\,q = 0$ annule l'un ou l'autre des facteurs. Le théorème clef est un Nullstellensatz [94]. Voir [108, chapter VII, paragraph 3, Theorem 14] pour la version non différentielle classique.

Théorème 1 *(Nullstellensatz)*
Tout idéal différentiel[1] *radical de R est une intersection d'idéaux différentiels premiers (cette intersection est finie et unique lorsqu'elle est rendue minimale).*

[1] Un sous–ensemble non vide $\mathfrak{A}$ d'un anneau R est un idéal de R s'il vérifie :

$$[p \in \mathfrak{A} \text{ et } q \in \mathfrak{A}] \Rightarrow p + q \in \mathfrak{A}, \qquad [p \in \mathfrak{A} \text{ et } q \in R] \Rightarrow p\,q \in \mathfrak{A}.$$

Si $\Sigma \subset R$ on note (Σ) le plus petit idéal de R contenant Σ. Un idéal $\mathfrak{A}$ d'un anneau différentiel R est un idéal différentiel de R s'il vérifie

$$p \in \mathfrak{A} \Rightarrow \theta p \in \mathfrak{A}$$

3.2 Les règles d'inférence

Prenons pour premier exemple un système différentiel ordinaire formé de l'unique équation (un célèbre exemple de Joseph Ritt) :

$$u_x^2 - 4\,u = 0.$$

La première règle d'inférence implique qu'on ne change pas les solutions de ce système en lui ajoutant les dérivées de l'équation

$$\left\{\begin{array}{l} u_x^2 - 4\,u = 0, \\ 2\,u_x\,u_{xx} - 4\,u_x = 0, \\ 2\,u_x\,u_{xxx} - 2\,u_{xx}^2 - 4\,u_{xx} = 0, \\ \qquad\vdots \end{array}\right.$$

On observe que la deuxième équation se factorise :

$$2\,u_x\,u_{xx} - 4\,u_x = 2\,u_x\,(u_{xx} - 2).$$

La deuxième règle d'inférence implique que les solutions du système annulent le premier ou le deuxième facteur.

$$u_x\,(u_{xx} - 2) = 0 \quad \Rightarrow \quad u_x = 0 \quad \text{ou} \quad u_{xx} - 2 = 0.$$

Par conséquent, le système est équivalent à une disjonction de deux systèmes :

$$\left\{\begin{array}{l} u_x^2 - 4\,u = 0, \\ u_x = 0 \end{array}\right. \qquad \text{ou} \qquad \left\{\begin{array}{l} u_x^2 - 4\,u = 0, \\ u_{xx} - 2 = 0 \end{array}\right.$$

Si on résout au sens de l'analyse traditionnelle les deux systèmes obtenus, on trouve effectivement deux solutions de l'équation initiale : la fonction nulle $u(x) = 0$ et la famille de paraboles $u(x) = (x + c)^2$ où c est une constante.

Au sens de l'analyse traditionnelle toujours, la première règle d'inférence implique qu'on cherche des solutions qui soient des fonctions indéfiniment dérivables (sur un ouvert qui reste à préciser). C'est une restriction. Considérons la fonction f d'une variable réelle, nulle sur $\mathbb{R}^-$ et égale à x^2 sur $\mathbb{R}^+$. Elle n'est qu'une seule fois dérivable en $x = 0$. Si on cherche des solutions sur un ouvert contenant zéro alors cette solution $u = f$ est « perdue ». Bien sûr, elle ne l'est pas si on cherche des solutions sur un intervalle ouvert ne contenant pas zéro.

---

où θp désigne une dérivée quelconque de p. Si $\Sigma \subset R$ on note $[\Sigma]$ le plus petit idéal différentiel de R contenant Σ. Si $\mathfrak{A}$ est un idéal d'un anneau R on définit le radical de $\mathfrak{A}$ comme l'ensemble de tous les éléments de R dont une puissance appartient à $\mathfrak{A}$

$$\sqrt{\mathfrak{A}} = \{p \in R \mid \exists n \geq 0,\ p^n \in \mathfrak{A}\}.$$

Le radical d'un idéal est un idéal. Le radical d'un idéal différentiel est un idéal différentiel. Un idéal radical est un idéal égal à son radical. Un idéal $\mathfrak{A}$ est premier si $p\,q \in \mathfrak{p} \Rightarrow [p \in \mathfrak{p}$ ou $q \in \mathfrak{p}]$.

Considérons, comme deuxième exemple la fonction f d'une variable réelle, nulle sur $\mathbb{R}^-$, égale à $e^{-\frac{1}{x^2}}$ sur $\mathbb{R}^+$ et la fonction g dont le graphe est symétrique à celui de f par rapport à l'axe des ordonnées. Ces deux fonctions sont indéfiniment dérivables mais non analytiques[2] dans tout ouvert contenant zéro. Le couple $(u,\, v) = (f,\, g)$ est solution de l'équation

$$u\, v = 0$$

de l'anneau de polynômes différentiels $\mathbb{Q}(x)\{u,\, v\}$ muni de la dérivation $\delta_x = \partial/\partial x$. L'application de la deuxième règle d'inférence produit la disjonction

$$u = 0 \quad \text{ou} \quad v = 0.$$

Si on cherche des solutions sur un ouvert contenant zéro alors le couple de fonctions $(u,\, v) = (f,\, g)$ est solution du système initial mais d'aucun des deux systèmes obtenus après scindage : elle est « perdue ». Bien sûr, elle ne l'est pas si on cherche des solutions sur un ouvert ne contenant pas zéro. Synthétisons. Les deux règles d'inférence impliquent que

1. les structures algébriques G dans lesquelles on cherche des solutions de Σ soient des anneaux différentiels (et même des algèbres différentielles sur K) intègres ;
2. quel que soit p appartenant au radical de l'idéal différentiel engendré par Σ, l'équation $p = 0$ est conséquence de Σ.

3.3 Trois versions du théorème des zéros

3.3.1 Solutions abstraites

La première version est une conséquence directe des raisonnements tenus précédemment et du théorème clef.

Théorème et définition 1 *(théorème des zéros pour les solutions abstraites)*
Définissons une solution « abstraite » de Σ comme la donnée

1. *d'une extension de corps différentielle G de K,*
2. *d'un n–uplet $(\overline{u}_1, \ldots, \overline{u}_n) \in G^n$ qui annule les éléments de Σ.*

Alors un polynôme différentiel $p \in R$ s'annule sur toutes les solutions abstraites de Σ si et seulement si $p \in \sqrt{[\Sigma]}$. En particulier Σ est sans solution abstraite si et seulement si $1 \in [\Sigma]$.

[2] On dit qu'une fonction f d'une variable réelle ou complexe x, définie au voisinage de x_0 est *développable en série entière* en x_0 s'il existe une série formelle $S(X) = \sum a_n\, X^n$ dont le rayon de convergence soit non nul et qui satisfasse à

$$f(x) = \sum a_n\, (x - x_0)^n \qquad \text{pour } |x - x_0| \text{ assez petit.}$$

On dit qu'une fonction f d'une variable réelle ou complexe x, définie dans un ouvert $\mathscr{D}$ est *analytique* dans $\mathscr{D}$ si elle est développable en série entière pour tout $x_0 \in \mathscr{D}$.

Preuve L'implication de droite à gauche est immédiate (cf. les raisonnements tenus plus haut).

L'autre implication. On considère un polynôme différentiel $p \notin \sqrt{[\Sigma]}$ et on montre que Σ admet une solution abstraite qui n'annule pas p. Le théorème 1 implique qu'il existe un idéal différentiel premier $\mathfrak{p}$ qui contient Σ mais pas p. L'anneau $R/\mathfrak{p}$ est un anneau intègre puisque $\mathfrak{p}$ est premier et différentiel[3] puisque $\mathfrak{p}$ est différentiel. Il suffit de prendre pour G le corps différentiel des fractions de $R/\mathfrak{p}$ et pour $\overline{u}_1, \ldots, \overline{u}_n$ les images des indéterminées différentielles $u_1, \ldots, u_n$ par le morphisme naturel $\phi : R \to G$. Évaluer un polynôme différentiel en $(\overline{u}_1, \ldots, \overline{u}_n)$ c'est prendre son image par ϕ. Tous les éléments de Σ s'évaluent donc en zéro. Le polynôme p s'évalue en une quantité différente de zéro. □

Les solutions abstraites sont peut–être complètement satisfaisantes d'un point de vue algébrique mais, en pratique, on aimerait quand même pouvoir interpréter les dérivations abstraites comme des dérivations par rapport à des variables indépendantes ($\delta_i = \partial/\partial x_i$) et on aimerait que les solutions soient des n–uplets de « fonctions » de m variables $u_j(x_1, \ldots, x_m)$.

3.3.2 Solutions en séries formelles

Nous allons commencer par faire la moitié du chemin et chercher des solutions dont les composantes $\overline{u}_j$ soient des séries formelles. On interprète donc les dérivations abstraites comme des dérivations par rapport à des variables indépendantes ($\delta_i = \partial/\partial x_i$) et on cherche des solutions de la forme

$$\overline{u}_j = \sum c_{j,a_1,\ldots,a_m} \frac{x_1^{a_1} \cdots x_m^{a_m}}{a_1! \cdots a_m!}.$$

[3]Par analogie avec un cas simple. Par définition, $\mathbb{Z}/n\mathbb{Z}$ est l'ensemble des n classes d'équivalence pour la relation d'équivalence « modulo n » c'est–à–dire modulo l'idéal $(n) = n\mathbb{Z}$ de l'anneau $\mathbb{Z}$. On munit $\mathbb{Z}/n\mathbb{Z}$ d'une structure d'anneau en posant (la « barre » signifie « classe d'équivalence ») :

$$\overline{a} + \overline{b} = \overline{a+b}, \text{ et } \overline{a} \times \overline{b} = \overline{a \times b}.$$

La définition a un sens parce que la classe de $a+b$ (resp. $a \times b$) ne dépend que des classes de a et de b et non des représentants choisis. Par exemple, pour $n = 5$ on vérifie que

$$\begin{array}{ccc} 1 & = & 6 \\ + & & + \\ 3 & = & 13 \\ \| & & \| \\ 4 & = & 19 \end{array} \qquad \begin{array}{ccc} 1 & = & 6 \\ \times & & \times \\ 3 & = & -2 \\ \| & & \| \\ 3 & = & -12 \end{array}$$

Si $\mathfrak{A}$ est un idéal d'un anneau R quelconque, on construit l'anneau quotient $R/\mathfrak{A}$ exactement de la même façon. Si maintenant $\mathfrak{A}$ est un idéal différentiel d'un anneau différentiel R on peut munir l'anneau $R/\mathfrak{A}$ d'une structure d'anneau différentiel en posant, pour toute dérivation δ

$$\delta\overline{a} = \overline{\delta a}.$$

Là aussi, parce que l'idéal est différentiel, la classe de la dérivée de a ne dépend que de la classe de a et pas du représentant choisi.

Les coefficients $c_{j,a_1,\ldots,a_m}$ appartiennent à une structure qui reste à préciser. On verra qu'il suffit de les prendre dans une extension algébrique de K qui dépend du système considéré. Remarque : la série ci–dessus est centrée sur l'origine pour faire simple mais les raisonnements tenus se généralisent immédiatement à des séries centrées en un élément quelconque de $\mathbb{R}^m$. Autre remarque : le cadre ci–dessus couvre aussi le cas de systèmes différentiels dont les coefficients appartiennent à $\mathbb{Q}(x_1, \ldots, x_m)$. Il suffit en effet de coder chaque variable indépendante x_i par une indéterminée différentielle z_i et d'ajouter au système considéré les équations $\delta_j\, z_i = 1$ si $i = j$ et 0 sinon. Nous supposons donc sans perte de généralité que K est un corps de constantes.

Commençons par un exemple : l'idéal différentiel $\mathfrak{A} = [u_x^2 - 4\,u]$ de l'anneau de polynômes différentiels $R = \mathbb{Q}\{u\}$ muni d'une unique dérivation δ_x. On « regarde » $\mathfrak{A}$ comme un idéal non différentiel de l'anneau $\mathbb{Q}[u,\, u_x,\, u_{xx},\, \ldots]$ et on considère une de ses solutions $\hat{u}$ (dans une extension de corps G_0 du corps de base $\mathbb{Q}$ des équations). Concrètement, sur l'exemple, $\hat{u}$ est une solution du système formé de p et de l'infinité de ses dérivées

$$\left\{\begin{array}{l} u_x^2 - 4\,u = 0, \\ 2\,u_x\,u_{xx} - 4\,u_x = 0, \\ 2\,u_x\,u_{xxx} - 2\,u_{xx}^2 - 4\,u_{xx} = 0, \\ \qquad\vdots \end{array}\right.$$

Par exemple

$$\hat{u} = (\hat{u},\, \hat{u}_x,\, \hat{u}_{xx},\, \ldots) = (9,\, 6,\, 2,\, 0,\, \ldots).$$

J'appelle une telle solution une « solution purement algébrique » de $\mathfrak{A}$. Plus formellement,

Définition 1 *(solution purement algébrique)*
Soient $\mathfrak{A}$ un idéal différentiel d'un anneau de polynômes différentiels $K\{U\}$ et ϕ une application de l'ensemble des dérivées ΘU dans une extension de corps non différentielle G_0 de K. L'application ϕ, qui se prolonge de façon unique en un morphisme d'anneau non différentiel $K[\Theta U] \to G_0$, est une « solution purement algébrique » de $\mathfrak{A}$ si $\phi\mathfrak{A} = (0)$.

Formons la série

$$\overline{u} \;=\; \hat{u} + \hat{u}_x\,x + \frac{\hat{u}_{xx}}{2}\,x^2 + \cdots$$

et substituons la dans $u_x^2 - 4\,u = 0$ vue comme une équation différentielle, la dérivation δ_x étant $\partial/\partial x$.

$$\begin{array}{rcl} \overline{u}_x & = & \hat{u}_x + \hat{u}_{xx}\,x + \dfrac{\hat{u}_{xxx}}{2}\,x^2 + \cdots \\ \overline{u}_x^2 & = & \hat{u}_x^2 + 2\,\hat{u}_x\,\hat{u}_{xx}\,x + \cdots \\ \overline{u}_x^2 - 4\,\overline{u} & = & (\hat{u}_x^2 - 4\,\hat{u}) + (2\,\hat{u}_x\,\hat{u}_{xx} - 4\,\hat{u}_x)\,x + \cdots \\ & = & 0. \end{array}$$

On constate que les coefficients de la série $\overline{u}_x^2 - 4\,\overline{u}$ sont donnés par p et ses dérivées évalués sur $\hat{u}$ c'est–à–dire $p(\hat{u})$, $p_x(\hat{u})$, ... Tous les coefficients de la série sont donc nuls. La série $\overline{u}$ est donc une solution de l'équation différentielle. On peut d'ailleurs montrer sur cet exemple précis que les valeurs $\hat{u}_{x^k}$ sont nulles pour $k \geq 3$. La série $\overline{u}$ est donc le polynôme $\overline{u} = 9 + 6\,x + x^2$. On a trouvé l'une des paraboles $u(x) = (x+3)^2$.

Inversement, supposons que la série formelle

$$\overline{u} = \sum \hat{u}_{x^k} \frac{x^k}{k!}$$

soit une solution de l'idéal différentiel $\mathfrak{A}$. Alors, quel que soit $p \in \mathfrak{A}$, la série

$$p(\overline{u}) = \sum p_{x^k}(\hat{u}) \frac{x^k}{k!}$$

est identiquement nulle, donc $p_{x^k}(\hat{u}) = 0$ pour tout k et donc $\hat{u}$ est une solution purement algébrique de $\mathfrak{A}$.

Le raisonnement tenu sur l'exemple se généralise. Soit $\mathfrak{A}$ un idéal différentiel de $R = K\{u_1, \ldots, u_n\}$ muni de dérivations $\partial/\partial x_i$ pour $1 \leq i \leq m$. Un n–uplet $(\overline{u}_1, \ldots, \overline{u}_n)$ de séries formelles

$$\overline{u}_j = \sum c_{j,a_1,\ldots,a_m} \frac{x_1^{a_1} \cdots x_m^{a_m}}{a_1! \cdots a_m!} \qquad (1 \leq j \leq n)$$

(dont les coefficients $c_{j,a_1,\ldots,a_m}$ appartiennent à une extension de corps (non différentielle) G_0 du corps de base K) constitue une solution de $\mathfrak{A}$ si et seulement si l'application

$$\frac{\partial^{a_1+\cdots+a_m} u_j}{\partial x_1^{a_1} \cdots \partial x_m^{a_m}} \mapsto c_{j,a_1,\ldots,a_m}$$

constitue une solution purement algébrique de $\mathfrak{A}$. On en déduit le

Théorème et définition 2 *(théorème des zéros pour les solutions en série formelle)*
Définissons une solution en série formelle de Σ comme la donnée

1. *d'une extension de corps non différentielle G_0 de K (on peut prendre $G_0 = \mathbb{C}$ dans tous les cas),*
2. *d'un n–uplet $(\overline{u}_1, \ldots, \overline{u}_n) \in G = G_0[[x_1, \ldots, x_m]]^n$ qui annule les éléments de Σ.*

Alors un polynôme différentiel $p \in R$ s'annule sur toutes les solutions en série formelle de Σ si et seulement si $p \in \sqrt{[\Sigma]}$. En particulier Σ est sans solution en série formelle si et seulement si $1 \in [\Sigma]$.

Preuve L'implication de droite à gauche est immédiate.

L'implication de gauche à droite. On considère un polynôme différentiel $p \notin \sqrt{[\Sigma]}$ et on montre que Σ admet une solution en série formelle qui n'annule pas p. Le théorème 1 implique qu'il existe un idéal différentiel premier $\mathfrak{p}$ qui contient Σ mais pas p. D'après le théorème des zéros (la version purement algébrique) $\mathfrak{p}$ admet une solution purement algébrique dans une

extension de corps non différentielle G_0 de K (et donc une solution en série formelle dans $G_0[[x_1, \ldots, x_m]]$ d'après ce qui précède) qui n'annule pas p. □

On montre en section 6.1.4, page 97 comment calculer le développement en série formelle d'un idéal différentiel présenté par un ensemble caractéristique différentiel (ou une chaîne différentielle régulière).

3.3.3 Solutions analytiques

Il ne reste plus qu'à parcourir la deuxième moitié du chemin commencé en section 3.3.2 : démontrer que si $\mathfrak{p}$ est un idéal différentiel premier et $p \notin \mathfrak{p}$ alors $\mathfrak{p}$ admet une solution en série formelle, qui converge dans un ouvert $\mathscr{D}$ et qui n'annule pas p.

Proposition 7 *Il suffit de montrer que tout idéal différentiel premier admet une solution analytique.*

Preuve On veut montrer que si $\mathfrak{p}$ est un idéal différentiel premier et $p \notin \mathfrak{p}$ alors $\mathfrak{p}$ admet une solution analytique qui n'annule pas p. Considérons l'idéal différentiel $[\mathfrak{p}, p\,u_{n+1} - 1]$ où u_{n+1} est une nouvelle indéterminée différentielle. Cet idéal admet des solutions abstraites (ou en série formelle) d'après les hypothèses et l'un des théorèmes des zéros que nous avons déjà démontrés. Il est donc différent de l'idéal unité et est contenu dans un idéal différentiel premier $\mathfrak{p}'$ de $R\{u_{n+1}\}$. D'une part, aucune solution de $\mathfrak{p}'$ n'annule p d'autre part, il y a bijection entre les solutions de $\mathfrak{p}'$ et les solutions de $\mathfrak{p}$ qui n'annulent pas p. Il suffit donc de démontrer que $\mathfrak{p}'$ admet au moins une solution analytique et donc que tout idéal différentiel premier admet une solution analytique. □

La démonstration en est faite dans la proposition 9. Le résultat est connu depuis les travaux de Charles Riquier [85]. Le théorème de Riquier, qui est une généralisation du théorème de Cauchy–Kovalevska, est à la base de théorèmes de Joseph Ritt [86] et du [96, 97, Embedding Theorem] d'Abraham Seidenberg. Ariane Péladan–Germa a bien clarifié dans sa thèse [80] le lien existant entre les ensembles caractéristiques et les hypothèses du théorème de Riquier. Récemment, François Lemaire a complètement redémontré [62, Théorème d'analyticité, page 50] ce dernier avec un formalisme plus moderne et en séparant distinctement la preuve d'existence de solutions en séries formelles de la preuve d'analyticité. C'est sur ce dernier travail que je m'appuie ici.

Proposition 8 *(théorème d'analyticité)*
Soit C un ensemble caractéristique différentiel (ou encore une chaîne différentielle régulière) représentant un idéal différentiel premier $\mathfrak{p}$ pour un classement à la fois de Riquier et compatible avec l'ordre total[4]*. Soient L l'ensemble des dérivées dominantes de C et N l'ensemble de toutes les dérivées qui ne sont les dérivées d'aucun élément de L. Soit $(\overline{u}_1, \ldots, \overline{u}_n)$ une solution en série formelle de $\mathfrak{p}$, les coefficients $c_{j,a_1,\ldots,a_m}$ des séries étant pris dans un*

[4] cf. chapitre 2.

sous–corps G_0 du corps des nombres complexes. Soit $(\tilde{u}_1, \ldots, \tilde{u}_n)$ la « restriction à N » de la solution :

$$\tilde{u}_j = \sum \tilde{c}_{j,a_1,\ldots,a_m} \frac{x_1^{a_1} \cdots x_m^{a_m}}{a_1! \cdots a_m!}$$

définie par $\tilde{c}_{j,a_1,\ldots,a_m} = c_{j,a_1,\ldots,a_m}$ si

$$\frac{\partial^{a_1+\cdots+a_m} u_j}{\partial x_1^{a_1} \cdots \partial x_m^{a_m}} \in N$$

et zéro sinon. Les séries $\tilde{u}_j$ sont toutes analytiques au voisinage de l'origine si et seulement si les séries $\overline{u}_j$ le sont aussi.

Proposition 9 *Tout idéal différentiel premier admet une solution analytique.*

Preuve On s'appuie dans cette preuve sur la notion d'ensemble caractéristique différentiel (ou encore de chaîne différentielle régulière) qui est développée dans le chapitre 5.

Tout idéal différentiel premier peut être présenté par un ensemble caractéristique différentiel C pour un classement de Riquier compatible avec l'ordre total. Soient N l'ensemble des dérivées sous les escaliers et X l'ensemble des dérivées dominantes de C. Le système C, vu comme un système non différentiel de $\mathbb{Q}[X \cup N]$ admet une infinité de solutions ayant un nombre fini de coordonnées non nulles. D'après la proposition 32, page 100 et pour peu qu'elles n'annulent pas certains polynômes en nombre fini qui dépendent de C, ces solutions définissent des séries formelles solutions de l'idéal différentiel premier. Les restrictions à N de ces séries sont analytiques (ce sont des polynômes). D'après la proposition précédente, les séries elles–mêmes sont analytiques. □

Illustrons la proposition 8 sur un exemple (linéaire) célèbre[5] : l'équation de la chaleur

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}.$$

Cette équation forme un ensemble caractéristique de l'idéal différentiel qu'elle engendre, quel que soit le classement choisi. Sur cet exemple, la seule chose qui dépende du classement c'est la dérivée dominante de l'équation. Si le classement est compatible avec l'ordre total, la dérivée dominante est $\partial^2 u/\partial x^2$ sinon c'est $\partial u/\partial t$. Pour montrer l'importance des hypothèses, supposons que la dérivée dominante soit $\partial u/\partial t$. La solution purement algébrique

$$\frac{\partial^{k+\ell} u}{\partial x^k \partial t^\ell} \mapsto (k+2\ell)!$$

de l'équation de la chaleur fournit un solution en série formelle

$$u(x,t) = \sum \frac{(k+2\ell)!}{k!\,\ell!}\, x^k\, t^\ell.$$

[5]Historiquement, c'est l'exemple dont s'est servie Sophie Kovalevska pour montrer l'importance d'exprimer les variables les plus dérivées en fonction des autres, c'est–à–dire l'importance des classements compatibles avec l'ordre total.

Pour $x > 0$ et $t > 0$ la série ne converge pas puisqu'elle croît plus vite que la série divergente bien connue :

$$\sum \ell!\, t^\ell.$$

Pourtant, la restriction à

$$N = \left\{ \frac{\partial^k u}{\partial x^k} \mid k \geq 0 \right\}$$

de la série formelle est bien une série convergente :

$$u(x,0) = \sum x^k = \frac{1}{1-x}\cdot$$

La proposition (le théorème de Cauchy–Kovalevska suffit) nous indique que cette situation ne se produit pas si la dérivée dominante est $\partial^2 u/\partial x^2$. L'exemple suivant, dû à François Lemaire [60], montre l'importance des classements de Riquier.

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial x\,\partial t} + \frac{\partial^2 u}{\partial t^2} + v, \quad \frac{\partial^2 v}{\partial t^2} = \frac{\partial^2 v}{\partial x\,\partial t} + \frac{\partial^2 v}{\partial x^2} + u.$$

On choisit les deux membres gauches des équations comme dérivées dominantes du système. Il faut pour cela que le classement choisi ne soit pas de Riquier. Il peut par contre être compatible avec l'ordre total. L'ensemble N est formé des dérivées de u dérivées au plus une fois par rapport à x et des dérivées de v dérivées au plus une fois par rapport à t. On peut former une solution en série formelle $(\overline{u}, \overline{v})$ dont la restriction à N est définie par

$$\overline{u}(0,t) = \frac{\partial u}{\partial x}\,(0,t) = e^t, \quad \overline{v}(x,0) = \frac{\partial v}{\partial t}\,(x,0) = e^x.$$

On peut montrer que les séries $\overline{u}$ et $\overline{v}$ ne sont pas analytiques au voisinage de l'origine. Pourtant la restriction à N de la solution l'est.

Théorème et définition 3 *(théorème des zéros pour les fonctions analytiques)*
Définissons une solution analytique de Σ comme la donnée d'un n–uplet de fonctions $(\overline{u}_1, \ldots, \overline{u}_n)$ de m variables réelles ou complexes, analytiques dans un ouvert $\mathscr{D}$.
Alors un polynôme différentiel $p \in R$ s'annule sur toutes les solutions analytiques de Σ si et seulement si $p \in \sqrt{[\Sigma]}$. En particulier Σ est sans solution analytique si et seulement si $1 \in [\Sigma]$.

Pour des fonctions de m variables complexes, les propriétés suivantes sont équivalentes : être une fois dérivable (c'est–à–dire holomorphe), être indéfiniment dérivable et être analytique. Ce n'est pas le cas pour les fonctions de m variables réelles (cf. les exemples donnés en début de chapitre).

Il est remarquable que le théorème des zéros est valable aussi pour les fonctions indéfiniment dérivables de m variables réelles. En effet, si $p \in \sqrt{[\Sigma]}$ alors toute solution indéfiniment dérivable de Σ est solution de p (c'est immédiat si $p \in [\Sigma]$ et, si une puissance d'une fonction

indéfiniment dérivable p est nulle sur un ouvert alors p est nulle sur cet ouvert). Inversement, si $p \notin \sqrt{[\Sigma]}$ alors Σ admet une solution analytique (et donc indéfiniment dérivable) qui n'annule pas p.

C'est le « mécanisme de scindages » qui est faux pour les fonctions indéfiniment dérivables de m variables réelles : l'ensemble des solutions indéfiniment dérivables de l'équation $u\,v = 0$ contient des solutions qui ne sont ni solution de $u = 0$ ni solution de $v = 0$.

Enfin, l'usage du théorème des zéros nous force à chercher des fonctions analytiques à image dans $\mathbb{C}$ mais pas nécessairement des fonctions analytiques de m variables complexes. Par exemple, l'équation « différentielle »

$$u^2 + 1 = 0$$

de l'anneau de polynômes différentiels $R = \mathbb{Q}\{u\}$, muni de la dérivation δ_x admet pour solutions les fonctions $u(x) = \pm i$. Ces fonctions constantes sont à image dans $\mathbb{C}$ mais la variable x peut être aussi bien une variable réelle que complexe.

On n'a rien précisé au sujet du domaine $\mathscr{D}$ pour éviter certains problèmes d'indécidabilité. La section 6.1.5 page 100 fournit quelques précisions.

Deuxième partie

La théorie des chaînes différentielles régulières

Chapitre 4

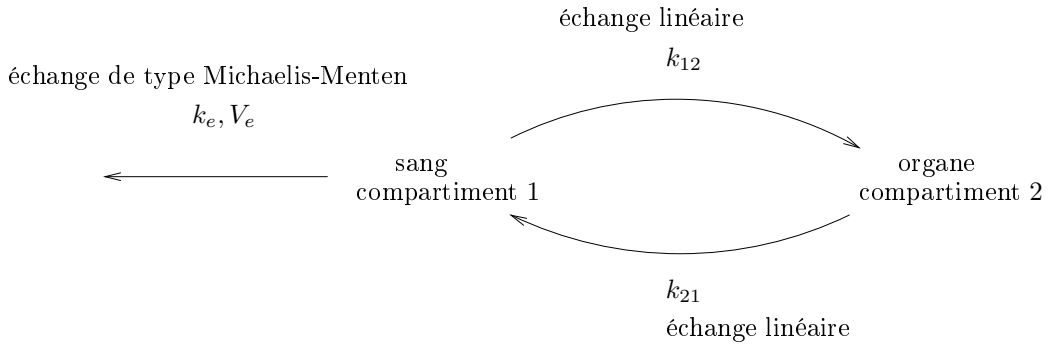
Élimination différentielle et estimation de paramètres

Comme dans la partie précédente, je commence par montrer comment on peut utiliser certains outils algébriques et algorithmiques. Ces outils sont étudiés de façon approfondie dans les chapitres suivants.

On décrit dans ce chapitre une application de l'élimination différentielle et en particulier des algorithmes de changement de classement dans les chaînes différentielles régulières. Cette application est mise en œuvre dans le projet LÉPISME dont le principe, mis au point par Lilianne Denis–Vidal, Ghislaine Joly–Blanchard et Céline Noiret [30], est exposé dans la thèse [75]. Cette application consiste à tenter d'estimer les valeurs de paramètres de systèmes dynamiques dont toutes les variables ne sont pas « observées ». Lorsque toutes les variables sont observées, la méthode fonctionne toujours mais n'a plus besoin d'élimination différentielle. L'étude de l'identifiabilité locale ou globale d'un système par des méthodes d'élimination différentielle est un problème ancien [105, 42, 78, 32, 34, 33, 64] et toujours actuel [3, 93]. L'originalité de la thèse de Céline Noiret réside d'une part dans l'utilisation d'algorithmes d'élimination différentielle rigoureux, d'autre part dans l'étude complète d'exemples, c'est–à–dire jusqu'à l'estimation finale des paramètres. Elle mélange calcul symbolique et calcul numérique. En pratique, elle suppose qu'on dispose d'une modélisation précise des phénomènes qu'on observe et d'un nombre suffisant de mesures suffisamment précises. Dans le cadre du projet LÉPISME, nous avons tenté de l'appliquer à des systèmes issus de la biologie. Bien que la méthode ait été appliquée avec un certain succès en pharmacocinétique [27, 102], je pense que la biologie n'est probablement pas le domaine le plus adapté à la méthode : les modèles sont souvent approximatifs et les mesures fortement bruitées. Je vais tout de même présenter l'application à partir d'un exemple issu de la biologie mais en présentant le problème comme un « challenge » académique plutôt que comme une vraie application.

Le diagramme suivant représente ce qu'on appelle un « modèle à compartiments » tel qu'on les présente dans [20]. Les deux compartiments représentent le sang et un organe. Un médicament est injecté dans le sang. Il peut passer du sang dans l'organe, revenir de l'organe dans le sang. Il peut aussi sortir du système, sous l'action des reins par exemple. On précise le modèle en énonçant des hypothèses sur la nature des échanges. On suppose ici que les

échanges entre les deux compartiments sont linéaires, c'est–à–dire que, sur tout intervalle de temps suffisamment petit, la concentration de médicament passant du compartiment i au compartiment j est proportionnelle à la concentration de médicament dans le compartiment i. La constante de proportionnalité est un paramètre du modèle. Elle est notée k_{ij}. On suppose aussi que l'échange entre le compartiment 1 et les reins suit une loi de type « Michaelis–Menten ». Cette loi est plus difficile à expliquer. Elle se dérive normalement à partir de la « loi d'action de masse » en chimie sous certaines hypothèses. Elle fait apparaître un terme non linéaire dans les équations qui dépend de deux paramètres : une vitesse maximale V_e et une autre constante k_e.



À partir du diagramme ci–dessus, il est possible de dériver automatiquement un système différentiel non linéaire. On associe à chaque compartiment 1 et 2 une fonction du temps $x_1(t)$ et $x_2(t)$ représentant la concentration de médicament qui s'y trouve à tout instant t. Pour construire les équations différentielles, il suffit de considérer les échanges les uns après les autres. Chaque échange ajoute un terme au membre droit de l'équation différentielle qui régit le compartiment de départ (avec un signe moins) et le même terme (avec un signe plus) au membre droit de l'équation différentielle qui régit le compartiment d'arrivée, ce qui implique qu'il y a conservation du médicament. Attention au piège : ce sont les *quantités* de médicament qui sont conservées alors que échanges sont calculés à partir des *concentrations*. Pour simplifier, on suppose ci–dessous que les deux compartiments ont un volume unitaire et on confond les deux notions. On obtient par ce procédé un système de deux équations différentielles non linéaires dépendant de quatre paramètres : k_{12}, k_{21}, k_e et V_e. La seconde est soit linéaire soit polynomiale (ça dépend de la façon dont on voit les paramètres). La première est une fraction rationnelle mais elle est équivalente à une équation polynomiale : on peut la multiplier par son dénominateur dont on est sûr qu'il n'est jamais nul : les paramètres sont tous des réels positifs et les concentrations aussi.

$$\begin{aligned} \dot{x}_1 &= -k_{12}\,x_1 + k_{21}\,x_2 - \frac{V_e\,x_1}{k_e + x_1}, \\ \dot{x}_2 &= k_{12}\,x_1 - k_{21}\,x_2. \end{aligned}$$

On en a fini avec le modèle générique. On s'intéresse maintenant à une instance (à un cas particulier) de ce modèle. Dans ce cas particulier, on dispose de quelques informations sur les paramètres : on suppose que k_{12} et k_{21} sont totalement inconnus, qu'on connaît un intervalle de valeurs $70 \leq V_e \leq 110$ pour V_e (il est assez réaliste de supposer connues des fourchettes

de valeurs pour les paramètres) et que $k_e = 7$ est connu (c'est nettement moins réaliste quoiqu'il arrive qu'on puisse normaliser les équations en divisant chaque paramètre par l'un d'eux). Surtout, on dispose de quelques informations sur les compartiments : on suppose que le compartiment 1 est « observé », c'est–à–dire qu'on dispose d'un fichier de mesures pour $x_1(t)$ mais que le compartiment 2 ne l'est pas. On sait juste qu'initialement $x_2(0) = 0$ c'est–à–dire que l'organe ne contient pas de médicament initialement. Pour fixer les idées, voici un extrait du fichier de mesures dont on dispose pour $x_1(t)$.

```
#    t          x1(t)
0.00000e-01 5.00000e+01
5.00000e-02 4.45078e+01
1.00000e-01 3.93623e+01
...
1.40000e+00 6.63179e-02
1.45000e+00 5.72966e-02
1.50000e+00 4.95270e-02
```

Le « challenge académique » posé est le suivant : le fichier de mesures ci–dessus a été engendré à partir du système différentiel, par intégration numérique pour certaines valeurs des trois paramètres k_{12}, k_{21} et V_e. Il s'agit de retrouver les valeurs de ces paramètres.

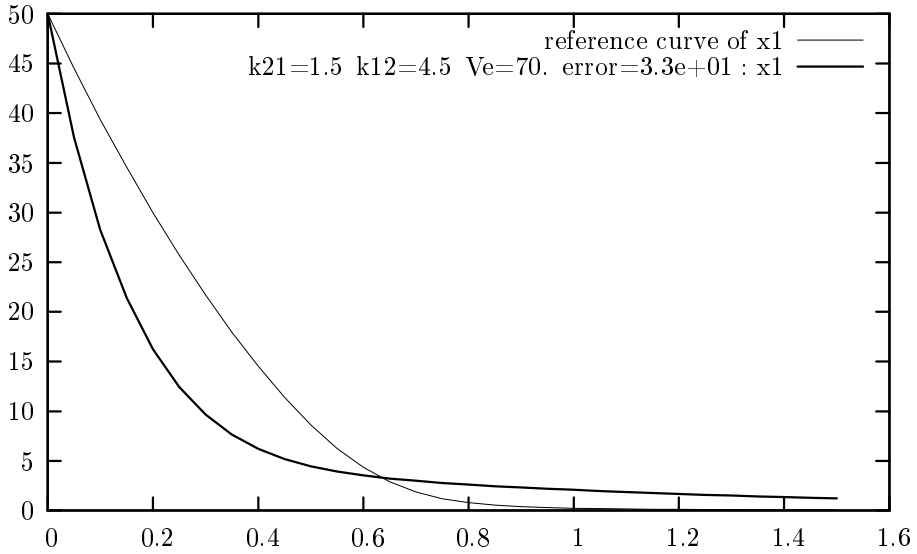
4.1 Une solution purement numérique

Il existe une solution purement numérique pour résoudre ce problème, basée sur une technique de moindres carrés *non linéaires* c'est–à–dire sur une méthode de Newton (plus précisément, une méthode de Gauss–Newton et même mieux une méthode de Levenberg–Marquardt). L'idée est très simple : on tire « au hasard » des valeurs initiales pour les trois paramètres et on intègre numériquement le système différentiel avec ces valeurs. On obtient une « courbe simulée ». On la compare à la « courbe observée » donnée par le fichier de mesures. On définit l'erreur entre les deux courbes comme la somme, pour toutes les abscisses, des carrés des écarts entre les ordonnées des deux courbes. Si les deux courbes ne sont pas assez proches, on utilise la méthode de Levenberg–Marquardt pour améliorer les valeurs des paramètres. On recommence jusqu'à–ce que l'erreur soit suffisamment petite ou que la méthode ait atteint un point stationnaire.

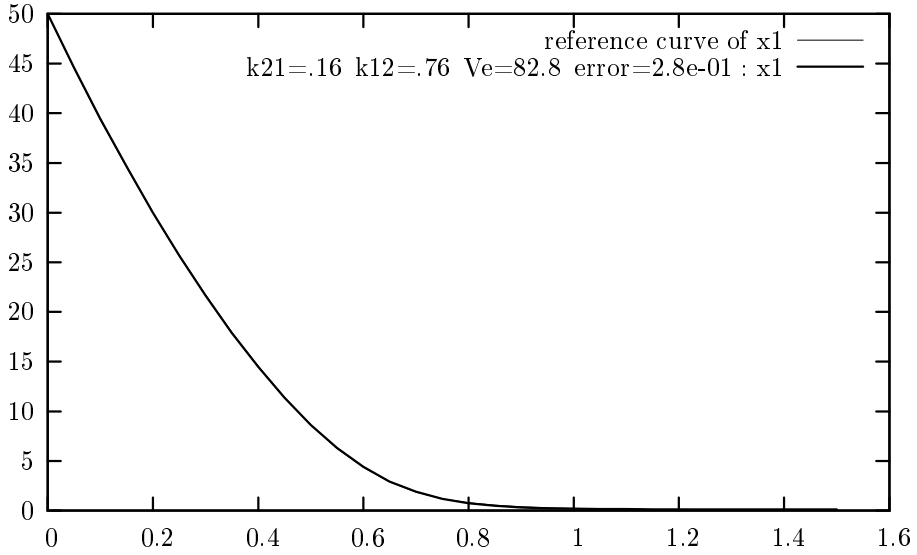
Essayons. On prend pour les trois paramètres les valeurs initiales suivantes :

$$V_e = 70, \quad k_{12} = 4.5, \quad k_{21} = 1.5.$$

On obtient les deux courbes suivantes. Les calculs numériques (intégration numérique des équations différentielles, méthode de Levenberg–Marquardt) ont été effectués par la *Gnu Scientific Library* (*GSL*). Le graphique est obtenu grâce au logiciel *gnuplot*.



Après quelques itérations de la méthode de Levenberg–Marquardt, on finit par obtenir les valeurs des paramètres et le graphique suivants.



La méthode purement numérique semble fonctionner parfaitement mais les valeurs obtenues sont fausses. La méthode de Levenberg–Marquardt s'est arrêtée dans un minimum local. Avant de présenter la méthode mixte symbolique–numérique, dite de l'idéal entrée–sortie, voici deux commentaires.

4.1.1 Identifiabilité

Le modèle est ce qu'on appelle « globalement identifiable ». C'est–à–dire qu'en théorie, à toute courbe $x_1(t)$ correspond un unique jeu de valeurs pour les trois paramètres. La difficulté vient en pratique du fait que la fonction $x_1(t)$ n'est pas parfaitement connue : on ne dispose

que d'une approximation discrète de son graphe.

4.1.2 Matrice de sensibilité de Fisher

La méthode de Levenberg–Marquardt est une méthode de Newton. Elle a besoin d'une « dérivée ». La dérivée qui nous intéresse ici, c'est celle qui donne la sensibilité du graphe de $x_1(t)$ par rapport à une petite variation de l'un des trois paramètres. Pour tout $1 \leq i \leq 2$ et pour tout paramètre p, on définit

$$s_{i,p}(t) \underset{\text{def}}{=} \frac{\partial x_i(t)}{\partial p}.$$

Il y a donc $2 \times 3 = 6$ fonctions $s_{i,p}(t)$. On ne connaît pas ces fonctions mais on peut calculer des équations différentielles qui les définissent. Voici le calcul pour $s_{2,k_{12}}$.

$$\frac{\mathrm{d}}{\mathrm{d}t}\, s_{2,k_{12}} = \frac{\mathrm{d}}{\mathrm{d}t}\, \frac{\partial\, x_2}{\partial\, k_{12}} = \frac{\partial}{\partial\, k_{12}}\, \frac{\mathrm{d}x_2}{\mathrm{d}t} = \frac{\partial}{\partial\, k_{12}}\, (k_{12}\, x_1 - k_{21}\, x_2) = x_1 + k_{12}\, s_{1,k_{12}} - k_{21}\, s_{2,k_{12}}.$$

On obtient ainsi une matrice 2×3 d'équations différentielles ordinaires, dite matrice de sensibilité de Fisher, qui doivent être intégrées numériquement en même temps que les deux équations différentielles initiales. On voit que même dans le cadre d'une méthode purement numérique, il vaut mieux engendrer automatiquement ces équations avec des bibliothèques symboliques telles que *BLAD* que les calculer à la main.

Dans le calcul ci–dessus, on a fait commuter la dérivation totale par rapport au temps et la dérivée partielle par rapport à p. Comment cela se justifie–t–il ? On regarde les variables du modèle comme des fonctions du temps et des paramètres. Soit $x(p,t)$ une telle fonction. On s'intéresse à l'effet d'une petite perturbation du paramètre p sur la fonction. Il suffit pour cela de s'intéresser au développement limité suivant, qu'on a tronqué à l'ordre 1 :

$$x(p+\varepsilon,t) = x(p,t) + \varepsilon\, \frac{\partial}{\partial\, p} x(p,t).$$

On s'intéresse maintenant à l'effet d'une petite perturbation du paramètre p sur la dérivée par rapport au temps de la fonction. Il y a deux façons de le faire. La première consiste à dériver le développement limité ci–dessus. Le point clef, c'est qu'on suppose que la perturbation ε ne dépend pas du temps. On obtient :

$$\frac{\mathrm{d}}{\mathrm{d}t}\, x(p+\varepsilon,t) = \frac{\mathrm{d}}{\mathrm{d}t}\, x(p,t) + \varepsilon\, \frac{\mathrm{d}}{\mathrm{d}t}\, \frac{\partial}{\partial\, p} x(p,t).$$

La seconde façon de faire consiste à calculer le développement limité de la dérivée de x par rapport au temps. On obtient :

$$\frac{\mathrm{d}}{\mathrm{d}t}\, x(p+\varepsilon,t) = \frac{\mathrm{d}}{\mathrm{d}t}\, x(p,t) + \varepsilon\, \frac{\partial}{\partial\, p}\, \frac{\mathrm{d}}{\mathrm{d}t} x(p,t).$$

En comparant les deux formules, on déduit que la dérivation totale par rapport au temps et la dérivation partielle par rapport à x commutent.

4.2 Méthode symbolique–numérique

La méthode précédente a un défaut : elle s'appuie sur des moindres carrés non linéaires qui supposent qu'on connaisse *a priori* une bonne approximation des valeurs des paramètres. En utilisant conjointement de l'élimination différentielle ainsi que des moindres carrés *linéaires* (qui ne supposent la connaissance *a priori* d'aucun jeu de valeurs !) on peut proposer une première approximation des paramètres qui peut servir de jeu de valeurs initial pour la méthode purement numérique décrite ci–dessus.

On cherche à éliminer les variables non observées du modèle. En d'autres termes, on cherche, par élimination différentielle, à calculer une équation différentielle qui soit conséquence des équations du modèle mais qui ne dépende que de la variable observée x_1, de ses dérivées d'ordre quelconque et des paramètres. Pour cela, on pourrait utiliser l'algorithme *Rosenfeld–Gröbner* mais on préfère utiliser l'algorithme *PARDI* (il s'agit en fait ici de sa variante *PODI* pour les systèmes différentiels ordinaires), plus spécialisé et donc plus efficace, de changement de classement dans les chaînes régulières différentielles. La théorie de cet algorithme est développée dans les chapitres qui suivent. Dans ce chapitre, il suffit de savoir que *PODI* prend en entrée une chaîne différentielle régulière C pour un classement r ainsi qu'un autre classement $\bar{r}$. Il retourne une chaîne différentielle régulière $\bar{C}$ pour le classement $\bar{r}$, équivalente à C. Il y a une restriction technique : l'idéal différentiel premier défini par C doit être premier. On admet que tout système de la forme

$$\dot{x}_i = f_i(x_j)$$

où les fonctions f_i sont des fractions rationnelles forme une chaîne différentielle régulière « évidente » vis–à–vis de tout classement compatible avec l'ordre total sur les variables x_i (voir section 6.1.1, page 86). On admet aussi que l'idéal différentiel défini par un tel système est premier. Le programme C suivant utilise *PARDI* pour éliminer x_2 des équations du modèle.

```
/* File change_ordering.c */

#include "bad.h"

int main ()
{   struct bad_regchain C, Cbar;
    bav_Iordering r, rbar;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);
/*
   The natural ordering
*/
    ba0_sscanf2 ("ordering (derivations = [t], \
                  blocks = [[x1, x2], [k12, k21, ke, Ve]])", "%ordering", &r);
```

```
    bav_R_push_ordering (r);
/*
   The model equations
*/
    bad_init_regchain (&C);
    ba0_sscanf2
        ("regchain ([x1[t] + k12*x1 - k21*x2 + Ve*x1/(ke + x1), \
                     x2[t] - k12*x1 + k21*x2, \
                     k12[t], k21[t], ke[t], Ve[t]], \
                    [prime, differential, autoreduced, squarefree, primitive])",
         "%regchain", &C);
/*
   The target ordering
*/
    ba0_sscanf2 ("ordering (derivations = [t], \
         blocks = [x2, [x1, ke], [k12, k21, Ve]])", "%ordering", &rbar);
    bad_init_regchain (&Cbar);
    ba0_sscanf2 ("regchain ([], [autoreduced, squarefree, primitive])",
                 "%regchain", &Cbar);
/*
   Change of ordering over C. Result in Cbar.
*/
    bad_pardi (&Cbar, rbar, &C);

    ba0_printf ("%regchain\n", &Cbar);
/*
   Exit
*/
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Le programme commence par créer un classement vis–à–vis duquel les équations du modèle forment une chaîne différentielle régulière évidente. On remarque que les paramètres du modèle sont vus comme des indéterminées différentielles, c'est–à–dire comme des fonctions du temps. Plutôt que de fixer un pur classement compatible avec l'ordre total, on préfère un classement d'élimination par blocs :

$$(variables) \gg (paramètres).$$

Le programme affecte ensuite à la chaîne régulière C les équations du modèle. L'idéal défini par la chaîne est différentiel et premier. On ajoute aux équations du modèle quatre équations supplémentaires signifiant que les paramètres sont des constantes. Le classement cible est

ensuite défini. Il s'agit là aussi d'un classement d'élimination par blocs :

$$(\textit{vars non observées}) \gg (\textit{vars observées} + \textit{param connu}) \gg (\textit{params inconnus}).$$

Une chaîne différentielle régulière vide $\bar{C}$ est créée et initialisée avec quelques « propriétés désirées » puis *PARDI* est appelé. Voici le résultat obtenu à l'exécution.

```
$ ./change_ordering
regchain ([Ve[t], k21[t], k12[t], ke[t], x1[t,t]*x1^2 + 2*x1[t,t]*x1*ke + x1[t,
t]*ke^2 + x1[t]*x1^2*k12 + x1[t]*x1^2*k21 + 2*x1[t]*x1*ke*k12 + 2*x1[t]*x1*ke*
k21 + x1[t]*ke^2*k12 + x1[t]*ke^2*k21 + x1[t]*ke*Ve + x1^2*k21*Ve + x1*ke*k21*
Ve, x2*x1*k21 + x2*ke*k21 - x1[t]*x1 - x1[t]*ke - x1^2*k12 - x1*ke*k12 - x1*Ve]
, [differential, prime, autoreduced, primitive, squarefree, coherent])
```

L'équation qui nous intéresse est la cinquième. On l'appelle « équation entrée–sortie ». En regroupant entre crochets les « blocs de paramètres » à estimer on peut l'écrire :

$$\ddot{x}_1\,(x_1+k_e)^2 + [k_{12}+k_{21}]\,\dot{x}_1\,(x_1+k_e)^2 + [V_e]\,\dot{x}_1\,k_e + [k_{21}\,V_e]\,x_1\,(x_1+k_e) = 0.$$

Cette équation nous apprend que le modèle est globalement identifiable. En effet, supposons les *fonctions* x_1, $\dot{x}_1$ et $\ddot{x}_1$ « connues ». On peut alors les évaluer pour trois valeurs de t et remplacer le paramètre connu k_e par sa valeur. On obtient ainsi un système d'équations linéaires exactement déterminé dont les inconnues sont les blocs de paramètres. Ce système admet une unique solution. Les valeurs des blocs étant fixées, il est évident (sur cet exemple !) que les valeurs des paramètres le sont aussi. CQFD.

En pratique, la fonction x_1 est connue d'après le fichier de mesures et on peut essayer d'estimer numériquement les valeurs de la dérivée première $\dot{x}_1$ et de la dérivée seconde. En l'absence de bruits, la dérivée première est assez bien estimée mais ce n'est pas forcément le cas de la dérivée seconde. Pour tenter de pallier les erreurs dues à ces approximations numériques, on construit donc un système linéaire surdéterminé qu'on résout par une méthode de moindres carrés *linéaires*. Sur l'exemple, voici la solution obtenue :

$$[k_{12}+k_{21}] = 2.1, \quad [V_e] = 87.29, \quad [k_{21}\,V_e] = 144.01.$$

On dispose des valeurs des blocs de paramètres. Il reste à trouver les valeurs des paramètres en résolvant le système ci–dessus, vu comme un système algébrique dont les indéterminées sont les paramètres du modèle. C'est très facile sur l'exemple et on obtient :

$$V_e = 87.29, \quad k_{12} = 0.45, \quad k_{21} = 1.65.$$

Le jeu de valeurs ci–dessus peut maintenant être utilisé comme jeu de valeurs initiales pour la méthode purement numérique décrite précédemment. Toujours sur l'exemple, on trouve les valeurs correctes des paramètres :

$$V_e = 101, \quad k_{12} = 0.5, \quad k_{21} = 3.$$

4.2.1 Les difficultés

Dans le cas général, rien ne permet de garantir que les valeurs proposées par la méthode symbolique–numérique conduisent bien la méthode purement numérique dans le minimum global.

Le problème de l'estimation de paramètres n'a de sens que pour des modèles identifiables au moins localement mais tester cette propriété ne représente pas une vraie difficulté. On dispose pour cela d'algorithmes semi–numériques efficaces tels que ceux mis au point par Alexandre Sedoglavic dans [93]. Il s'agit d'algorithmes probabilistes pour lesquels on dispose d'une estimation de la probabilité d'erreur, qu'on peut diminuer à volonté.

Une importante difficulté réside dans l'estimation numérique des dérivées, surtout en présence de données bruitées. Nous avons procédé en calculant des polynômes d'interpolation passant par les points du fichier de mesures et en estimant les dérivées sur les polynômes interpolants. On peut essayer d'atténuer l'effet du bruit en remplaçant autant que possible les équations différentielles par des équations intégrales comme le suggère Lilianne Denis–Vidal dans [29] mais ce n'est pas toujours possible.

Il reste une dernière importante difficulté, qui n'est résolue que partiellement dans le logiciel actuel : il peut exister des relations algébriques entre les blocs de paramètres. Sur l'exemple, il n'y en a pas. Mais supposons que la relation entrée–sortie ait comporté les blocs suivants, de telle sorte que le troisième bloc soit le produit des deux premiers.

$$[V_e], \quad [k_{21}], \quad [V_e\, k_{21}].$$

Lors de la résolution par moindres carrés linéaires du système surdéterminé, il est certain que les valeurs numériques attribuées aux blocs de paramètres ne satisferaient pas cette relation. Par conséquent, lors de la résolution algébrique finale, où on tente de déduire les valeurs des paramètres à partir des valeurs des blocs, on trouverait en toute rigueur un système sans solution.

La thèse [75] fournit une solution à ce problème : il suffit de résoudre le système algébrique final par une méthode de moindres carrés non linéaires : on minimise la somme des carrés des équations. C'est un peu dommage parce qu'on retrouve le problème de choisir un jeu de valeurs initiales pour les paramètres.

Je pense qu'il serait très intéressant de mettre au point une méthode symbolique, basée sur de l'élimination algébrique. En effet, non seulement on obtiendrait la solution cherchée mais on pourrait décider si le système a un nombre fini de solutions et, dans l'affirmative, obtenir *toutes* ces solutions. On résoudrait ainsi non seulement le problème de l'estimation des paramètres mais aussi en quelque sorte le problème de l'identifiabilité du modèle. Quand le système algébrique a un nombre infini de solutions, c'est que le modèle n'est pas identifiable et que l'estimation de paramètres n'a aucun sens. Quand le système algébrique a un nombre fini de solutions, c'est que le système est identifiable.

Est–ce déraisonnable de penser utiliser une méthode fondée sur l'élimination algébrique ? Je ne pense pas, pour peu que le système dynamique comporte de nombreuses variables observées (au moins la moitié). Dans ce cas, l'élimination différentielle est vite faite, elle

ne fait pas grossir les relations entrées–sorties, les blocs de paramètres sont petits et les éliminations algébriques devraient être peu coûteuses.

Toutes ces difficultés ont une contrepartie que je trouve très positive : ils sont nombreux, ceux qui prétendent que les méthodes d'élimination différentielle sont inutilisables en pratique en raison de leur complexité. Mais on voit bien que les difficultés énoncées ci–dessus sont principalement dues à la partie numérique de la chaîne de traitement et pas à l'élimination différentielle.

4.2.2 L'implantation

Un premier jet de la méthode décrite ci–dessus a été complètement programmé dans le cadre du projet *LÉPISME*. La *Gnu Scientific Library* a été utilisée pour interpoler les fichiers de mesures disponibles pour les variables observées (des splines d'ordre 3 ont été utilisées), évaluer numériquement les dérivées et appliquer la méthode des moindres carrés linéaires.

François Lemaire a réalisé un prototype d'interface en langage JAVA à partir d'un premier jet dû à Thibaut Henin. Natacha Skrzypczak a étudié [98] de façon approfondie le langage SBML qui constitue un standard de description de modèles biologiques. C'est une variante de XML. L'utilisation de ce standard devrait nous permettre d'améliorer l'interopérabilité de nos logiciels avec ceux des autres et en particulier des éditeurs de modèles tels que *CellDesigner*.

La méthode est assez compliquée : elle enchaîne plusieurs étapes : élimination différentielle, simplification heuristique des relations entrées–sorties, estimation des blocs de paramètres par moindres carrés linéaires, résolution de systèmes d'équations algébriques, moindres carrés non linéaires. Pour chacune de ces étapes, plusieurs méthodes sont souvent disponibles (à commencer par le choix entre la méthode purement numérique ou la méthode mixte symbolique–numérique).

Idéalement, le logiciel devrait pouvoir être assemblé à la demande par son utilisateur en sélectionnant les méthodes qu'il souhaite utiliser. Idéalement, le logiciel devrait décrire synthétiquement à l'utilisateur la nature des problèmes rencontrés en cas d'échec pour l'aider dans son choix. C'est bien sûr beaucoup plus facile à dire qu'à faire et l'implantation actuelle du logiciel n'offre pas cette souplesse.

Les bibliothèques *BLAD* comportent toutefois un mécanisme qui rend possible ce type d'implantation : au début de toute suite d'appels aux bibliothèques *BLAD*, il est possible de spécifier des limites en temps de calcul et en espace mémoire à ne pas dépasser avant la fin de la suite d'appels. Dans le cas où ces bornes sont atteintes, une exception est levée qui permet au programme appelant non seulement de reprendre la main mais aussi de récupérer un environnement de travail propre : toutes les ressources consommées par les bibliothèques *BLAD* lors de la suite d'appels sont libérées. Cette fonctionnalité est indispensable : en effet, il est impossible de prévoir à l'avance le temps de calcul ou l'espace mémoire consommé par les algorithmes connus d'élimination algébrique et différentielle (à la différence de l'élimination de Gauss par exemple, pour laquelle on dispose de bornes de complexité qui décrivent fidèlement le comportement de l'algorithme) et personne ne voudrait utiliser une fonction logicielle au comportement aussi imprévisible. Enfonçons le clou : on rencontre régulièrement

des systèmes impossibles à résoudre en pratique alors que des variantes syntaxiquement très proches se résolvent instantanément.

Le programme C ci–dessous montre sur un exemple minimal comment les bibliothèques *BLAD* (il suffit de *ba0* en l'occurence) permettent de mettre en œuvre ce mécanisme.

```
/* File out_of_time.c */

#include "ba0.h"

int main ()
{   struct ba0_mark M;
    struct ba0_exception_code code;
    void* pointer;

    ba0_restart (5, 128);
    ba0_record (&M);

    BA0_PUSH_EXCEPTION (code);
    if (ba0_exception_is_raised (code))
    {   if (ba0_mesgerr == BA0_ERRALR)
            ba0_fprintf (stderr, "error: the program ran out of time\n");
        else if (ba0_mesgerr == BA0_ERROOM)
            ba0_fprintf (stderr, "error: the program ran out of memory\n");
        pointer = ba0_alloc (10);
        printf ("%x\n", pointer);
        ba0_restore (&M);
        ba0_terminate (ba0_init_level);
        return 1;
    }

    pointer = ba0_alloc (10);
    printf ("%x\n", pointer);
    do { } while (true);

    ba0_restore (&M);
    ba0_terminate (ba0_init_level);
    return 0;
}
```

Les deux paramètres fournis à *ba0_ restart* spécifient les bornes à ne pas dépasser : 5 secondes et 128 Mo. Un point de traitement d'exception est posé (une sorte d'équivalent de *try* en C++). Ensuite vient la suite d'instructions à exécuter dans le cas où une exception tente de remonter (une sorte d'équivalent de *catch*). Le traitement d'exception consiste ici simplement à tester la nature de l'exception, à imprimer un message sur l'erreur standard et à arrêter le processus avec un code de retour indiquant une fin anormale. Le programme principal alloue une zone de 10 octets à la variable *pointer* puis entre dans une boucle infinie.

Au bout d'à peu près 5 secondes l'exception *BA0_ERRALR* est automatiquement levée et les instructions de traitement d'exceptions sont exécutées. Les 10 octets alloués à *pointer* sont automatiquement restitués au système de gestion de la mémoire. Cela se voit au fait que les deux valeurs affichées sont identiques. Voici l'affichage obtenu à l'exécution :

```
$./out_of_time
f630f008
error: the program ran out of time
f630f008
```

On trouve dans le code des bibliothèques *BLAD* plusieurs endroits où des points de traitement d'exceptions sont posés et où des exceptions sont levées. Parmi ces exceptions, certaines sont très faciles à traiter à l'intérieur de la bibliothèque (un nombre premier « malchanceux » ou un diviseur de zéro est découvert, un algorithme heuristique échoue ...) et d'autres impossibles à traiter (typiquement, des dépassements des bornes en temps et en espace attribuées aux calculs). À l'extérieur de la bibliothèque, les rôles sont inversés : les exceptions faciles à traiter ne le sont plus du tout alors que les dépassements des bornes en temps et en espace se gèrent très naturellement.

Chapitre 5

Algèbre différentielle et simplification canonique

Ce chapitre aurait pu s'intituler « l'algèbre commutative et différentielle racontée par l'algorithme d'Euclide ». Les idées qui y sont exposées sont le résultat d'au moins dix ans de travail (1990 – 2000) dans plusieurs équipes. Les articles de Mickael Kalkbrener [54] et de Daniel Lazard [58] sont les articles « fondateurs » dans le domaine purement algébrique. La clarification et la correction des idées initiales a ensuite été faite par des étudiants de Daniel Lazard : Renaud Rioboo, Marc Moreno Maza et Philippe Aubry [71, 69, 1, 70]. J'ai adapté leurs résultats au cas différentiel en collaboration avec François Lemaire [62] et Marc Moreno Maza. La construction des formes normales est due à François Lemaire et à moi [12]. La présentation que je fais ici du cas différentiel est originale.

Soit $\mathfrak{A}$ un idéal d'un anneau de polynômes R. On cherche un simplificateur canonique pour la relation d'équivalence modulo $\mathfrak{A}$. L'idéal $\mathfrak{A}$ est présenté, en un sens qui reste à préciser, par un système d'équations polynomiales A. Les équations de A sont interprétées comme un système de réécriture. Pour fixer les idées, supposons que l'équation en quatre indéterminées $x\,y - z\,t = 0$ fasse partie de A. Il y a deux grandes façons de l'interpréter comme une règle de réécriture. Soit on la regarde comme une règle permettant de réécrire l'un des monômes en l'opposé de l'autre soit on la regarde comme une règle permettant de réécrire l'une des variables en une fraction rationnelle :

$$x\,y \longrightarrow z\,t \qquad \text{ou} \qquad x \longrightarrow \frac{z\,t}{y}.$$

Dans le premier cas, l'idéal $\mathfrak{A}$ est l'idéal engendré par A, la forme normale d'un polynôme de R est un polynôme de R et on aboutit à la théorie des bases de Gröbner. Dans le second cas, l'idéal $\mathfrak{A}$ est l'idéal engendré par A mais saturé par les dénominateurs des membres droits des règles de réécriture, la forme normale d'un polynôme de R est une fraction rationnelle et on aboutit à la théorie des systèmes triangulaires. C'est à cette dernière approche que je m'intéresse dans ce chapitre parce que c'est la seule qui se soit généralisée de façon satisfaisante aux équations différentielles. Les lecteurs intéressés trouveront en annexe le support

d'un cours qui présente la théorie des bases de Gröbner à des étudiants de premier cycle universitaire.

On a utilisé l'expression « forme normale » ci–dessus sans la définir précisément. Ce chapitre se propose justement d'en construire la définition petit à petit. Pour le moment, on se contentera de la définition suivante.

Définition 2 *(définition imprécise des formes normales)*
On appelle forme normale *d'un polynôme $f \in R$ par un système de réécriture A toute fraction rationnelle p/q appartenant au corps des fractions de R, équivalente à f modulo l'idéal défini par le système de réécriture A, dont le numérateur et le dénominateur sont irréductibles par A.*

On rappelle qu'une congruence $f = p/q \mod \mathfrak{A}$ n'a de sens que si q est régulier modulo $\mathfrak{A}$. Voici les propriétés désirées :

1. la forme normale d'un polynôme f doit être unique et ne dépendre que de la classe d'équivalence de f modulo $\mathfrak{A}$;
2. la forme normale d'un polynôme f doit être calculable.

Les deux problèmes clefs sont, étant donnés un élément $f \in R$ et l'idéal $\mathfrak{A}$

1. décider si $f = 0$ dans $R/\mathfrak{A}$,
2. décider si f est diviseur de zéro dans $R/\mathfrak{A}$.

Rappelons qu'un élément a d'un anneau R divise zéro si $a \neq 0$ et s'il existe $b \in R$, $b \neq 0$ tel que $a\,b = 0$. Les qualificatifs « non diviseur de zéro » et « régulier » sont synonymes. Un élément $a \in R$ est inversible s'il admet un inverse dans R, c'est–à–dire s'il existe $b \in R$ tel que $a\,b = 1$. Tout élément inversible est régulier (puisque si $a\,b = 1$ et $a\,c = 0$ alors $0 = (a\,c)\,b = (a\,b)\,c = c$). La réciproque est fausse.

5.1 Un polynôme en une indéterminée

Il s'agit du cas d'un idéal $\mathfrak{A} = (p)$ engendré par un seul élément dans un anneau de polynômes en une indéterminée et à coefficients dans un corps K.

5.1.1 Définition de la forme normale

Définition 3 *On appelle* forme normale *d'un polynôme f par p tout polynôme g équivalent à f modulo $\mathfrak{A}$ et tel que* $\deg(g) < \deg(p)$.

Une forme normale d'un polynôme f modulo $\mathfrak{A}$ est le reste de la division euclidienne de f par p. Elle existe donc et il est bien connu qu'elle est unique.

$$\mathrm{NF}(f,\, A) = \mathrm{rem}(f,\, p).$$

5.1.2 Décider de la nullité et de l'inversibilité

Proposition 10 *Décider si $f = 0$ dans $R/\mathfrak{A}$ revient à décider si le reste de la division euclidienne de f par p est nul.*

Proposition 11 *Décider si f est régulier dans $R/\mathfrak{A}$ revient à décider si le pgcd de f et de p vaut* 1.

Preuve Un élément de R appartient à $\mathfrak{A}$ si et seulement s'il est un multiple de p. Dire que f divise zéro dans $R/\mathfrak{A}$, c'est dire qu'il existe un élément h tel que $f\,h$ est multiple de p mais que ni f ni h ne le sont. C'est dire en d'autres termes que f et p ont un diviseur commun g non trivial (tel que $0 < \deg g < \deg p$). □

5.1.3 Algorithmique

L'algorithme d'Euclide suffit pour calculer le pgcd de deux polynômes. Voici l'algorithme d'Euclide étendu.

```
fonction Euclide_étendu (a, b)
début
  U := (1, 0, a)
  V := (0, 1, b)
  tant que v3 ≠ 0 faire
    q := le quotient de la division euclidienne de u3 par v3
    T := V
    V := U − q V
    U := T
  fait
  c := le coefficient de x^(deg u3) dans u3
  retourner (1/c) U
fin
```

Le calcul se fait en parallèle sur les trois composantes des vecteurs U et V. Voici quelques « invariants de boucle » (propriétés vraies à chaque fois qu'on évalue le test $v_3 \neq 0$) :

1. $u_1\,a + u_2\,b = u_3$ et $v_1\,a + v_2\,b = v_3$,
2. l'ensemble des diviseurs communs de a et de b est égal à l'ensemble des diviseurs communs de u_3 et de v_3.

Par conséquent, lorsque la boucle s'arrête, le plus grand diviseur commun de a et de b est égal au plus grand diviseur commun de u_3 et de zéro, c'est–à–dire à u_3 et on obtient une identité de Bézout :

$$u_1\,a + u_2\,b = u_3 = \mathrm{pgcd}(a, b).$$

On en déduit la fonction suivante.

```
fonction inverse_algébrique (a, p)
début
  (u1, u2, u3) := Euclide_étendu (a, p)
  si u3 ≠ 1 alors
    le calcul échoue car a divise zéro modulo 𝔄
  sinon
    u1
  fin si
fin
```

Prenons par exemple $p = x^2 - 2$ et $f = x$. La fonction *Euclide_ étendu*, appliquée à f et à p retourne l'identité de Bézout

$$u_1\, f + u_2\, p = u_3 \quad \text{avec} \quad (u_1,\, u_2,\, u_3) = \left(\frac{x}{2},\, \frac{-1}{2},\, 1\right).$$

On en déduit que modulo l'idéal $\mathfrak{A} = (p)$ on a

$$\frac{1}{x} = \frac{x}{2}.$$

On retrouve la relation bien connue :

$$\frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}.$$

5.2 Systèmes triangulaires unitaires comportant autant d'équations que d'inconnues

Il s'agit du cas où $R = K[x_1, \ldots, x_n]$ et $A = \{f_1, \ldots, f_n\}$ est un système triangulaire unitaire tel que $\mathrm{ld}\, p_i = x_i$. Par « unitaire » on entend que les éléments de A ont tous 1 pour coefficient initial. Ici, l'idéal $\mathfrak{A} = (A)$. Un système triangulaire et unitaire est une base de Gröbner de l'idéal qu'il engendre. La notion de forme normale que nous allons établir coïncide avec celle des bases de Gröbner.

5.2.1 Définition de la forme normale

Définition 4 *On appelle* forme normale *d'un polynôme f par un système A tout polynôme p équivalent à f modulo $\mathfrak{A}$ et tel que* $\deg(p,\, x_i) < \deg(f_i,\, x_i)$ *pour tous* $1 \leq i \leq n$.

Le lemme clef est le suivant.

Lemme 2 *Si p est un polynôme de l'idéal $\mathfrak{A}$ tel que* $\deg(p,\, x_i) < \deg(f_i,\, x_i)$ *pour tous* $1 \leq i \leq n$ *alors* $p = 0$.

Preuve Notons $d_k = \deg(f_k, x_k)$. On suppose $p \in \mathfrak{A}$ et $\deg(p,\, x_k) < d_k$ pour $1 \leq k \leq n$ et on cherche à conclure que $p = 0$. Comme $p \in \mathfrak{A}$ on a une formule $p = \lambda_1\, f_1 + \cdots + \lambda_k\, f_k$. Si $k = 0$ alors $p = 0$ et le lemme est prouvé. Supposons donc qu'on ait affaire à une formule de longueur $k > 0$ avec $\lambda_k \neq 0$, notons $f_k = x_k^{d_k} + g_k$ et substituons $x_k^{d_k} \to f_k - g_k$ dans $\lambda_1,\, \ldots,\, \lambda_{k-1}$. On obtient une nouvelle formule $p = \mu_1\, f_1 + \cdots + \mu_k\, f_k$ qu'on peut réarranger de telle sorte que $\deg(\mu_i,\, x_k) < d_k$ pour $1 \leq i < k$. Comme $\deg(p,\, x_k) < d_k$ et $\deg(f_i,\, x_k) = 0$ pour $1 \leq i < k$ on conclut que $\mu_k = 0$. On a donc obtenu pour p, une nouvelle formule de longueur strictement inférieure à k. En continuant de proche en proche, on déduit que $p = 0$. □

Proposition 12 *La forme normale d'un polynôme f par un système A est unique. Deux polynômes équivalents modulo $\mathfrak{A}$ ont même forme normale.*

Preuve Il suffit de montrer que deux polynômes f et f' équivalents modulo $\mathfrak{A}$ ont même forme normale. Soient g une forme normale de f et g' une forme normale de f'. Le polynôme $g - g'$ est une forme normale et appartient à l'idéal. Il doit donc être nul. □

Proposition 13 *La définition de la forme normale est algorithmique.*

Preuve Le polynôme suivant constitue une forme normale du polynôme f.

$$\mathrm{NF}(f,\, A) = \mathrm{rem}(f,\, A) = \mathrm{rem}(\ldots \mathrm{rem}(\mathrm{rem}(f,\, f_n,\, x_n),\, f_{n-1},\, x_{n-1}) \ldots,\, f_1,\, x_1).$$

□

5.2.2 Décider de la nullité et de l'inversibilité

Proposition 14 *Décider si $f = 0$ dans $R/\mathfrak{A}$ revient à décider si* $\mathrm{rem}(f,\, A) = 0$. *Plus généralement, décider si $f \in K$ dans $R/\mathfrak{A}$ revient à décider si* $\mathrm{rem}(f,\, A) \in K$.

Intéressons–nous maintenant au test d'inversibilité dans $R/\mathfrak{A}$.

On dispose d'une généralisation naturelle de l'algorithme d'Euclide étendu. Soient a et b deux polynômes d'indéterminée principale x_k. Notons $R_{k-1} = K[x_1,\, \ldots,\, x_{k-1}]$. On voit les polynômes a et b comme polynômes en x_k et à coefficients pris modulo $\mathfrak{A}$ c'est–à–dire comme des éléments de $(R_{k-1}/(\mathfrak{A} \cap R_{k-1}))[x_k]$ et on cherche à calculer une identité de Bézout $u_1\, a + u_2\, b = u_3$ dans cet anneau.

On cherche donc à généraliser *Euclide_ étendu* en un algorithme d'entête *Euclide_ étendu* $(a,\, b,\, x_k,\, A)$. Les deux premières instructions d'*Euclide_ étendu* n'ont pas besoin d'être changées. On peut décider de l'égalité à zéro modulo $\mathfrak{A}$ (cf. les propositions ci–dessus). On peut donc déterminer le degré d'un polynôme à coefficients modulo $\mathfrak{A}$ et en particulier réaliser le test $v_3 \neq 0$. Le calcul de l'inverse d'un élément de $R_{k-1}/(\mathfrak{A} \cap R_{k-1})$ reste la dernière opération non évidente à réaliser : on la trouve explicitement à la fin de la boucle et implicitement dans la division euclidienne. On utilise pour cela la fonction suivante.

```
fonction inverse_ algébrique (a, A)
début
  si a ∈ K alors
    si a ≠ 0 alors
      1/a
    sinon
      le calcul d'inverse échoue (division par zéro)
    fin si
  sinon
    soit x_i l'indéterminée principale de a
    (u_1, u_2, u_3) := Euclide_ étendu (a, f_i, x_i, A)
    si u_3 ≠ 1 alors
      le calcul d'inverse échoue (inversion d'un diviseur de zéro)
    sinon
      u_1
    fin si
  fin si
fin
```

Voici l'algorithme *Euclide_ étendu* généralisé.

```
fonction Euclide_ étendu (a, b, x_i, A)
début
  U := (1, 0, a)
  V := (0, 1, b)
  tant que v_3 ≠ 0 faire
    q := le quotient de la division euclidienne de u_3 par v_3
    T := V
    V := U − q V
    U := T
  fait
  c := le coefficient de x_i^(deg u_3) dans u_3
  retourner inverse_ algébrique (c, A) U
fin
```

Terminaison. Les fonctions *inverse_ algébrique* et *Euclide_ étendu* s'appellent récursivement l'une l'autre. Les calculs terminent dans tous les cas parce que l'indéterminée principale x_i décroît strictement à chaque appel à *Euclide_ étendu*.

Correction. Point clef : les invariants de boucle de la version généralisée de *Euclide_ étendu* sont les mêmes que ceux de la version traditionnelle. Précisément,

1. $u_1\,a + u_2\,b = u_3 \mod \mathfrak{A}$ et $v_1\,a + v_2\,b = v_3 \mod \mathfrak{A}$,

2. dans $(R_{k-1}/(\mathfrak{A} \cap R_{k-1}))[x_k]$, l'ensemble des diviseurs communs de a et de b est égal à l'ensemble des diviseurs communs de u_3 et de v_3.

Noter que la preuve du deuxième invariant ne fait intervenir que des opérations d'anneau. Il n'est pas nécessaire de supposer l'anneau factoriel. Ces invariants ont deux corollaires.

1. si la fonction *inverse_ algébrique* termine normalement son exécution, l'élément retourné u_1 est un inverse de a dans $R_{k-1}/(\mathfrak{A} \cap R_{k-1})$ et *a fortiori* dans $R/\mathfrak{A}$,
2. si $u_3 \neq 1$ alors u_3 est un facteur non trivial de f_i dans $(R_{k-1}/(\mathfrak{A} \cap R_{k-1}))[x_k]$.

Il est normal que le calcul puisse échouer puisque $R/\mathfrak{A}$ n'est pas nécessairement intègre. Cela n'implique pas d'ailleurs que a soit un diviseur de zéro puisque la fonction est amenée à tester l'inversibilité d'éléments autres que a. En cas d'échec, une factorisation d'un $f_i \in A$ est exhibée. Cette factorisation $f_i = g\,h$ (les facteurs s'obtiennent par simple division euclidienne) permet de scinder le système A en deux systèmes triangulaires unitaires, obtenus en remplaçant f_i par l'un ou l'autre de ses facteurs. Pour résumer,

Proposition 15 *La fonction* inverse_algébrique*, appliquée à un polynôme f et à A, soit prouve que f est inversible dans $R/\mathfrak{A}$ soit permet de scinder le système A en deux systèmes plus simples.*

Si on ne s'intéresse qu'au test d'inversibilité et non au calcul de l'inverse, on peut utiliser une version non étendue de l'algorithme d'Euclide. La version non étendue est nettement moins coûteuse : le pgcd de deux polynômes est petit (il vaut souvent 1) mais les coefficients des identités de Bézout sont gros. La fonction *inverse_ algébrique* ne fournit qu'un algorithme de semi–décision. On peut la compléter pour obtenir un algorithme de décision mais c'est là à mon avis un problème peu intéressant en pratique. Considérons par exemple le système A formé des deux équations suivantes :

$$x_2^2 + x_1 - 2 = 0, \quad x_1^2 - 1 = 0.$$

La fonction *inverse_ algébrique*, appliquée au polynôme $3\,x_2$, retourne son inverse $(x_2\,x_1 + 2\,x_2)/9$. La fonction *inverse_ algébrique*, appliquée au polynôme $x_1 + x_2$ échoue. Elle exhibe la factorisation $x_1^2 - 1 = (x_1 - 1)(x_1 + 1)$ qui permet de scinder le système A en deux systèmes plus simples :

$$x_2^2 - 1 = 0, \quad x_1 - 1 = 0 \quad \text{ou} \quad x_2^2 - 3 = 0, \quad x_1 + 1 = 0.$$

Pour paraphraser Michel Demazure [26, page 42], les idéaux décrits par des systèmes triangulaires admettant autant d'équations que d'inconnues sont des « idéaux premiers à usage commercial » : ce sont des idéaux qui se comportent comme des idéaux premiers jusqu'à–ce qu'on découvre qu'ils ne le sont pas. Dans le domaine du calcul formel, cette idée est connue sous le nom de « principe D^5 » du nom des auteurs de [35] : Jean Della Dora, Claire Dicrescenzo et Dominique Duval.

Le programme C suivant montre comment ce mécanisme est implanté dans les bibliothèques *BLAD*. La chaîne régulière précédente est lue et affectée à C. Les deux polynômes

sont lus et affectés au tableau de polynômes T. Une boucle calcule leur inverse algébrique et imprime soit l'inverse soit le diviseur de zéro exhibé. En *BLAD*, les tableaux sont des structures à trois champs : un champ *alloc* contenant le nombre d'éléments alloués au tableau, un champ *size* contenant le nombre d'éléments utilisés dans le tableau et un champ *tab* qui pointe sur les éléments. Comme il est normalement rarissime qu'un calcul d'inverse algébrique échoue, la fonction *bad_invert_polynom_mod_regchain* est programmée pour lever une exception lorsque cet événement se produit. Il s'agit d'une variante du mécanisme présenté au chapitre précédent qui permet à la fonction qui lève l'exception de retourner un objet (ici le diviseur de zéro exhibé) à la fonction qui traite l'exception. Le diviseur de zéro est retourné via la variable *ddz*.

```
/* File inverse_algebrique.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bad_regchain C;
    struct bap_product_mpz U, G;
    struct bap_tableof_polynom_mpz T;
    bap_polynom_mpz ddz;
    struct ba0_exception_code code;
    struct ba0_mark M;
    int i;

    bad_restart (0, 0);
    ba0_record (&M);
    ba0_sscanf2 ("ordering (derivations = [], blocks = [x2, x1])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bad_init_regchain (&C);
    ba0_sscanf2 ("regchain ([x2^2 + x1 - 2, x1^2 - 1], [])", "%regchain", &C);
    ba0_init_table ((ba0_table)&T);
    bap_init_product_mpz (&U);
    bap_init_product_mpz (&G);
    ba0_sscanf2 ("[3*x2, x1 + x2]", "%t[%Az]", &T);
    for (i = 0; i < T.size; i++)
    {   BA0_PUSH_EXCEPTION (code);
        if (ba0_exception_is_raised (code))
        {   ba0_printf ("Inversion of %Az failed. ", T.tab [i]);
            ba0_printf ("Zero divisor found : %Az\n", ddz);
        } else
        {   bad_invert_polynom_mod_regchain (&U, &G, T.tab [i], &C, &ddz);
            ba0_pull_exception (code);
            ba0_printf ("Inversion of %Az succeeded\n", T.tab [i]);
```

```
            ba0_printf
                ("%Pz * %Az = %Pz modulo %regchain\n", &U, T.tab [i], &G, &C);
        }
    }
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu à l'exécution.

```
Inversion of 3*x2 succeeded
x2*(x1 + 2) * 3*x2 = 9 modulo regchain ([x1^2 - 1, x2^2 + x1 - 2], [])
Inversion of x2 + x1 failed. Zero divisor found : x1 - 1
```

Les bibliothèques *BLAD* offrent aussi une fonction nommée *bad_check_regularity_polynom_mod_regchain* qui vérifie qu'un polynôme passé en paramètre est inversible (mais sans calculer l'inverse) et qui lève une exception sinon.

5.3 Systèmes triangulaires

Il s'agit du cas où $R = K[x_1, \ldots, x_n, t_1, \ldots, t_m]$ et $A = \{f_1, \ldots, f_n\}$ est un système triangulaire tel que $\mathrm{ld}\, f_i = x_i$. Les x_i sont les indéterminées principales du système, les t_i les indéterminées non principales. Comme on ne suppose pas A unitaire, on ne dispose plus de l'algorithme de division euclidienne. On cherche alors des méthodes de décision fondés sur l'algorithme disponible le plus proche : l'algorithme de pseudo–division, noté prem. Si $f \in R$ est pseudo–réduit à zéro alors $f \in (A) : I_A^\infty$ où I_A désigne l'ensemble des initiaux des éléments de A. L'idéal qu'on est donc naturellement amené à associer au système n'est donc plus l'idéal engendré par A mais[1] $\mathfrak{A} = (A) : I_A^\infty$.

5.3.1 Le point clef et la définition des chaînes régulières

Le point clef ci–dessous est démontré dans le chapitre 7.

Point clef 1 *Décider de la nullité ou de la régularité dans $R/\mathfrak{A}$ est strictement équivalent à décider de la nullité ou de la régularité dans $R_0/\mathfrak{A}_0$ où $R_0 = K(t_1, \ldots, t_m)[x_1, \ldots, x_n]$ est l'anneau de polynômes obtenu en « faisant passer les indéterminées non principales dans le corps des coefficients » et $\mathfrak{A}_0 = (A) : I_A^\infty$ dans R_0.*

[1] « Oui ... mais si on s'intéresse quand même à l'idéal (A) ? »
« Eh bien, soit on l'étudie directement avec les bases de Gröbner soit on le décompose en une intersection d'idéaux de la forme $(A) : I_A^\infty$ avec A triangulaire. Des méthodes existent pour cela dont je ne parlerai pas dans ce chapitre. Pour être tout–à–fait correct, je précise que ces méthodes ne représentent pas exactement l'idéal (A) mais un idéal compris entre (A) et son radical.

Ce théorème s'applique à l'idéal $(A) : I_A^\infty$. Il est faux pour l'idéal (A). Plaçons–nous dans R_0. Nous sommes presque ramenés au problème précédent. « Presque » parce que, si le système A admet autant d'équations que d'inconnues, il n'est pas unitaire. On cherche une condition suffisante pour le rendre unitaire. Considérons f_1. Le coefficient initial c_1 de f_1 appartient à $K(t_1, \ldots, t_m)$. Son inverse $\overline{c}_1$ existe. On ne change pas l'idéal $\mathfrak{A}_0$ en remplaçant f_1 par le polynôme unitaire $f_1' = \overline{c}_1 f_1$. Considérons ensuite f_2. Son coefficient initial c_2 appartient à $K(t_1, \ldots, t_m)[x_1]$. On peut tenter de l'inverser avec l'algorithme *inverse_ algébrique* de la section précédente. C'est possible parce que ce calcul ne fait intervenir que f_1', qui est unitaire, et pas $f_2, \ldots, f_n$ qui ne le sont pas. Supposons que son inverse $\overline{c}_2$ (modulo l'idéal $(f_1') \subset \mathfrak{A}_0$ donc) existe. On ne change pas l'idéal $\mathfrak{A}_0$ en remplaçant f_2 par $f_2' = \overline{c}_2 f_2$. On voit, en continuant de proche en proche, que dans R_0, si pour tout $1 \leq k \leq n$, le coefficient initial de f_k est inversible modulo l'idéal $(f_1, \ldots, f_{k-1})$ alors A est équivalent à un système triangulaire unitaire comportant autant d'équations que d'inconnues. Ce sont de tels systèmes qu'on appelle des chaînes régulières. La définition suivante synthétise tous ces raisonnements.

Définition 5 *Un ensemble triangulaire A est une* chaîne régulière *si pour tout $2 \leq k \leq n$ le coefficient initial de f_k est régulier modulo l'idéal $(f_1, \ldots, f_{k-1}) : (i_1 \cdots i_{k-1})^\infty$.*

Les chaînes régulières sont des ensembles équivalents à des systèmes triangulaires unitaires comportant autant d'équations que d'inconnues pour peu qu'on fasse passer les indéterminées non principales dans le corps des coefficients. Dans cette définition encore, c'est le fait de saturer par les initiaux qui permet de faire passer les indéterminées non principales dans le corps des coefficients. Considérons par exemple le système A suivant de $R = K[t, x_1, x_2]$.

$$(x_1 + 1)\, x_2^2 + x_1 + 1 = 0, \quad t\, x_1^2 - 1 = 0.$$

On a $R_0 = K(t)[x_1, x_2]$. Construisons A_0 de proche en proche. On rend unitaire la première équation sans difficulté :

$$(x_1 + 1)\, x_2^2 + x_1 + 1 = 0, \quad x_1^2 - 1/t = 0.$$

La fonction *inverse_ algébrique* permet de calculer l'inverse $t\,(x_1 - 1)/(1 - t)$ de l'initial $x_1 + 1$ de la deuxième équation, modulo l'idéal engendré par la première. On obtient ainsi le système A_0 suivant.

$$x_2^2 + \frac{t\,(x_1 - 1)}{1 - t}\, x_2 + \frac{t\,(x_1 - 1)}{1 - t} = 0, \quad x_1^2 - 1/t = 0.$$

Le système A constitue donc une chaîne régulière. La fonction suivante soit, prouve qu'un ensemble triangulaire A est une chaîne régulière soit, exhibe une factorisation d'un élément de A qui permet de scinder A en deux systèmes plus simples.

fonction *est_ une_ chaîne_ régulière* (A)
début
  $A_0 := \emptyset$

```
  pour i variant de 1 à n faire
    c := le coefficient dominant de f_i
    c̄ := inverse_algébrique (c, A_0) dans K(t_1, ..., t_m)[x_1, ..., x_{i-1}]
    si le calcul d'inverse a échoué alors
      retourner faux
    fin si
    A_0 := A_0 ∪ {rem(c̄ f_i, A_0)}
  fin
  retourner vrai
fin
```

5.3.2 Décider de la nullité et de la régularité

Le « point clef » étant admis, les deux propositions ci–dessous sont immédiates.

Proposition 16 *Soit A une chaîne régulière. Décider si $f = 0$ dans $R/\mathfrak{A}$ revient à décider si* $\mathrm{prem}(f, A) = 0$.

Proposition 17 *Soit A une chaîne régulière. Décider si f est régulier dans $R/\mathfrak{A}$ revient à décider si f est inversible dans $R_0/\mathfrak{A}_0$.*

5.3.3 Définition de la forme normale

Définition 6 *On appelle* forme normale *d'un polynôme f par une chaîne régulière A toute fraction rationnelle de la forme suivante telle que $f = p/q \mod \mathfrak{A}$ et $\deg(p, x_i) < \deg(p_i, x_i)$ pour tous $1 \leq i \leq n$:*

$$\mathrm{NF}_{alg}(f, A) = \frac{p}{q} \in K(t_1, \ldots, t_m)[x_1, \ldots, x_n]$$

Le fait que $q \in K[t_1, \ldots, t_m]$ assure que q est régulier dans $R/\mathfrak{A}$ et donc que la congruence $f = p/q \mod \mathfrak{A}$ est sensée.

Proposition 18 *La forme normale d'un polynôme f par une chaîne régulière A est unique. Deux polynômes équivalents modulo $\mathfrak{A}$ ont même forme normale.*

Preuve Il suffit de montrer que deux polynômes équivalents modulo $\mathfrak{A}$ ont même forme normale. On suppose $f = g \mod \mathfrak{A}$ et $\mathrm{NF}_{alg}(f, A) = p/q$ et $\mathrm{NF}_{alg}(g, A) = r/s$. Alors $sp - qr \in \mathfrak{A}$. Parce que les dénominateurs q et s ne dépendent que des variables t_i, le polynôme $sp - qr$ est irréductible par A (il est égal à son propre pseudo–reste par A). D'après l'une des propositions ci–dessus, il doit être nul. □

Proposition 19 *La définition des formes normales est algorithmique.*

Preuve Soient f un polynôme et A une chaîne régulière de R. Soit A_0 le système de R_0 triangulaire, unitaire, comportant autant d'équations que d'inconnues, auquel A est équivalente. La fraction rationnelle $\mathrm{rem}(f,\, A_0)$ est une forme normale de f par A. □

Reprenons l'exemple précédent.

$$\mathrm{NF}_{alg}(x_1^2\, x_2^2,\, A) = \mathrm{rem}(x_1^2\, x_2^2,\, A_0) = \frac{(x_1 - 1)\, x_2 + x_1 - 1}{t - 1}.$$

5.4 Systèmes différentiels ordinaires

Il s'agit du cas où l'anneau de polynômes différentiels $R = K\{U\}$ est muni d'une seule dérivation et où le système $A = \{f_1,\, \ldots,\, f_n\}$ est « différentiellement triangulaire », c'est–à–dire où aucune dérivée dominante[2] d'un polynôme de A n'est la dérivée de la dérivée dominante d'un autre polynôme de A. Dans le cas d'une seule dérivation, un système différentiellement triangulaire est un système dont les dérivées dominantes sont des dérivées d'indéterminées différentielles de U distinctes deux–à–deux. On ne suppose pas A partiellement autoréduit (c'est une nouveauté). On suppose l'ensemble ΘU des dérivées muni d'un « classement ». Voici un exemple avec trois indéterminées différentielles : u, v et w.. L'anneau de polynômes différentiels est $R = K\{u,\, v,\, w\}$. L'ensemble ΘU est ordonné suivant un classement d'élimination :

$$w < \dot{w} < \ddot{w} < \cdots < v < \dot{v} < \ddot{v} < \cdots < u < \dot{u} < \ddot{u} < \cdots$$

L'ensemble des dérivées dominantes du système A suivant est $X = \{\dot{u},\, \dot{v}\}$. Le système est bien différentiellement triangulaire.

$$\dot{v}^2 + \dot{v} + w = 0, \quad \ddot{v}\, \dot{u} + u = 0.$$

Tout système différentiel A est équivalent à un système algébrique infini ΘA, formé des polynômes de A et de leurs dérivées d'ordre quelconque. Voici un extrait du système ΘA associé à l'exemple ci–dessus.

$$\left\{\begin{array}{rcl}
\dot{v}^2 + \dot{v} + w &=& 0 \\
(2\,\dot{v} + 1)\, \ddot{v} + \dot{w} &=& 0 \\
(2\,\dot{v} + 1)\, \dddot{v} + 2\,\ddot{v} + \ddot{w} &=& 0 \\
&\vdots& \\
\ddot{v}\, \dot{u} + u &=& 0 \\
\ddot{v}\, \ddot{u} + (\dddot{v} + 1)\, \dot{u} &=& 0 \\
&\vdots&
\end{array}\right.$$

Le système ΘA est triangulaire parce que A est différentiellement triangulaire et qu'on se restreint à une unique dérivation. Les coefficients initiaux de ΘA sont les coefficients initiaux et les séparants de A. Ces polynômes sont en nombre fini. On cherche naturellement à généraliser la construction des chaînes régulières établie en section 5.3 à ces systèmes infinis. L'idéal représenté par un système différentiel A est donc $\mathfrak{A} = (\Theta A) : I_{\Theta A}^{\infty} = [A] : H_A^{\infty}$.

[2] On dit « dérivée dominante » au lieu d' « indéterminée principale » dans le cas différentiel.

5.4.1 La définition des chaînes différentielles régulières

On commence par définir ce qu'on appelle les « dérivées sous l'escalier » d'un système A qui jouent pour les systèmes différentiels le rôle des indéterminées $t_1, \ldots, t_m$ de la section 5.3. L'expression « escalier » prendra tout son sens lorsqu'on traitera les systèmes aux dérivées partielles.

Définition 7 *Soient A un système de $R = K\{U\}$ et X l'ensemble des dérivées dominantes de A. On appelle* dérivées sous l'escalier *de A l'ensemble $T = \Theta U \setminus \Theta X$ des dérivées qui ne sont la dérivée d'aucune dérivée dominante de A.*

Sur l'exemple précédent, l'ensemble des dérivées sous l'escalier de A est l'ensemble infini

$$T = \{u,\, v,\, w,\, \dot{w},\, \ddot{w},\, \ldots\}.$$

La proposition ci–dessous est une généralisation très naturelle du point clef 1. Cette généralisation ne coule pas de source parce que la démonstration du point clef 1 s'appuie sur les théorèmes de Lasker–Nœther et de Macaulay qui ne s'appliquent qu'à des anneaux nœtheriens or les anneaux de polynômes différentiels ne sont pas nœtheriens. Pour un argument précis, voir le théorème 11, page 121.

Proposition 20 *Décider de la nullité ou de la régularité dans $R/\mathfrak{A}$ est strictement équivalent à décider de la nullité ou de la régularité dans $R_0/\mathfrak{A}_0$ où $R_0 = K(T)[\Theta X]$ est l'anneau de polynômes obtenu en « faisant passer les dérivées sous l'escalier de A dans le corps des coefficients » et $\mathfrak{A}_0 = (\Theta A) : I_{\Theta A}^{\infty}$ dans R_0.*

Définition 8 *Un système différentiel ordinaire A est une* chaîne différentielle régulière *si ΘA est une chaîne régulière.*

Cette définition est algorithmique. Ce n'est pas évident parce que ΘA est infini. On montre sur l'exemple comment prouver que A est une chaîne différentielle régulière. On établit l'algorithme ensuite. Il suffit de procéder par dérivée dominante croissante. On considère le polynôme $f_1 = \dot{v}^2 + \dot{v} + w$. Il est unitaire et constitue donc une chaîne régulière de $K[\dot{v},\, w]$. Le séparant de f_1 est le polynôme $2\,\dot{v} + 1$. Un appel à la fonction *inverse_ algébrique* permet de calculer son inverse.

$$\frac{1}{2\,\dot{v}+1} = \frac{2\,\dot{v}+1}{1-4\,w}.$$

Ce calcul est justifié implicitement par la proposition 20. Il ne fait intervenir que la chaîne régulière f_1. On considère le polynôme $f_2 = \ddot{v}\,\dot{u} + u$. Pour calculer l'inverse de son initial, la fonction *inverse_ algébrique* doit utiliser une dérivée de f_1. On voit–là l'importance de vérifier que le séparant de f_1 est inversible avant de considérer f_2. Le calcul d'inverse produit :

$$\frac{1}{\ddot{v}} = -\frac{2\,\dot{v}+1}{\dot{w}}.$$

On a montré la régularité des initiaux de tous les polynômes de ΘA. Le système A est donc bien une chaîne différentielle régulière. La fonction suivante fournit un algorithme qui décide, dans le cas d'une seule dérivation, si un système différentiellement triangulaire A est une chaîne différentielle régulière.

fonction *est_une_chaîne_différentielle_régulière* (A)
début
  $X :=$ l'ensemble des dérivées dominantes de A
  $T :=$ l'ensemble des dérivées sous l'escalier de A
  $A' := \emptyset$
  pour i variant de 1 à n faire
    $c :=$ l'initial de f_i
    $\bar{c} :=$ *inverse_algébrique* $(c, \Theta A')$ dans $K(T)[\Theta X]$
    si le calcul d'inverse a échoué alors
      retourner faux
    fin si
    $s :=$ le séparant de f_i
    $\bar{s} :=$ *inverse_algébrique* $(s, \Theta A' \cup \{f_i\})$ dans $K(T)[\Theta X]$
    si le calcul d'inverse a échoué alors
      retourner faux
    fin si
    $A' := A' \cup \{f_i\}$
  fait
  retourner vrai
fin

Un système différentiel A est une chaîne différentielle régulière si ΘA est équivalent à un système $(\Theta A)_0$ triangulaire unitaire, comportant une équation par inconnue, pour peu qu'on fasse passer les dérivées sous l'escalier dans le corps des coefficients. Voici un extrait du système $(\Theta A)_0$ sur l'exemple.

$$\left\{\begin{array}{rcl} \dot{v}^2+\dot{v}+w & = & 0 \\ \ddot{v}+\frac{\dot{w}\,(2\,\dot{v}+1)}{1-4\,w} & = & 0 \\ \dddot{v}+\frac{(2\,\ddot{v}+\dot{w})\,(2\,\dot{v}+1)}{1-4\,w} & = & 0 \\ & \vdots & \\ \dot{u}-\frac{u\,(2\,\dot{v}+1)}{\dot{w}} & = & 0 \\ \ddot{u}-\frac{(\ddot{v}+1)\,\dot{u}\,(2\,\dot{v}+1)}{\dot{w}} & = & 0 \\ & \vdots & \end{array}\right.$$

5.4.2 Décider de la nullité et de la régularité

La proposition 20 étant admise, les deux propositions ci–dessous sont immédiates.

Proposition 21 *Soit A une chaîne différentielle régulière. Décider si $f = 0$ dans $R/\mathfrak{A}$ revient à décider si* $\mathrm{prem}(f, \Theta A) = 0$*, c'est–à–dire si le reste complet de f par A pour la réduction de Ritt est nul.*

Proposition 22 *Soient A une chaîne différentielle régulière. Décider si f est régulier dans l'anneau $R/\mathfrak{A}$ revient à décider si f est régulier dans $R_0/\mathfrak{A}_0$.*

On peut aussi montrer (formulation plus classique) que décider si f est régulier dans l'anneau $R/\mathfrak{A}$ revient à décider si le reste partiel de f par A pour la réduction de Ritt est régulier modulo l'idéal non différentiel défini par A.

5.4.3 Définition de la forme normale

Définition 9 *Soient A une chaîne différentielle régulière, X l'ensemble de ses dérivées dominantes et T l'ensemble des dérivées sous l'escalier de A. On appelle* forme normale *d'un polynôme différentiel f par A toute fraction rationnelle de la forme suivante, telle que $f = p/q \mod \mathfrak{A}$ et $\deg(p, \mathrm{ld}\, f_i) < \deg(f_i, \mathrm{ld}\, f_i)$ pour tout $f_i \in A$.*

$$\mathrm{NF}(f, A) = \frac{p}{q} \in K(T)[X]$$

La proposition 20 implique que tout polynôme non nul de $K[T]$ est régulier dans $R/\mathfrak{A}$. La congruence énoncée dans la définition ci–dessus est donc sensée. La preuve de la proposition suivante est calquée sur celle de la proposition 18.

Proposition 23 *La forme normale d'un polynôme différentiel f par une chaîne différentielle régulière A est unique. Deux polynômes différentiels équivalents modulo $\mathfrak{A}$ ont même forme normale.*

Proposition 24 *La définition des formes normales est algorithmique.*

Preuve Soient f un polynôme différentiel et A une chaîne différentielle régulière. La fraction rationnelle suivante est une forme normale de f par A.

$$\mathrm{NF}(f, A) = \mathrm{NF}_{alg}(f, \Theta A).$$

□

Le programme C suivant montre comment on peut calculer la forme normale d'une fraction rationnelle de polynômes différentiels en *BLAD*. Le calcul de la forme normale implique un calcul d'inverse algébrique. Ce calcul peut échouer. Une exception est donc susceptible d'être levée. Dans l'exemple, on choisit de ne pas poser de point de traitement d'exceptions. Comme la fraction est peu lisible, on factorise complètement son numérateur et son dénominateur et on affiche le résultat.

```
/* File normal_form.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bap_product_mpz numer, denom;
    struct bap_ratfrac_mpz A, NF;
```

```
    struct bad_regchain C;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);

    ba0_sscanf2 ("ordering (derivations = [t], blocks = [u, v, w])",
                 "%ordering", &r);
    bav_R_push_ordering (r);

    bad_init_regchain (&C);
    bap_init_ratfrac_mpz (&A);
    bap_init_ratfrac_mpz (&NF);

    ba0_sscanf2
        ("regchain ([v[t]^2 + v[t] + w, v[t,t]*u[t] + u], [differential])",
         "%regchain", &C);
    ba0_sscanf2 ("u[t,t]/v[t]", "%Qz", &A);

    bad_normal_form_ratfrac_mod_regchain (&NF, &A, &C, (bap_polynom_mpz*)0);

    ba0_printf ("normalf form (%Qz) = %Qz\n", &A, &NF);

    bap_init_product_mpz (&numer);
    bap_init_product_mpz (&denom);
    bap_factor_polynom_mpz (&numer, &NF.numer);
    bap_factor_polynom_mpz (&denom, &NF.denom);
    ba0_printf ("numer = %Pz\n", &numer);
    ba0_printf ("denom = %Pz\n", &denom);

    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu à l'exécution.

```
$ ./normal_form
normalf form ((u[t,t])/(v[t])) = (8*u*v[t]^2*w[t,t]*w - 2*u*v[t]^2*w[t,t] - 4*u
*v[t]^2*w[t]^2 + 12*u*v[t]*w[t,t]*w - 3*u*v[t]*w[t,t] - 6*u*v[t]*w[t]^2 + 16*u*
v[t]*w^2 - 8*u*v[t]*w + u*v[t] + 4*u*w[t,t]*w - u*w[t,t] - 2*u*w[t]^2 + 16*u*w^
2 - 8*u*w + u)/(4*w[t]^2*w^2 - w[t]^2*w)
numer = u*(8*v[t]*w[t,t]*w - 2*v[t]*w[t,t] - 4*v[t]*w[t]^2 + 4*w[t,t]*w - w[t,t
] - 2*w[t]^2 + 16*w^2 - 8*w + 1)*(v[t] + 1)
denom = w[t]^2*w*(4*w - 1)
```

5.5 Systèmes aux dérivées partielles

On s'intéresse à des systèmes où l'anneau de polynômes différentiels $R = K\{U\}$ est muni de plusieurs dérivations $\delta_1, \ldots, \delta_m$ supposées commuter entre elles. Considérons le système A suivant :

$$u_x - v = 0, \quad u_y = 0.$$

Supposons que le classement impose $u_x > v$. Le système A est différentiellement triangulaire mais ΘA n'est pas triangulaire. Il comporte deux équations de dérivée dominante u_{xy} :

$$u_{xy} - v_y = 0, \quad u_{xy} = 0.$$

Cette non triangularité de ΘA pose des difficultés. L'idéal différentiel $[A]$ engendré par les deux polynômes différentiels contient v_y mais v_y n'est pas réduit à zéro par ΘA. Autre formulation du problème : certains polynômes différentiels peuvent avoir plusieurs formes normales différentes, en supposant qu'on utilise tel quel l'algorithme établi dans le cas différentiel ordinaire. C'est le cas de u_{xy} :

$$\mathrm{NF}(u_{xy},\, A) = v_y \quad \text{ou} \quad \mathrm{NF}(u_{xy},\, A) = 0.$$

Dernière conséquence de la non triangularité de ΘA : l'algorithme de calcul d'inverse algébrique peut donner des résultats faux. Le polynôme v_y est nul modulo l'idéal différentiel engendré par A mais il ne dépend que de dérivées sous l'escalier de A. Il serait considéré comme inversible par la fonction *inverse_algébrique*.

Les difficultés énoncées ci–dessus peuvent se résoudre au prix d'une condition supplémentaire : la « cohérence ». Le théorème clef est le « lemme de Rosenfeld » qui est énoncé précisément dans le chapitre 7. On se contente ici d'énoncer informellement l'idée sous–jacente en s'appuyant sur la construction des formes normales.

Bien qu'il ne soit pas triangulaire, on souhaite pouvoir traiter ΘA comme un système triangulaire, de façon à pouvoir appliquer dans le cas des systèmes aux dérivées partielles les constructions faites dans le cas différentiel ordinaire. Informellement, dire qu'un système est « cohérent » c'est dire que deux équations quelconques de ΘA qui ont la même dérivée dominante v sont en fait « la même équation » modulo l'idéal défini par les équations de ΘA de dérivée dominante strictement inférieure à v. On peut tester si deux équations $s_1\, v + q_1$ et $s_2\, v + q_2$ sont « la même équation », en testant que les deux fractions rationnelles q_1/s_1 et q_2/s_2 ont même forme normale et en utilisant l'algorithme établi pour les systèmes différentiels ordinaires, pour peu qu'on procède à ces vérifications par dérivée croissante. Dernier point, et non des moindres : bien qu'il y ait une infinité de paires d'équations appartenant à ΘA de même dérivée dominante, il n'y a qu'un nombre fini de tests à effectuer.

La fonction suivante précise quelques aspects du texte ci–dessus. Elle s'applique à un système différentiellement triangulaire A qu'on ne suppose pas partiellement autoréduit (c'est une nouveauté). Pour toute dérivée v, on note $(\Theta A)_{<v}$ (respectivement $(\Theta A)_{\leq v}$) l'ensemble des éléments de ΘA de dérivée dominante strictement inférieure (respectivement inférieure ou égale) à v. Pour toute paire $\{v,\, w\}$ de dérivées d'une même indéterminée différentielle,

on note $\mathrm{ppcd}(v, w)$ la plus petite dérivée commune de v et de w. Si X est un ensemble de dérivées, on note

$$\mathrm{ppcd}(X) = \{\mathrm{ppcd}(v,\, w) \mid v,\, w \in X,\ \text{sont dérivées d'une même indéterminée différentielle}\}.$$

La fonction met en œuvre le schéma de preuve utilisé par Abraham Seidenberg dans [95, Theorem VI]. La justification s'appuie sur les théorèmes 10 et 11, page 121. Le fait que les dérivées de Y soient traitées par ordre croissant est important : pour toute dérivée v de Y, les calculs de formes normales sont effectués modulo $(\Theta A)_{<v}$ dont l'algorithme a précédemment établi qu'il est équivalent à une chaîne régulière.

fonction *est_une_chaîne_aux_dérivées_partielles_régulière* (A)
début
  $T :=$ l'ensemble des dérivées sous l'escalier de A
  $X :=$ l'ensemble des dérivées dominantes de A
  $Y := X \cup \mathrm{ppcd}(X)$
  pour $v \in Y$ par ordre croissant suivant le classement faire
    si v est la dérivée dominante d'un $f \in A$ alors
      $c :=$ l'initial de f
      $\bar{c} :=$ *inverse_algébrique* $(c,\, (\Theta A)_{<v})$ dans $K(T)[\Theta X]$
      si le calcul d'inverse a échoué alors
        retourner faux
      fin si
      $s :=$ le séparant de f
      $\bar{s} :=$ *inverse_algébrique* $(s,\, (\Theta A)_{\leq v})$ dans $K(T)[\Theta X]$
      si le calcul d'inverse a échoué alors
        retourner faux
      fin si
    sinon
      Soient $f_1, \ldots, f_k \in A$ et $\theta_1, \ldots, \theta_k \in \Theta$ tels que $\theta_i f_i = s_i\, v + q_i$ pour tous $1 \leq i \leq k$
      pour i de 1 à $k-1$ faire
        si $\mathrm{NF}(q_i/s_i,\, (\Theta A)_{<v}) \neq \mathrm{NF}(q_{i+1}/s_{i+1},\, (\Theta A)_{<v})$ alors
          retourner faux
        fin si
      fait
    fin si
  fait
  retourner vrai
fin

Les affirmations ci–dessus étant admises, les tests de nullité et de régularité dans $R/[A]$: H_A^∞ ainsi que l'algorithme de forme normale sont exactement les mêmes pour les systèmes aux dérivées partielles cohérents que pour les systèmes différentiels ordinaires. À titre

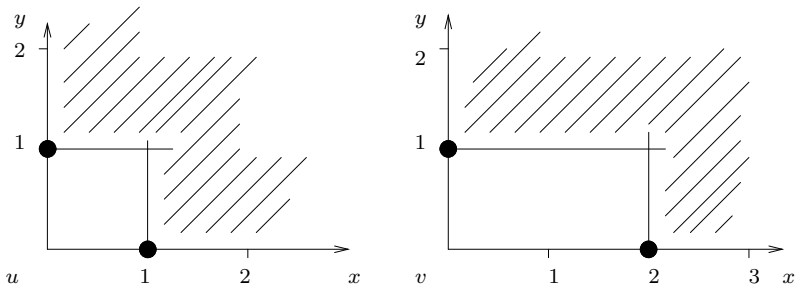
d'exemple, appliquons la fonction précédente sur le système $A = \{f_1,\, f_2,\, f_3,\, f_4\}$ suivant :

$$v_{xx} - u_x = 0, \quad v_y - \frac{u_x\, u_y}{4} = 0, \quad u_x^2 - 4\, u = 0, \quad u_y^2 - 2\, u = 0$$

pour le classement compatible avec l'ordre total suivant :

$$u < v < u_y < u_x < v_y < v_x < u_{yy} < u_{xy} < u_{xx} < v_{yy} < \cdots$$

Il est différentiellement triangulaire. L'ensemble de ses dérivées dominantes est l'ensemble $X = \{v_{xx},\, v_y,\, u_x,\, u_y\}$. L'ensemble $T = \{u,\, v,\, v_x\}$ des dérivées sous les escaliers de A est fini. Les voici représentées graphiquement. Il y a un diagramme par indéterminée différentielle, un axe par dérivation. On reporte les dérivées dominantes sous la forme de disques noirs et on hachure leurs dérivées. Les dérivées sous les escaliers sont les dérivées non hachurées.



On a $\mathrm{ppcd}(X) = \{v_{xxy},\, u_{xy}\}$. Graphiquement, ces dérivées correspondent aux angles des marches des escaliers. Les dérivées à considérer sont donc dans l'ordre

$$u_y,\, u_x,\, v_y,\, u_{xy},\, v_{xx},\, v_{xxy}.$$

Étape 1 : la dérivée u_y. Le polynôme $f_4 = u_y^2 - 2\, u$ est unitaire. Son séparant admet pour inverse le polynôme $u_y/(2\, u) \in K(T)[u_y]$. Étape 2 : la dérivée u_x. Le polynôme $f_3 = u_x^2 - 4\, u$ est unitaire. Son séparant admet pour inverse le polynôme $u_x/(4\, u) \in K(T)[u_y,\, u_x]$. Étape 3 : la dérivée v_y. Le polynôme f_2 admet 1 pour initial et pour séparant. Étape 4 : la dérivée u_{xy}. C'est la plus petite dérivée commune des dérivées dominantes de f_4 et de f_3.

$$\delta_x\, f_4 = 2\, u_y\, u_{xy} + 2\, u_x, \quad \delta_y\, f_3 = 2\, u_x\, u_{xy} + 4\, u_y.$$

Les calculs donnent deux formes normales égales :

$$\frac{u_x\, u_y}{2\, u} = \mathrm{NF}\left(-\frac{u_x}{u_y},\, (\Theta A)_{<u_{xy}}\right) \longleftarrow u_{xy} \longrightarrow \mathrm{NF}\left(-\frac{2\, u_y}{u_x},\, (\Theta A)_{<u_{xy}}\right) = \frac{u_x\, u_y}{2\, u}.$$

Étape 5 : la dérivée v_{xx}. Le polynôme $f_1 = v_{xx} - u_x$ admet 1 pour initial et pour séparant. Étape 6 : la dérivée v_{xxy}. C'est la plus petite dérivée commune des dérivées dominantes de f_1 et de f_2. Les calculs donnent deux formes normales égales (elles sont aussi égales aux précédentes mais c'est un hasard dû à l'exemple) :

$$\frac{u_x\, u_y}{2\, u} = \mathrm{NF}\left(u_{xy},\, (\Theta A)_{<v_{xxy}}\right) \longleftarrow v_{xxy} \longrightarrow \mathrm{NF}\left(\frac{u_{xxx}\, u_y + 2\, u_{xx}\, u_{xy} + u_x\, u_{xxy}}{4},\, (\Theta A)_{<v_{xxy}}\right) =$$
$$\mathrm{NF}\left(\frac{0 + 2 \times 2 \times u_{xy} + 0}{4},\, (\Theta A)_{<v_{xxy}}\right) = \frac{u_x\, u_y}{2\, u}.$$

La fonction retourne vrai. Le système de quatre polynômes différentiels constitue bien une chaîne différentielle régulière.

5.6 Attributs des chaînes différentielles régulières

Dans les bibliothèques *BLAD*, une chaîne régulière peut porter deux types d'attributs : les attributs « structurels » qui ne peuvent pas être changés par algorithme (ce sont des propriétés de l'idéal défini par la chaîne) et les attributs « désirés » qui peuvent, eux, être obtenus par algorithme (ce sont des propriétés de la chaîne elle–même). Soit A une chaîne régulière. Il y a deux attributs structurels :

differential s'il est satisfait, l'idéal défini par A est l'idéal différentiel $[A] : H_A^\infty$ sinon, c'est l'idéal non différentiel $(A) : I_A^\infty$.

prime s'il est satisfait, l'idéal défini par A est premier sinon il ne l'est pas nécessairement.

Il y a cinq attributs désirés. On les énonce en employant le vocabulaire différentiel :

autoreduced s'il est satisfait, la chaîne est complètement autoréduite.

primitive s'il est satisfait, chacun des polynômes qui constituent la chaîne est « primitif » dans le sens suivant : si on le regarde comme un polynôme en sa dérivée dominante et à coefficients dans l'anneau des polynômes en les autres dérivées alors le pgcd de ses coefficients vaut 1.

normalized s'il est satisfait, les initiaux des polynômes de la chaîne ne dépendent d'aucune des dérivées dominantes de la chaîne.

squarefree s'il est satisfait, le séparant de chaque polynôme p_i de la chaîne est inversible modulo l'idéal défini par les polynômes $p_1, \ldots, p_i \in A$ de dérivée dominante inférieure ou égale à la dérivée dominante de p_i. Si l'attribut structurel *differential* est satisfait alors l'attribut *squarefree* l'est aussi.

coherent s'il est satisfait, la chaîne différentielle régulière A est cohérente. Ce qualificatif n'est pertinent que pour les systèmes aux dérivées partielles.

L'attribut « *normalized* » correspond en fait à la propriété de « normalisation forte » décrite dans [12]. Satisfaire cette propriété peut être coûteux puisque cela nécessite en général des calculs d'inverses algébriques.

Concrètement, l'attribut « *squarefree* » est satisfait si la fonction *inverse_algébrique* (ou plutôt sa variante qui n'explicite pas les inverses mais teste leur existence) permet d'inverser chaque séparant modulo l'idéal défini par la chaîne. Le qualificatif « *squarefree* » (on dit aussi « *sans facteur carré* » ou « *séparable* ») a été choisi par analogie avec le cas des polynômes en une indéterminée à coefficients dans un corps. Prenons l'exemple du polynôme $p = (x+1)^2 (x-2)$. Son séparant $\partial p/\partial x = (x+1)(3x-1)$ contient en facteur le facteur irréductible multiple (appelé encore « facteur carré ») de p. Le séparant de p n'est donc pas inversible modulo l'idéal engendré par p. Un polynôme p dont le séparant est inversible modulo l'idéal (p) est un polynôme dont tous les facteurs irréductibles sont simples : un polynôme sans facteur carré.

Proposition 25 *Si A est une chaîne régulière sans facteur carré alors $(A) : I_A^\infty = (A) : H_A^\infty$.*

Preuve L'inclusion $(A) : I_A^\infty \subset (A) : H_A^\infty$ est claire. L'autre inclusion. Soit $p \in (A) : H_A^\infty$. Il existe alors un produit h de séparants de A tel que $h\,p \in (A) : I_A^\infty$ c'est–à–dire tel que $h\,p = 0$ dans $R/(A) : I_A^\infty$. Comme h est régulier dans cet anneau, $p = 0$ et donc $p \in (A) : I_A^\infty$. □

La proposition suivante est un corollaire du lemme de Lazard. Elle est plus difficile à prouver que la précédente. Voir [13, Corollary 3.3] par exemple.

Proposition 26 *Si A est une chaîne (différentielle) régulière sans facteur carré alors l'idéal (différentiel) qu'elle définit est radical.*

Chapitre 6

Applications algorithmiques des chaînes différentielles régulières

Ce chapitre décrit plusieurs applications algorithmiques directes du chapitre 5. Ce sont des applications internes à la théorie. Le chapitre est découpé en deux parties. Dans la première, on considère des applications de l'algorithme de forme normale modulo un idéal défini par une chaîne différentielle régulière. Dans la seconde, on présente les idées maîtresses d'algorithmes pour calculer des chaînes différentielles régulières en s'appuyant uniquement sur l'algorithmique du chapitre 5. Les détails sont disponibles dans les articles joints en annexe. L'algorithme général *Rosenfeld–Gröbner* n'est pas présenté ici mais dans le chapitre 9.

6.1 Applications de l'algorithme de forme normale

6.1.1 Un critère de primalité

La proposition suivante généralise un peu un théorème classique de Ritt sur les systèmes « orthonomiques » c'est–à–dire les systèmes où les dérivées dominantes apparaissent linéairement avec un initial égal à 1. C'est surtout l'argument de la preuve du critère qui est intéressant : un polynôme ne peut diviser zéro modulo l'idéal défini par un chaîne régulière que s'il fait échouer la fonction *inverse_ algebrique* et exhibe donc une factorisation d'un élément de la chaîne. On peut l'adapter au cas par cas pour démontrer que certains idéaux sont premiers. On doit aussi pouvoir s'en inspirer pour obtenir des algorithmes de décomposition en idéaux premiers.

Proposition 27 *Soit $A = \{p_1, \ldots, p_t\}$ une chaîne différentielle régulière d'un anneau de polynômes différentiels R. Si $\deg(p_i, \mathrm{ld}\, p_i) = 1$ pour tous $1 \leq i \leq t$ alors l'idéal différentiel $[A] : H_A^\infty$ est premier.*

Preuve Si l'idéal n'était pas premier, alors il existerait un polynôme $p \in R$ non inversible modulo $[A] : H_A^\infty$. L'appel de fonction *inverse_ algebrique* (p, A) échouerait en exhibant une

factorisation non triviale d'un $p_i \in A$ mais c'est impossible parce que les polynômes p_i sont de degré 1 en leur dérivée dominante. □

Pour une chaîne différentielle orthonomique A, il existe une autre preuve, fondée sur l'existence de l'algorithme de forme normale : l'anneau $R/[A] : H_A^\infty$ est isomorphe à l'algèbre des formes normales modulo A. Si A est orthonomique alors l'algèbre des formes normales est une algèbre libre : c'est l'anneau des polynômes en les dérivées sous les escaliers de A. C'est un anneau intègre donc l'idéal est premier.

6.1.2 Détecter des dépendances linéaires dans un anneau quotient

Soient A une chaîne régulière (éventuellement différentielle) d'un anneau R et $\mathfrak{A}$ l'idéal qu'elle définit. Disposer d'un algorithme de forme normale permet de tester l'égalité entre deux expressions dans l'anneau $R/\mathfrak{A}$. C'est une application des formes normales bien connue des spécialistes de la théorie de la réécriture. En algèbre commutative et différentielle en tous cas, on dispose d'une application supplémentaire, qui est au cœur de l'algorithme $FGLM$ [41] : elles permettent de chercher des dépendances linéaires entre éléments de $R/\mathfrak{A}$. L'argument est élémentaire : si R est une algèbre sur K alors toute combinaison linéaire sur K de formes normales est encore une forme normale. Soit $\{g_1, \ldots, g_k\}$ une famille d'éléments de R. On cherche des coefficients $\lambda_1, \ldots, \lambda_k \in K$ tels que

$$\lambda_1\, g_1 + \cdots + \lambda_k\, g_k = 0 \mod \mathfrak{A}.$$

Il suffit de chercher (et c'est suffisant) des coefficients $\lambda_1, \ldots, \lambda_k \in K$ tels que

$$\lambda_1\ \mathrm{NF}(g_1, A) + \cdots + \lambda_k\ \mathrm{NF}(g_k, A) = 0.$$

À titre d'exemple, reprenons le système $A = \{f_1,\ f_2,\ f_3,\ f_4\}$ suivant :

$$v_{xx} - u_x = 0, \quad v_y - \frac{u_x\, u_y}{4} = 0, \quad u_x^2 - 4\, u = 0, \quad u_y^2 - 2\, u = 0$$

pour le classement compatible avec l'ordre total :

$$u < v < u_y < u_x < v_y < v_x < u_{yy} < u_{xy} < u_{xx} < v_{yy} < \cdots$$

et cherchons s'il existe une équation différentielle linéaire à coefficients rationnels ne dépendant que de v et de ses dérivées dans l'idéal différentiel défini par la chaîne différentielle régulière.

Il suffit d'énumérer les dérivées de v, de calculer leur forme normale par la chaîne différentielle régulière connue et de chercher des dépendances linéaires. On le fait ici à l'œil nu

mais on peut automatiser le procédé par un pivot de Gauss :

dérivée	forme normale
v	v
v_y	$\dfrac{u_x\,u_y}{4}$
v_x	v_x
v_{yy}	$\dfrac{u_x}{2}$
v_{xy}	u_y
v_{xx}	u_x

On voit immédiatement que $v_{xx} - 2\,v_{yy} \in \mathfrak{A}$. Et cette équation est nécessairement d'ordre minimale parmi celles qu'on cherche. On peut aussi s'intéresser aux équations non linéaires. On voit immédiatement que $v_{xy}\,v_{yy} - 2\,v_y \in \mathfrak{A}$.

Dans le cas très fréquent des systèmes orthonomiques, les algèbres de formes normales sont des algèbres libres. On peut donc évaluer numériquement les formes normales et chercher des dépendances linéaires entre des vecteurs de nombres.

J'ai publié ces idées avec quelques approfondissements fondés sur l'usage des différentielles de Kähler (voir ci–dessous) dans [8].

6.1.3 Calculer dans les modules de différentielles de Kähler

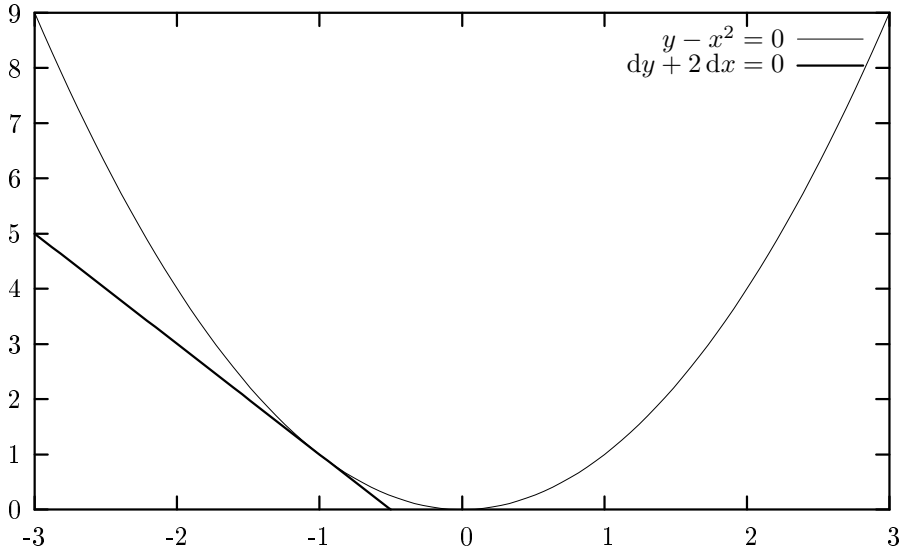
Avant de présenter algébriquement les modules de différentielles de Kähler, je présente une façon de les comprendre. Considérons le polynôme

$$y - x^2.$$

La variété algébrique qu'il définit (l'ensemble des zéros du polynôme) forme une parabole. La différentielle de ce polynôme est l'expression linéaire

$$\mathrm{d}y - 2\,x\,\mathrm{d}x$$

qu'on peut interpréter comme suit : si on identifie l'axe des x et l'axe des $\mathrm{d}x$ d'une part, l'axe des y et celui des $\mathrm{d}y$ d'autre part, alors en tout zéro $(x,\,y) \in \mathbb{R}^2$ du polynôme, l'ensemble des couples $(\mathrm{d}x,\,\mathrm{d}y) \in \mathbb{R}^2$ tels que $\mathrm{d}y - 2\,x\,\mathrm{d}x = 0$ décrit la tangente à la parabole en $(x,\,y)$. Sur le graphique suivant, on a tracé la parabole ainsi que la droite $\mathrm{d}y - 2\,x\,\mathrm{d}x = 0$ pour $(x,\,y) = (-1,\,1)$.



La même construction s'applique aux équations différentielles ordinaires et aux dérivées partielles. Considérons le polynôme différentiel ordinaire

$$\dot{x} - x^2.$$

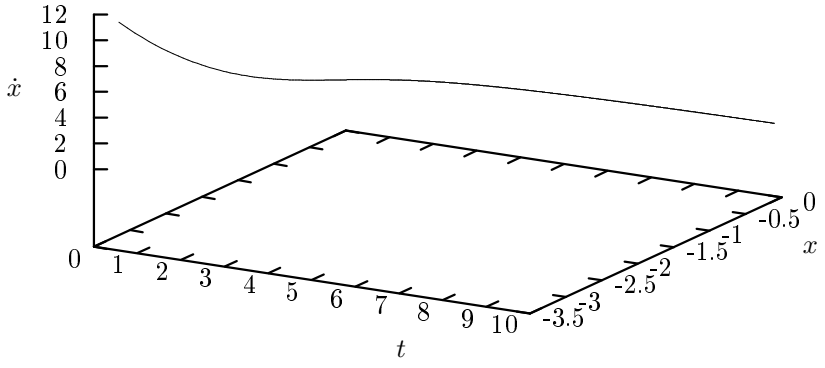
La variété algébrique différentielle qu'il définit est l'ensemble de fonctions

$$x(t) = -\frac{1}{t-c} \quad \text{où } c \in \mathbb{R}.$$

Pour chacune des fonctions ci–dessus, on a

$$\dot{x}(t) = \frac{1}{(t-c)^2}.$$

Pour une valeur de c fixée, l'ensemble des triplets $(t,\, x(t),\, \dot{x}(t))$ décrit une courbe dans $\mathbb{R}^3$ dont voici un exemple.



La différentielle du polynôme différentiel $\dot{x} - x^2$ est l'expression linéaire

$$\mathrm{d}\dot{x} - 2\,x\,\mathrm{d}x$$

qu'on peut interpréter comme suit : si on identifie l'axe des x et l'axe des $\mathrm{d}x$, l'axe des $\dot{x}$ et l'axe des $\mathrm{d}\dot{x}$, l'axe des t avec l'axe des $\mathrm{d}t$, alors en tout point $(t,\, x(t),\, \dot{x}(t))$ de la courbe, l'ensemble des triplets $(t,\, \mathrm{d}x(t),\, \mathrm{d}\dot{x}(t))$ décrit un plan, tangent en $(t,\, x(t),\, \dot{x}(t))$, et perpendiculaire au plan $(x,\, \dot{x})$. Pour obtenir la tangente à la courbe, il faudrait intersecter ce plan avec le plan défini par la forme de contact $\mathrm{d}x - \dot{x}\,\mathrm{d}t$ mais ce n'est pas nécessaire pour notre explication. Projetons maintenant les graphiques sur le plan $(x,\, \dot{x})$. La courbe devient une parabole $(x(t),\, \dot{x}(t))$. Les plans tangents deviennent des tangentes $(\mathrm{d}x(t),\, \mathrm{d}\dot{x}(t))$ à la parabole. Si on intègre le polynôme différentiel pour une certaine condition initiale on obtient un point qui se déplace au cours du temps le long de la parabole. Si on intègre en même temps sa différentielle pour une autre condition initiale, on obtient un vecteur tangent à la parabole qui se déplace avec le point. Sur l'exemple, les intégrations peuvent être menées formellement. Elles consistent à résoudre le système

$$\dot{x} - x^2 = 0, \quad \mathrm{d}\dot{x} - 2\,x\,\mathrm{d}x = 0.$$

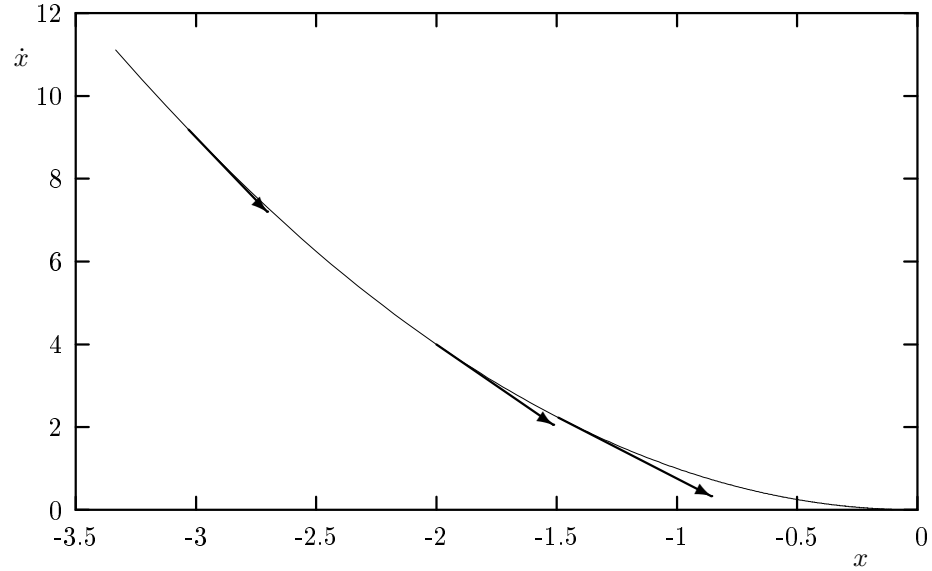
On trouve

$$x(t) = -\frac{1}{t-c} \quad \text{et} \quad \mathrm{d}x(t) = \frac{c'}{(t-c)^2}, \quad \text{où } c,\, c' \in \mathbb{R}$$

et donc en évaluant les équations du système

$$\dot{x}(t) = \frac{1}{(t-c)^2} \quad \text{et} \quad \mathrm{d}\dot{x}(t) = -\frac{2\,c'}{(t-c)^3}.$$

Le graphique suivant a été obtenu pour un couple $(c,\, c')$ fixé, en faisant varier t sur un intervalle et en évaluant les quatre fractions rationnelles ci–dessus. Pour garder un graphique lisible, on n'a tracé que trois vecteurs $(\mathrm{d}x(t),\, \mathrm{d}\dot{x}(t))$ d'origine $(x(t),\, \dot{x}(t))$ et on les a normalisés. On constate visuellement qu'ils sont tangents à la parabole.



Dernière constatation : l'opération « d », qui à un polynôme différentiel associe sa différentielle, commute avec la dérivation par rapport au temps. La fonction $\mathrm{d}\dot{x}(t)$, dont l'expression analytique est donnée ci–dessus, est égale non seulement à $2\,x(t)\,\mathrm{d}x(t)$ mais aussi à la dérivée par rapport au temps de la fonction $\mathrm{d}x(t)$. Dit autrement,

$$\mathrm{d}(\dot{x}(t)) = \dot{\widehat{\mathrm{d}x}}(t).$$

En un sens, c'est évident mais on peut prendre un autre exemple pour achever de se convaincre : considérons la différentielle de la dérivée du polynôme différentiel qui nous sert d'exemple. Elle permet de définir $\mathrm{d}\ddot{x}$.

$$\mathrm{d}(\ddot{x} - 2\,x\,\dot{x}) = \mathrm{d}\ddot{x} - 2\,x\,\mathrm{d}\dot{x} - 2\,\dot{x}\,\mathrm{d}x.$$

On obtient la valeur de $\mathrm{d}\ddot{x}$ en remplaçant x, $\dot{x}$, $\mathrm{d}x$, $\mathrm{d}\dot{x}$ par leur valeur. On constate qu'on trouve la même fonction qu'en dérivant $\mathrm{d}\dot{x}$ par rapport au temps :

$$\mathrm{d}\ddot{x} = \frac{6\,c'}{(t-c)^4}.$$

Formalisation

Les calculs menés sur l'exemple se généralisent. Ils sont formalisés dans la théorie des différentielles d'Erwin Kähler [53] (voir aussi [37, chapitre 16]) dont l'adaptation au cas des systèmes différentiels est due à Joseph Johnson [52].

Définition 10 *Si K est un corps et G est une algèbre sur K alors le module des différentielles de Kähler de G sur K, noté $\Omega_{G/K}$, est le module sur G engendré par l'ensemble $\{\mathrm{d}b \mid b \in G\}$ tel que*

$$\begin{array}{rcll} \mathrm{d}(b+b') & = & \mathrm{d}b + \mathrm{d}b' & \textit{pour tous } b,\, b' \in G \\ \mathrm{d}(b\,b') & = & b\,\mathrm{d}b' + b'\,\mathrm{d}b & \textit{pour tous } b,\, b' \in G \\ \mathrm{d}a & = & 0 & \textit{pour tout } a \in K \end{array}$$

L'opérateur « d » commute avec les dérivations :

Proposition 28 *Si K est un corps différentiel et G est une algèbre différentielle sur K alors $\Omega_{G/K}$ peut être canoniquement muni d'une structure de module différentiel sur G tel que, pour toute dérivation δ sur G, on ait $\mathrm{d}\,\delta b = \delta\,\mathrm{d}b$.*

Preuve Voir [52, Proposition, page 93]. □

Le théorème suivant est purement algébrique. Il est très important algorithmiquement parce qu'il permet de réduire la question de l'existence d'une dépendance algébrique entre des éléments en la question de l'existence d'une dépendance linéaire entre leurs différentielles.

Théorème 2 *Si K est un corps de caractéristique nulle et G est une extension de corps de K alors les éléments $\eta_1, \ldots, \eta_r$ de G sont algébriquement indépendants sur K si et seulement si les éléments $\mathrm{d}\eta_1, \ldots, \mathrm{d}\eta_r$ sont linéairement indépendants sur G.*

Preuve Voir [37, Theorem 16.14, page 400] ou [52, Lemma, page 94]. □

On verra qu'on va appliquer le théorème 2 dans le cas où G est le corps des fractions du quotient $R/\mathfrak{p}$ d'un anneau de polynômes R par l'un de ses idéaux premiers, lui–même présenté par une chaîne différentielle régulière. En général, les chaînes différentielles régulières ne définissent pas des idéaux premiers mais des idéaux radicaux (lemme de Lazard). L'anneau total des fractions[1] de l'anneau $R/\mathfrak{A}$ dans le cas où $\mathfrak{A}$ est radical n'est pas un corps mais un « produit de corps » c'est–à–dire un anneau produit dont les composantes sont des corps. C'est même un produit de corps différentiels dans le cas où $\mathfrak{A}$ est un idéal différentiel (lifting du lemme de Lazard). Avec des mots simples, cela ne change rien sur le fond puisqu'il suffit de traiter l'anneau total des fractions de $R/\mathfrak{A}$ comme un corps jusqu'à–ce qu'on s'aperçoive qu'il ne l'est pas (principe D^5). Plus rigoureusement, il suffit d'appliquer la proposition suivante.

Proposition 29 *Si $G_1, \ldots, G_r$ sont des algèbres sur K et $G = G_1 \times \cdots \times G_r$ alors*

$$\Omega_{G/K} = \Omega_{G_1/K} \times \cdots \times \Omega_{G_r/K}.$$

Preuve [37, Proposition 16.10, page 398]. □

Forme normale d'une différentielle de Kähler

Soit A une chaîne différentielle régulière d'un anneau $R = K\{U\}$ pour un classement $\mathscr{O}$. En s'appuyant sur le fait que l'opérateur « d » commute avec les dérivations, on assimile la différentielle $\mathrm{d}u$ de chaque indéterminée différentielle $u \in U$ à une nouvelle indéterminée différentielle. On étend le classement $\mathscr{O}$ en un classement noté

$$\mathrm{d}\mathscr{O} \gg \mathscr{O}$$

de la façon suivante :

1. toute dérivée de $\Theta\mathrm{d}U$ est supérieure à toute dérivée de ΘU ;
2. les dérivées de ΘU sont ordonnées suivant $\mathscr{O}$;
3. pour toutes dérivées $\mathrm{d}v$, $\mathrm{d}w \in \Theta\mathrm{d}U$ on pose $\mathrm{d}v < \mathrm{d}w$ si et seulement si $v < w$.

À titre d'exemple, reprenons le système $A = \{f_1,\ f_2,\ f_3,\ f_4\}$ suivant :

$$v_{xx} - u_x = 0, \quad v_y - \frac{u_x\, u_y}{4} = 0, \quad u_x^2 - 4\,u = 0, \quad u_y^2 - 2\,u = 0$$

[1] L'anneau total des fractions d'un anneau R s'obtient en rendant inversibles tous les non diviseurs de zéro de R. Si R est intègre, son anneau total des fractions est égal à son corps des fractions.

pour le classement compatible avec l'ordre total :

$$\cdots > v_{yy} > u_{xx} > u_{xy} > u_{yy} > v_x > v_y > u_x > u_y > v > u.$$

L'ensemble $\mathrm{d}A$ est

$$\mathrm{d}v_{xx} - \mathrm{d}u_x = 0, \quad \mathrm{d}v_y - \frac{u_x\,\mathrm{d}u_y + u_y\,\mathrm{d}u_x}{4} = 0, \quad 2\,u_x\,\mathrm{d}u_x - 4\,\mathrm{d}u = 0, \quad 2\,u_y\,\mathrm{d}u_y - 2\,\mathrm{d}u = 0.$$

Le classement $\mathrm{d}\mathscr{O} \gg \mathscr{O}$ est

$$\cdots > \mathrm{d}v_y > \mathrm{d}u_x > \mathrm{d}u_y > \mathrm{d}v > \mathrm{d}u > \cdots > v_y > u_x > u_y > v > u.$$

Proposition 30 *Si A est une chaîne différentielle régulière de $K\{U\}$ pour un classement $\mathscr{O}$ alors $A \cup \mathrm{d}A$ est une chaîne différentielle régulière de $K\{U \cup \mathrm{d}U\}$ pour le classement $\mathrm{d}\mathscr{O} \gg \mathscr{O}$.*

Preuve On note $\mathfrak{A}$ l'idéal différentiel défini par la chaîne A. Si $p \in A$ admet v pour dérivée dominante alors $\mathrm{d}p$ admet $\mathrm{d}v$ pour dérivée dominante et le séparant de p pour initial et séparant. Le système $\mathrm{d}A$ est donc différentiellement triangulaire. Les initiaux et les séparants de ce système vérifient les mêmes conditions de régularité que ceux de A. Reste, dans le cas des systèmes aux dérivées partielles, le problème de la cohérence. Considérons le Δ–polynôme engendré par deux éléments de $\mathrm{d}A$ complètement réduit par $A \cup \mathrm{d}A$ pour la réduction de Ritt. Il est nécessairement nul puisqu'il fournirait autrement une relation linéaire sur l'anneau total des fractions G de $R/\mathfrak{A}$ entre les différentielles des dérivées sous les escaliers de A (on utilise le fait que les classements sur $\mathscr{O}$ et $\mathrm{d}\mathscr{O}$ correspondent) et donc, d'après le théorème 2, une dépendance algébrique entre les dérivées sous les escaliers de A. Or l'ensemble des dérivées sous les escaliers de A est algébriquement indépendant modulo $\mathfrak{A}$. Le système $A \cup \mathrm{d}A$ est donc cohérent. □

Avant d'énoncer la définition suivante, on note que l'algorithme de calcul de la forme normale d'un polynôme différentiel modulo une chaîne différentielle régulière s'étend sans difficulté au calcul de la forme normale d'une fraction rationnelle différentielle modulo une chaîne différentielle régulière. Cette extension de l'algorithme de forme normale permet ainsi de calculer dans un produit de corps différentiel présenté par générateurs et relations.

Définition 11 *Soient A une chaîne différentielle régulière de $R = K\{U\}$ et $\mathfrak{A}$ l'idéal différentiel qu'elle définit. Soit G l'anneau total des fractions de $R/\mathfrak{A}$. Pour toute différentielle $b \in \Omega_{G/K}$ on définit la forme normale de b dans $\Omega_{G/K}$ comme la forme normale de b, vue comme une fraction rationnelle de deux polynômes de $K\{U \cup \mathrm{d}U\}$ par la chaîne différentielle régulière $A \cup \mathrm{d}A$:*

$$\underset{\Omega_{G/K}}{\mathrm{NF}}\,(b) = \mathrm{NF}(b,\, A \cup \mathrm{d}A).$$

Calculons à titre d'exemple les formes normales des différentielles $\mathrm{d}v_{yy}$ et $\mathrm{d}v_y$ dans $\Omega_{G/K}$ où G est l'anneau total des fractions de $R/\mathfrak{A}$ et $\mathfrak{A}$ est l'idéal défini par la chaîne différentielle régulière A ci–dessus. Pour simplifier les calculs, on commence par calculer les formes normales des dérivées v_{yy} et v_y par la chaîne A (en fait, il suffit de consulter le tableau de la section 6.1.2, page 87). On calcule ensuite les formes normales des différentielles de ces fractions rationnelles.

$$\operatorname*{NF}_{\Omega_{G/K}}(\mathrm{d}v_{yy}) = \mathrm{NF}\left(\frac{\mathrm{d}u_x}{2},\, A \cup \mathrm{d}A\right) = \mathrm{NF}\left(\frac{\mathrm{d}u}{u_x},\, A \cup \mathrm{d}A\right) = \frac{u_x}{4\,u}\,\mathrm{d}u.$$

$$\operatorname*{NF}_{\Omega_{G/K}}(\mathrm{d}v_{y}) = \mathrm{NF}\left(\frac{u_x\,\mathrm{d}u_y + u_y\,\mathrm{d}u_x}{4},\, A \cup \mathrm{d}A\right) = \frac{u_x\,u_y}{4\,u}\,\mathrm{d}u.$$

On observe que ces deux formes normales sont linéairement dépendantes sur G puisqu'on a

$$u_y\,\mathrm{d}v_{yy} - \mathrm{d}v_y = 0.$$

Le théorème 2 s'applique : les dérivées v_{yy} et v_y sont algébriquement dépendantes sur K modulo l'idéal $\mathfrak{A}$. En énumérant les formes normales des termes de la forme $v_{yy}^a\,v_y^b$ par degré total croissant, on vérifie que

$$v_{yy}^4 - 2\,v_y^2 \in \mathfrak{A}.$$

Le programme C suivant vérifie les calculs ci–dessus. Les trois premières formes normales calculées sont les formes normales de trois différentielles de Kähler. Elles sont calculées en appliquant la définition 11 à la lettre. Les formes normales qui suivent sont des formes normales de termes de la forme $v_{yy}^a\,v_y^b$.

```
/* File dfglm.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bap_tableof_ratfrac_mpz T;
    struct bap_ratfrac_mpz NF;
    struct bad_regchain C;
    struct ba0_mark M;
    int i;

    bad_restart (0, 0);
    ba0_record (&M);

    ba0_sscanf2 ("ordering (derivations = [x, y], blocks = [[dv, du], [v, u]])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
```

```
    bad_init_regchain (&C);
    bap_init_ratfrac_mpz (&NF);
    ba0_init_table ((ba0_table)&T);
/*
   The system A union dA
*/
    ba0_sscanf2
        ("regchain ([dv[x,x] - du[x], v[x,x] - u[x], \
                    dv[y] - (u[x]*du[y] + u[y]*du[x])/4, v[y] - u[x]*u[y]/4, \
                    2*u[x]*du[x] - 4*du, u[x]^2 - 4*u, \
                    2*u[y]*du[y] - 2*du, u[y]^2 - 2*u], \
        [prime, differential, coherent, squarefree, autoreduced, primitive])",
         "%regchain", &C);
    ba0_sscanf2 ("[dv, dv[y], dv[y,y], \
                   1, v[y,y], v[y], v[y,y]^2, v[y,y]*v[y], v[y]^2, v[y,y]^4]",
                 "%t[%Qz]", &T);

    for (i = 0; i < T.size; i++)
    {   bad_normal_form_ratfrac_mod_regchain
                        (&NF, T.tab [i], &C, (bap_polynom_mpz*)0);
        ba0_printf ("normalf form (%Qz) = %Qz\n", T.tab [i], &NF);
    }

    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici les affichages obtenus à l'exécution.

```
normalf form (dv) = dv
normalf form (dv[y]) = (du*u[x]*u[y])/(4*u)
normalf form (dv[y,y]) = (du*u[x])/(4*u)
normalf form (1) = 1
normalf form (v[y,y]) = (u[x])/(2)
normalf form (v[y]) = (u[x]*u[y])/(4)
normalf form (v[y,y]^2) = u
normalf form (v[y,y]*v[y]) = (u[y]*u)/(8)
normalf form (v[y]^2) = (u^2)/(2)
normalf form (v[y,y]^4) = u^2
```

Les raisonnements tenus sur l'exemple se généralisent.

Proposition 31 *Un ensemble de dérivées $\{v_1, \ldots, v_r\}$ de ΘU est algébriquement indépendant sur K si l'ensemble des formes normales de ses différentielles, vues comme des vecteurs d'éléments de G dans la base $\Theta \mathrm{d}U$, est linéairement indépendant sur G.*

On remarque que les tests de dépendance linéaire entre les différentielles doivent être effectués au–dessus d'un corps différentiel (et même d'un produit de corps différentiels) présenté par générateurs et relations. Il faut être capable de reconnaître zéro dans G. L'algorithme de forme normale en fait plus mais suffit pour cela.

Dans certains cas, où les algèbres de formes normales sont des algèbres libres, on peut utiliser des algorithmes semi–numériques en évaluant les formes normales en des nombres et en cherchant les dépendances linéaires entre vecteurs de nombres. Cette situation se produit pour les équations d'Euler pour un fluide incompressible (voir en section 6.2.3, page 105). En utilisant un algorithme semi–numérique (et donc probabiliste), j'ai pu (presque) établir que l'idéal différentiel engendré par les équations contient un polynôme différentiel qui ne dépend que des dérivées suivantes de la pression :

$$p_{ttxxx},\ p_{ttxxy},\ p_{ttxyy},\ p_{ttyyy},\ p_{txxxx},\ p_{txxxy},\ p_{txxyy},\ p_{txyyy},\ p_{tyyyy},\ p_{xxxxx},\ p_{xxxxy},\ p_{xxxyy},$$
$$p_{xxyyy},\ p_{xyyyy},\ p_{yyyyy},\ p_{ttxx},\ p_{ttyy},\ p_{txxx},\ p_{txxy},\ p_{txyy},\ p_{tyyy},\ p_{xxxx},\ p_{xxxy},\ p_{xxyy},\ p_{xyyy},\ p_{yyyy},$$
$$p_{txx},\ p_{txy},\ p_{tyy},\ p_{xxx},\ p_{xxy},\ p_{xyy},\ p_{yyy},\ p_{xx},\ p_{xy},\ p_{yy},\ p_x,\ p_y$$

Application aux algorithmes de changement de classement

Soit A une chaîne différentielle régulière pour un certain classement $\mathscr{O}$ d'un anneau de polynômes différentiels $R = K\{U\}$. On cherche une chaîne différentielle régulière $\bar{A}$ définissant le même idéal différentiel $\mathfrak{A}$ que A mais pour un autre classement $\bar{\mathscr{O}}$.

La méthode décrite ci–dessus permet de déterminer, étant donné un ensemble de dérivées $\{v_1, \ldots, v_r\} \subset \Theta U$, si $\mathfrak{A} \cap K[v_1, \ldots, v_r]$ contient un polynôme non nul ou pas. Supposons qu'on sache que $\mathfrak{A} \cap K[v_1, \ldots, v_r] \neq (0)$. On peut alors énumérer les termes de la forme $v_1^{a_1} \cdots v_r^{a_r}$ par degré total croissant jusqu'à détecter une dépendance linéaire sur K entre eux par la méthode décrite en section 6.1.2. On obtient ainsi à coup sûr un élément non nul de $\mathfrak{A} \cap K[v_1, \ldots, v_r]$.

Dans le cadre du problème de changement de classement, l'idée consiste à s'intéresser à des ensembles $\{v_1, \ldots, v_r\}$ dont les éléments sont aussi petits que possible vis–à–vis du nouveau classement $\bar{\mathscr{O}}$ et donc à chercher des polynômes différentiels de petite dérivée dominante vis–à–vis de $\bar{\mathscr{O}}$ dans l'espoir qu'ils aideront à calculer la chaîne différentielle régulière $\bar{A}$. Cette idée est confortée par le fait que les polynômes différentiels intermédiaires engendrés par des algorithmes de facture classique tels que *PARDI* sont souvent beaucoup plus gros que les polynômes de $\bar{A}$.

Dans le cas particulier d'un idéal différentiel $\mathfrak{A}$ dont les solutions dépendent d'un nombre fini de constantes arbitraires, on sait que le nombre de dérivées sous les escaliers de toutes les chaînes différentielles qui décrivent $\mathfrak{A}$ est toujours le même : c'est le nombre de constantes arbitraires[2]. On peut alors appliquer un schéma algorithmique très fortement inspiré de celui de l'algorithme *FGLM* [41] pour calculer, pour chaque polynôme de la chaîne cible $\bar{A}$, l'ensemble des dérivées dont il dépend. Remarquer que les polynômes ainsi obtenus ne sont

[2]Ce nombre de constantes arbitraires est le degré de transcendance du corps des fractions de $R/\mathfrak{p}$ sur K où $\mathfrak{p}$ désigne un idéal différentiel premier minimal sur $\mathfrak{A}$ quelconque (lifting du lemme de Lazard).

pas nécessairement des éléments de la chaîne $\bar{A}$: il ne sont pas forcément de degré minimal en leur dérivée dominante et il n'y a aucune raison que leur coefficient dominant satisfasse les conditions de régularité nécessaires.

6.1.4 Calculer des solutions en série formelle

Dans le texte qui suit, on appelle « solution (en série) d'une chaîne différentielle régulière A » toute solution de l'idéal différentiel $[A] : H_A^\infty$. L'utilisation d'un algorithme de forme normale n'est pas indispensable pour expliquer comment calculer des développements en série formelle de solutions de chaînes différentielles régulières [10, 89] mais elle simplifie la présentation.

Soit A une chaîne différentielle régulière d'un anneau de polynômes différentiels $R = K\{u_1, \ldots, u_n\}$ muni de m dérivations. On suppose dans cette section que $K = \mathbb{Q}$ et que les m dérivations sont les m dérivées partielles par rapport à m variables indépendantes $x_1, \ldots, x_m$. Si

$$\theta = \frac{\partial^{a_1+\cdots+a_m}}{\partial x_1^{a_1} \cdots \partial x_m^{a_m}}$$

est un opérateur de dérivations, on note $\theta! = a_1! \cdots a_m!$ et $x^\theta = x_1^{a_1} \cdots a_m^{a_m}$. Avec ces notations, les solutions en série formelle de l'idéal $[A] : H_A^\infty$ centrées en l'origine sont des n–uplets $(\bar{u}_1, \ldots, \bar{u}_n)$ définis par :

$$\bar{u}_j = \sum_{\theta \in \Theta} \theta\, u_j(0, \ldots, 0)\, \frac{x^\theta}{\theta!}\cdot$$

Pour obtenir une expression des solutions en série formelle de l'idéal différentiel $[A] : H_A^\infty$ qui ne dépende que des dérivées sous les escaliers de A et, éventuellement, des dérivées dominantes de A, il suffit de remplacer chaque dérivée $\theta\, u_j$ dans la somme ci–dessus par sa forme normale :

$$\bar{u}_j = \sum_{\theta \in \Theta} \mathrm{NF}(\theta\, u_j,\, A)(0, \ldots, 0)\, \frac{x^\theta}{\theta!}\cdot$$

Le calcul du développement en série formelle des solutions d'une chaîne différentielle régulière n'est pas implanté en *BLAD* (par contre, il l'est en *diffalg*). On peut toutefois facilement calculer les formes normales des dérivées qui figurent dans les séries. C'est ce que fait le programme C suivant sur le système A :

$$v_{xx} - u_x = 0, \quad v_y - \frac{u_x\, u_y}{4} = 0, \quad u_x^2 - 4\, u = 0, \quad u_y^2 - 2\, u = 0$$

pour le classement compatible avec l'ordre total suivant :

$$u < v < u_y < u_x < v_y < v_x < u_{yy} < u_{xy} < u_{xx} < v_{yy} < \cdots$$

```
/* File serie_formelle.c */

#include "bad.h"
```

```
int main ()
{   bav_Iordering r;
    struct bap_tableof_polynom_mpz T;
    struct bap_ratfrac_mpz NF;
    struct bad_regchain C;
    struct ba0_mark M;
    int i;

    bad_restart (0, 0);
    ba0_record (&M);
    ba0_sscanf2 ("ordering (derivations = [x, y], blocks = [[v, u]])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bad_init_regchain (&C);
    ba0_sscanf2
        ("regchain ([v[x,x] - u[x], v[y] - u[x]*u[y]/4, \
                     u[x]^2 - 4*u, u[y]^2 - 2*u], [differential])",
         "%regchain", &C);
    ba0_init_table ((ba0_table)&T);
    ba0_sscanf2 ("[u, u[x], u[y], u[x,x], u[x,y], u[y,y], \
                   u[x,x,x], u[x,x,y], u[x,y,y], u[y,y,y], \
                   v, v[x], v[y], v[x,x], v[x,y], v[y,y], \
                   v[x,x,x], v[x,x,y], v[x,y,y], v[y,y,y], v[x,x,x,x], \
                   v[x,x,x,y], v[x,x,y,y], v[x,y,y,y], v[y,y,y,y]]",
                 "%t[%Az]", &T);
    bap_init_ratfrac_mpz (&NF);
    for (i = 0; i < T.size; i++)
    {   bad_normal_form_polynom_mod_regchain
                                (&NF, T.tab [i], &C, (bap_polynom_mpz*)0);
        ba0_printf ("normalf form (%Az) = %Qz\n", T.tab [i], &NF);
    }
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici les résultats affichés par le programme.

```
normalf form (u) = u
normalf form (u[x]) = u[x]
normalf form (u[y]) = u[y]
normalf form (u[x,x]) = 2
normalf form (u[x,y]) = (u[x]*u[y])/(2*u)
normalf form (u[y,y]) = 1
normalf form (u[x,x,x]) = 0
```

```
normalf form (u[x,x,y]) = 0
normalf form (u[x,y,y]) = 0
normalf form (u[y,y,y]) = 0
normalf form (v) = v
normalf form (v[x]) = v[x]
normalf form (v[y]) = (u[x]*u[y])/(4)
normalf form (v[x,x]) = u[x]
normalf form (v[x,y]) = u[y]
normalf form (v[y,y]) = (u[x])/(2)
normalf form (v[x,x,x]) = 2
normalf form (v[x,x,y]) = (u[x]*u[y])/(2*u)
normalf form (v[x,y,y]) = 1
normalf form (v[y,y,y]) = (u[x]*u[y])/(4*u)
normalf form (v[x,x,x,x]) = 0
normalf form (v[x,x,x,y]) = 0
normalf form (v[x,x,y,y]) = 0
normalf form (v[x,y,y,y]) = 0
normalf form (v[y,y,y,y]) = 0
```

On constate que les formes normales des dérivées de u d'ordre 3 et des dérivées de v d'ordre 4 sont nulles, ce qui prouve que les séries formelles $\bar{u}$ et $\bar{v}$ sont respectivement des polynômes de degré deux et trois.

On cherche des développements en série centrés sur l'origine. Il reste donc à évaluer les formes normales ci–dessus en $(x,\, y) = (0,\, 0)$. Les dérivées sous les escaliers de A sont u, v et v_x. On voit qu'on peut choisir pour $u(0,0)$ une valeur quelconque non nulle et des valeurs quelconques pour $v(0,0)$ et $v_x(0,0)$. Les autres dérivées figurant dans les formes normales sont des dérivées dominantes u_x et u_y de A. Les valeurs qu'on leur attribue doivent être compatibles avec les équations de A (les équations différentielles imposent des égalités entre fonctions c'est–à–dire pour toutes les valeurs de x et de y et donc, en particulier, à l'origine). Les équations à satisfaire sont ici :

$$u_x(0,0)^2 - 4\,u(0,0) = 0, \quad u_y(0,0)^2 - 2\,u(0,0) = 0.$$

Choisissons donc trois constantes arbitraires c_0, c_1, $c_2 \in \mathbb{R}$ telles que $c_0 \neq 0$ et c_1, $c_2 \geq 0$ puis posons

$$(u(0,0),\, v(0,0),\, v_x(0,0),\, u_x(0,0),\, u_y(0,0)) = (c_0,\, c_1,\, c_2,\, 2\,\sqrt{c_0},\, \sqrt{2\,c_0}).$$

On trouve les séries formelles (des polynômes dans ce cas–ci) :

$$\begin{aligned} u(x,y) &= c_0 + 2\,\sqrt{c_0}\,x + \sqrt{2\,c_0}\,y + x^2 + \sqrt{2}\,x\,y + \frac{1}{2}\,y^2, \\ v(x,y) &= c_1 + c_2\,x + \frac{\sqrt{2}\,c_0}{2}\,y + \sqrt{c_0}\,x^2 + \sqrt{2\,c_0}\,x\,y + \frac{\sqrt{c_0}}{2}\,y^2 \\ &+ \frac{1}{3}\,x^3 + \frac{\sqrt{2}}{2}\,x^2\,y + \frac{1}{2}\,x\,y^2 + \frac{\sqrt{2}}{12}\,y^3. \end{aligned}$$

En cherchant un peu, on voit que la condition $u(0,0) \neq 0$ est superflue sur cet exemple mais ce n'est pas le cas sur un exemple quelconque.

Les raisonnements tenus ci–dessus se généralisent à un système quelconque. La seule difficulté consiste à attribuer aux dérivées sous les escaliers de A des valeurs qui n'annulent pas les dénominateurs des formes normales. Ce n'est pas bien difficile puisque ces dénominateurs sont dans tous les cas des produits de puissances des dénominateurs des formes normales des inverses des initiaux et des séparants des éléments de A. Il n'y a donc dans tous les cas qu'un nombre fini d'inéquations ($\neq 0$) à considérer.

Proposition 32 *(construction des séries formelles solutions)*
Soit A une chaîne différentielle régulière de $\mathbb{Q}\{U\}$. Soient N l'ensemble des dérivées sous les escaliers et X l'ensemble des dérivées dominantes de A. Toute solution du système A, vu comme un système non différentiel de $\mathbb{Q}[X \cup N]$, qui n'annule pas les dénominateurs des formes normales des inverses des initiaux et des séparants de A, se prolonge en une unique série formelle solution du système différentiel A.

On obtient une formulation plus simple en demandant que la solution du système A n'annule pas les initiaux et les séparants des éléments de A. Les deux formulations sont justes mais pas tout–à–fait équivalentes.

La méthode exposée dans la section qui précède n'est pas très efficace. Voir [51] pour une méthode nettement meilleure, fondée sur un schéma de Newton, mais restreinte à une certaine classe de chaînes différentielles régulières.

6.1.5 Un résultat d'indécidabilité

On a montré dans la section précédente comment calculer des développements en série formelle à partir d'une chaîne différentielle régulière pour tout jeu de conditions initiales qui n'annule pas les dénominateurs des formes normales des inverses des initiaux et des séparants de A. Cette restriction est parfois superflue comme l'a montré l'exemple précédent. Parfois elle ne l'est pas, comme le montre le résultat suivant dû à Jan Denef et Leonard Lipshitz [28, Theorem 4.11]. Soit $p \in \mathbb{Z}[a_1, \ldots, a_m]$ un polynôme quelconque et u une indéterminée différentielle. L'expression suivante définit une équation aux dérivées partielles linéaires :

$$P\left(x_1 \frac{\partial}{\partial x_1}, \ldots, x_m \frac{\partial}{\partial x_m}\right) u \;=\; 0. \tag{6.1}$$

Prenons par exemple $P(a_1,\, a_2) = 3\,a_1^2 + 2\,a_2$. Alors

$$\begin{aligned} P\left(x_1 \frac{\partial}{\partial x_1},\, x_2 \frac{\partial}{\partial x_2}\right) u &= 3\,x_1 \frac{\partial}{\partial x_1}\left(x_1 \frac{\partial}{\partial x_1} u\right) + 2\left(x_2 \frac{\partial}{\partial x_2}\right) u \\ &= 3\,x_1 \frac{\partial}{\partial x_1}\left(x_1\, u_{x_1}\right) + 2\,x_2\, u_{x_2} \\ &= 3\,x_1^2\, u_{x_1 x_1} + 3\,x_1\, u_{x_1} + 2\,x_2\, u_{x_2}. \end{aligned}$$

Remarque 1. Si on substitue une série formelle

$$\bar{u} = \sum_{a_1,\ldots,a_m} c_{a_1,\ldots,a_m}\, x_1^{a_1} \cdots x_m^{a_m}$$

dont les coefficients $c_{a_1,\ldots,a_m}$ restent à déterminer dans l'équation (6.1), on obtient une expression :

$$P\left(x_1 \frac{\partial}{\partial x_1}, \ldots, x_m \frac{\partial}{\partial x_m}\right) \bar{u} \;=\; \sum_{a_1,\ldots,a_m} c_{a_1,\ldots,a_m}\, P(a_1,\ldots,a_m)\, x_1^{a_1} \cdots x_m^{a_m}.$$

Remarque 2.

$$\sum_{a_1,\ldots,a_m} x_1^{a_1} \cdots x_m^{a_m} = \left(\frac{1}{1-x_1}\right) \cdots \left(\frac{1}{1-x_m}\right).$$

En combinant les deux remarques, on conclut que, pour que l'équation aux dérivées partielles suivante

$$P\left(x_1 \frac{\partial}{\partial x_1}, \ldots, x_m \frac{\partial}{\partial x_m}\right) u \;=\; \left(\frac{1}{1-x_1}\right) \cdots \left(\frac{1}{1-x_m}\right)$$

ait une solution en série formelle $\bar{u}$ (qui si elle existe, est convergente), il est nécessaire et suffisant que les coefficients $c_{a_1,\ldots,a_m}$ satisfassent :

$$c_{a_1,\ldots,a_m} = \frac{1}{P(a_1,\ldots,a_m)}$$

et donc que $P(a_1,\ldots,a_m) \neq 0$ pour tous $(a_1,\ldots,a_m) \in \mathbb{N}^m$. D'après un célèbre théorème [68] de Yu Matijasevic, le problème de déterminer si un polynôme de $\mathbb{Z}[a_1,\ldots,a_m]$ admet une solution en nombres entiers est indécidable pour $m \geq 9$ (réponse négative au dixième problème de Hilbert).

Ce résultat s'étend immédiatement aux systèmes aux dérivées partielles polynomiaux dont les coefficients ne dépendent pas des variables indépendantes x_i, c'est–à–dire au cadre dans lequel nous nous sommes le plus souvent placés dans ce document. Il suffit de coder chaque variable dépendante x_i par une indéterminée différentielle z_i et de s'intéresser au système A suivant :

$$\begin{aligned}(1-z_1)\cdots(1-z_m)\, P\left(z_1 \frac{\partial}{\partial x_1}, \ldots, z_m \frac{\partial}{\partial x_m}\right) u &= 1 \\ \frac{\partial}{\partial x_j} z_i &= 1 \text{ si } i=j \text{ et } 0 \text{ sinon.}\end{aligned}$$

Le système A est une chaîne différentielle régulière pour tout classement tel que $u \gg (z_1,\ldots,z_m)$. Tous les monômes de la première équation de A admettent l'une des indéterminées différentielles z_i en facteur. Par conséquent, l'initial de la première équation de A

est de la forme $z_i^r\,(1-z_1)\cdots(1-z_m)$ où r est un entier positif et $1 \leq i \leq m$ est un indice. Les indéterminées z_j étant des dérivées sous les escaliers de A, cet initial est égal au dénominateur de la forme normale de son inverse.

Pour calculer un développement en série formelle en un point $(\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$ du système précédent, il faut procéder comme on l'a expliqué dans la section 6.1.4. Attention au fait que les valeurs attribuées aux dérivées $z_1, \ldots, z_m$ sont alors nécessairement les nombres $\alpha_1, \ldots, \alpha_m$. Le développement en série construit par Jan Denef et Leonard Lipshitz est centré en $(\alpha_1, \ldots, \alpha_m) = (0, \ldots, 0)$ or ces conditions initiales annulent l'initial de la première équation de A. Elles sont évitées par la proposition 32 et il n'y a donc (heureusement !) pas de contradiction entre tous ces résultats. Le théorème de Denef et Lipshitz conduit à la proposition suivante, qui explique pourquoi on a pris soin de ne jamais préciser l'ouvert $\mathscr{D}$ dans le chapitre 3.

Proposition 33 *(corollaire des résultats de Denef et Lipshitz)*
Le problème « étant donnés une chaîne différentielle régulière A et un point $(\alpha_1, \ldots, \alpha_m) \in \mathbb{Q}^m$, déterminer si l'idéal différentiel $[A] : H_A^\infty$ admet une solution en série formelle centrée en $(\alpha_1, \ldots, \alpha_m)$ » est indécidable.

6.2 Algorithmes de calcul de chaînes régulières

6.2.1 L'algorithme reg_characteristic

Il s'agit d'un algorithme non différentiel qu'on peut appliquer soit dans une situation purement algébrique soit sur des systèmes différentiels qui satisfont les hypothèses du lemme de Rosenfeld. Le principe de l'algorithme est fortement inspiré de l'algorithme *lexTriangular* dont le principe est dû à Daniel Lazard dans [59] et qui est bien clarifié dans la thèse [69] de Marc Moreno Maza. C'est Daniel Lazard qui m'a suggeré de l'appliquer dans le cadre différentiel lors d'un séminaire tenu à la fin de l'année 1996.

L'algorithme *reg_characteristic* prend en entrée un système $A = 0,\ S \neq 0$ d'un anneau de polynômes $K[t_1, \ldots, t_m, x_1, \ldots, x_n]$. Le système A est triangulaire. L'ensemble S contient les initiaux et séparants de A mais pas seulement. Il produit en sortie une famille finie de chaînes régulières $C_1, \ldots, C_k$ telles que

$$(A) : S^\infty = (C_1) : H_{C_1}^\infty \cap \cdots \cap (C_k) : H_{C_k}^\infty.$$

La famille produite peut être vide, ce qui signifie que $(A) : S^\infty$ est l'idéal unité. On peut montrer grâce au théorème des zéros et au lemme de Lazard que l'idéal $(A) : S^\infty$ est l'idéal des polynômes qui s'annulent sur toutes les solutions du système $A = 0,\ S \neq 0$. L'idée clef mise en œuvre est rappelée dans la proposition suivante.

Proposition 34 *Soit $\mathfrak{A}$ un idéal d'un anneau R et m un élément de R. L'idéal $\mathfrak{A} : m^\infty = \mathfrak{A}$ si et seulement si m est régulier dans $R/\mathfrak{A}$.*

L'algorithme applique l'idée en utilisant les méthodes décrites dans le chapitre 5. Il commence par transformer A en une chaîne régulière puis il teste la régularité des éléments de S modulo l'idéal défini par la chaîne. Tout élément prouvé régulier est supprimé. Il peut bien sûr advenir qu'un test de régularité échoue. Dans ce cas, une factorisation d'un élément de A est exhibée qui donne lieu à un scindage en deux branches. Il arrive fréquemment que l'un des deux facteurs exhibés soit un facteur d'un élément de S. Dans ce cas, on ne garde qu'une seule des deux branches. Il peut aussi arriver qu'un élément de S soit dans l'idéal défini par A. Dans ce cas, l'idéal $(A) : S^\infty$ est l'idéal unité.

L'algorithme *reg_ characteristic* est appliqué par l'algorithme *Rosenfeld–Gröbner* sur les « systèmes différentiels réguliers » produits à la fin du traitement purement différentiel, aussi bien dans le paquetage *diffalg* de MAPLE 9 que dans les bibliothèques *BLAD*. L'implantation en MAPLE 9 est due à François Lemaire. Cette application est suggerée dans [10] et publiée par François Lemaire et moi dans [12]. L'ancienne méthode suggerée dans [7, 9] et implantée dans les premières versions de *diffalg* consistait à calculer une base de Gröbner de l'idéal $(A) : S^\infty$ en utilisant le truc de Rabinovitch. En notant s le produit des éléments de S, il s'agissait d'introduire une nouvelle indéterminée x_{n+1} et d'appliquer :

$$(A) : S^\infty = (A,\, x_{n+1}\, s - 1) \cap R.$$

L'ancienne méthode présentait le gros inconvénient d'expliciter les inverses algébriques des éléments de S et de ne pas tenir compte de la structure triangulaire de A. L'algorithme *reg_ characteristic* exploite cette structure, teste l'existence des inverses mais ne les calcule pas.

Il existe une autre variante de *reg_ characteristic* qui s'applique dans le cadre d'algorithmes de changement d'ordre sur les indéterminées dans les chaînes régulières. On cherche à calculer une décomposition en chaînes régulières de l'idéal $(A) : S^\infty$ pour un ordre $\mathscr{O}$ sur les indéterminées. On suppose qu'on connaît à l'avance une chaîne régulière de l'idéal $(A) : S^\infty$ mais pour un ordre sur les indéterminées autre que $\mathscr{O}$. On se sert de la chaîne connue pour couper des branches lors des scindages. En particulier, lorsqu'on dispose d'une telle chaîne et qu'on sait à l'avance que l'idéal $(A) : S^\infty$ est premier, l'algorithme *reg_ characteristic* peut être implanté sans jamais produire le moindre scindage. En effet, chaque fois qu'une factorisation d'un élément de A est exhibée, on ne garde que le facteur qui appartient à l'idéal $(A) : S^\infty$. La chaîne connue est utilisée pour effectuer le test d'appartenance. Cette situation se produit en fin de traitement dans l'algorithme *PARDI*.

Le programme C suivant montre une implantation de *reg_ characteristic*. Dans cette implantation, on suppose que le système A initial est déjà une chaîne régulière. Le premier paramètre de la fonction est une intersection (un tableau) de chaînes régulières. La chaîne régulière en fin de tableau est la chaîne A. Le deuxième paramètre est la liste d'inéquations S. Le troisième paramètre est optionnel. S'il est présent, il est censé être une chaîne régulière (pour un autre ordre sur les indéterminées) de l'idéal $(A) : S^\infty$. La fonction supprime la chaîne A du tableau puis ajoute en fin de tableau les chaînes régulières calculées. Ce mécanisme peut sembler alambiqué. Il minimise les allocations dynamiques dans les deux cas de figure qui se produisent presque systématiquement : le cas où l'idéal $(A) : S^\infty$ est représenté par zéro ou une chaîne régulière.

```
/* File reg_characteristic.c */

#include "bad.h"

int main ()
{   bav_Iordering r;
    struct bad_intersectof_regchain tabC;
    bap_listof_polynom_mpz S;
    struct ba0_mark M;

    bad_restart (0, 0);
    ba0_record (&M);
    ba0_sscanf2 ("ordering (derivations = [], blocks = [x, y])",
                 "%ordering", &r);
    bav_R_push_ordering (r);
    bad_init_intersectof_regchain (&tabC);
    ba0_sscanf2 ("intersectof_regchain \
                         ([regchain ([y^2 - 1, (x + 2)*(x - y)], \
                          [autoreduced])], [autoreduced])",
                 "%intersectof_regchain", &tabC);
    ba0_sscanf2 ("[(y^2 + 2)*(y - 1)*(x + 3)]", "%l[%Az]", &S);
    bad_reg_characteristic_regchain (&tabC, S, (bad_regchain)0);
    ba0_printf ("%intersectof_regchain\n", &tabC);

    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

La chaîne régulière est $A = \{y^2-1, (x+2)(x-y)\}$. La liste S comporte le seul polynôme $(y^2+2)(y-1)(x+3)$ qui comporte en facteur un facteur d'un élément de A. La chaîne est simplifiée par *reg_ characteristic*. Aucun scindage n'est produit. Le troisième paramètre optionnel de la fonction n'est pas utilisé. Les chaînes régulières calculées sont complètement autoréduites.

```
$ ./reg_characteristic
intersectof_regchain ([regchain ([y + 1, x^2 + 3*x + 2], [autoreduced])], [
autoreduced])
```

6.2.2 L'algorithme regalise

Il s'agit là aussi d'un algorithme non différentiel qui peut être appliqué soit dans une situation purement algébrique soit sur des systèmes différentiels qui satisfont les hypothèses du lemme de Rosenfeld. Cet algorithme a été mis au point par François Lemaire, Marc

Moreno Maza et moi pour une version longue d'un article sur *PARDI* qui n'a jamais été publiée.

L'algorithme *regalise* s'applique dans le cadre d'algorithmes de changement d'ordre sur les indéterminées dans les chaînes régulières. Il prend en entrée un système $A = 0,\ S \neq 0$ d'un anneau de polynômes $R = K[t_1, \ldots, t_m, x_1, \ldots, x_n]$. Le système A est triangulaire pour un certain ordre sur les indéterminées $\mathscr{O}$. Les initiaux et les séparants de A appartiennent à S. L'idéal $(A) : S^\infty$ est supposé premier. L'algorithme prend aussi en entrée chaîne régulière C telle que $(A) : S^\infty = (C) : H_C^\infty$. En pratique, C est un chaîne régulière de $(A) : S^\infty$ mais pour un ordre sur les indéterminées autre que $\mathscr{O}$. L'algorithme produit en sortie une chaîne régulière $\bar{C}$ pour l'ordre $\mathscr{O}$ telle que $(A) : S^\infty = (\bar{C}) : H_{\bar{C}}^\infty$.

Voici le principe de l'algorithme. On sait que $(A){:}H_A^\infty \subset (C){:}H_C^\infty$. On utilise les méthodes décrites dans le chapitre 5 pour transformer A en une chaîne régulière, pour tester que les éléments de C appartiennent à $(A) : H_A^\infty$ et que les éléments de H_C sont réguliers modulo cet idéal. Si ces tests sont tous positifs, on a l'autre inclusion $(A) : H_A^\infty \supset (C) : H_C^\infty$ donc l'égalité entre les idéaux et A est la chaîne $\bar{C}$ recherchée. Bien sûr, un élément $p \in C$ peut ne pas appartenir à l'idéal $(A) : H_A^\infty$ mais c'est alors nécessairement un diviseur de zéro modulo cet idéal. Pourquoi ? Parce que $(C) : H_C^\infty$ s'obtient à partir de $(A) : H_A^\infty$ par un procédé de saturation : c'est donc un idéal premier associé à $(A){:}H_A^\infty$. Il suffit alors de tester la régularité de p modulo $(A){:}H_A^\infty$ par les méthodes du chapitre 5. Ce test échoue nécessairement et exhibe une factorisation d'un élément de A. On se sert de la chaîne connue C pour déterminer lequel des deux facteurs appartient à l'idéal $(C){:}H_C^\infty$ et ne garder que celui–là. De même, un élément de H_C peut être un diviseur de zéro modulo $(A) : H_A^\infty$. Dans ce cas aussi, une factorisation d'un élément de A est exhibée. On se sert de la chaîne connue C pour déterminer lequel des deux facteurs appartient à l'idéal $(C) : H_C^\infty$ et ne garder que celui–là.

Par rapport à l'une des variantes de *reg_ characteristic* décrites ci–dessus, l'algorithme *regalise* présente l'avantage de ne pas dépendre de S du tout. Il est donc plus efficace lorsque la liste S est volumineuse ou lorsqu'elle comporte des polynômes volumineux. Dans le cas différentiel par exemple, la chaîne connue est souvent formée de petits polynômes d'initiaux et de séparants égaux à 1. L'algorithme est dans ce cas très rapide.

6.2.3 L'algorithme PARDI (PODI)

Il s'agit initialement d'un algorithme pour les systèmes aux dérivées partielles : l'acronyme *PARDI* signifie « *Prime pARtial Differential Ideal* ». On peut évidemment l'appliquer aux systèmes différentiels ordinaires et aux systèmes non différentiels. Dans ces contextes, *PARDI* se simplifie et on appelle respectivement *PODI* (pour « *Prime Ordinary Differential Ideal* ») et *PALGIE* (pour « *Prime ALGebraic IdEal* ») les algorithmes simplifiés. Il a été mis au point par François Lemaire, Marc Moreno Maza et moi et publié dans [11]. Pour éviter d'avoir à décrire en cette fin de chapitre les mécanismes de complétion propres aux systèmes aux dérivées partielles, je choisis de décrire *PODI* plutôt que *PARDI*.

L'algorithme *PODI* prend en entrée une chaîne différentielle régulière C pour un classement $\mathscr{O}$ ainsi qu'un autre classement $\bar{\mathscr{O}}$. L'idéal différentiel $\mathfrak{A}$ défini par la chaîne C est supposé être premier. L'algorithme produit en sortie une chaîne différentielle régulière $\bar{C}$

pour le classement $\bar{\mathscr{O}}$ équivalente à C dans le sens où

$$\mathfrak{A} \underset{\text{def}}{=} [C] : H_C^\infty = [\bar{C}] : H_{\bar{C}}^\infty.$$

PARDI met en œuvre plusieurs idées.

Idée 1 : pas de scindage. Le calcul de la chaîne différentielle régulière $\bar{C}$ peut être effectué sans procéder à aucun scindage : la chaîne connue C permet de déterminer à chaque étape de l'algorithme si un polynôme donné appartient ou non à l'idéal $\mathfrak{A}$. Dans des contextes plus généraux, où on ne dispose pas d'une telle chaîne, les algorithmes sont amenés à procéder à des discussions de cas. C'est le cas de *Rosenfeld–Gröbner* par exemple. Cette première idée est due à François Ollivier et présentée dans sa thèse [78, page 97]. Je l'ai d'ailleurs implantée dès la première version du paquetage *diffalg* de MAPLE : il suffit de passer la chaîne différentielle régulière connue en paramètre supplémentaire à la fonction *Rosenfeld–Gröbner* de *diffalg*.

Idée 2 : un algorithme incrémental. Les algorithmes *PALGIE* et *PODI* peuvent être implantés en utilisant une liste P d'équations à traiter et une liste A d'équations déjà traitées. Initialement, P contient les éléments de C et A est vide. À la fin des calculs, P est vide et A contient la chaîne $\bar{C}$. À chaque itération, une équation de P est extraite de la liste, traitée pour fournir éventuellement une nouvelle équation de A. Le traitement peut amener à introduire de nouvelles équations dans P. L'appellation « équation déjà traitée » est un peu trompeuse pour les éléments de A dans la mesure où l'algorithme peut être amené à simplifier (voire à supprimer) les anciennes en utilisant les nouvelles. Dans le cas des systèmes aux dérivées partielles, une liste de paires critiques doit aussi être gérée comme en section 9.4, page 147.

Idée 3 : établir une relation professeur–étudiant. Tout polynôme réduit à zéro par la chaîne connue C doit au bout du compte l'être aussi par la chaîne en construction $\bar{C}$. S'il ne l'est pas, il doit être rajouté à la liste des équations à traiter de telle sorte que l'étudiant $\bar{C}$ « apprenne » les polynômes réduits à zéro par le professeur C.

Idée 4 : s'assurer du rang des polynômes. Soit $p = a_d\, v^d + \cdots + a_1\, v + a_0$ un polynôme différentiel de dérivée dominante v. Notons $R_{<v}$ l'anneau des polynômes différentiels de dérivée dominante strictement inférieure à v et $\mathfrak{A}_{<v}$ l'idéal $\mathfrak{A} \cap R_{<v}$. On peut s'assurer du degré de p en tant qu'élément de $(\mathfrak{A} \cap R_{<v})[v]$ en déterminant si $a_d \in \mathfrak{A}_{<v}$. Il suffit pour cela de tester si a_d est réduit à zéro par la chaîne connue. Si $a_d \in \mathfrak{A}_{<v}$ on simplifie le polynôme en le remplaçant par $p - a_d\, v^d$. Si le coefficient $a_d \notin \mathfrak{A}_{<v}$, on peut encore tester si son séparant $s = d\, a_d\, v^{d-1} + \cdots + a_1$ appartient à cet idéal. Si c'est le cas, on simplifie le polynôme en le remplaçant par $d\, p - v\, s$.

Idée 5 : les racines communes de deux polynômes sont les racines de leur pgcd. C'est un principe bien connu pour les polynômes en une indéterminée à coefficients dans un

corps mais généralement très difficile à mettre en œuvre dans les algorithmes généraux de décomposition en chaînes régulières.

Le problème d'un tel calcul se pose lorsqu'une équation à traiter p_1 extraite de P a même dérivée dominante v qu'une équation déjà traitée $p_2 \in A$. Alors, en notant $R_{<v}$ l'anneau des polynômes différentiels de dérivée dominante strictement inférieure à v et $\mathfrak{A}_{<v}$ l'idéal $\mathfrak{A} \cap R_{<v}$, ces deux polynômes ont un pgcd non trivial g dans $(R_{<v}/\mathfrak{A}_{<v})[v]$ et il suffit de remplacer p_2 par g dans A.

Comment calculer ce pgcd ? On remarque d'abord que la chaîne régulière connue C permet de reconnaître zéro dans $R_{<v}/\mathfrak{A}_{<v}$. Il est donc possible d'appliquer au moins l'algorithme d'Euclide naïf. Maintenant, on voudrait pouvoir appliquer un algorithme un peu plus efficace. À défaut de méthodes modulaires qui commencent seulement à être mises au point [14, 24] on voudrait utiliser des implantations sophistiquées [36, 65] de l'algorithme de calcul de la suite des sous–résultants. La seule difficulté consiste à réaliser dans $R_{<v}/\mathfrak{A}_{<v}$ les divisions exactes prédites par la théorie des sous–résultants et mises en œuvre avec subtilité par ces méthodes. Cette difficulté se lève de la façon la plus élémentaire qui soit : il suffit de mener les calculs comme dans $R_{<v}[v]$ et de s'assurer uniquement[3] que les coefficients dominants des sous–résultants calculés sont non nuls modulo $\mathfrak{A}_{<v}$. Si l'un des sous–résultants a un coefficient dominant nul modulo $\mathfrak{A}_{<v}$, il suffit de le simplifier et de réamorcer le calcul de la suite à partir de ce sous–résultant et de celui qui le précède.

Une application en *BLAD* de *PODI* est décrite dans le chapitre 4.

Équations d'Euler pour un fluide incompressible

Dans le cadre des systèmes aux dérivées partielles, des implantations en *BLAD* et en MAPLE de l'algorithme *PARDI* ont permis de mener à terme en 2001 des éliminations qui n'avaient jamais pu l'être. Le plus gros exemple traité concerne les équations d'Euler pour un fluide incompressible :

$$\vec{v} + (\vec{v} \cdot \vec{\nabla})\, \vec{v} + \vec{\nabla} p = \vec{0}, \quad \vec{\nabla} \vec{v} = 0.$$

En deux dimensions, en notant $\vec{v} = (v^1,\, v^2)$ et $\vec{\nabla} = (\partial/\partial x,\, \partial/\partial y)$, on obtient trois équations différentielles polynomiales :

$$v^1_t + v^1 v^1_x + v^2 v^1_y + p_x = 0, \quad v^2_t + v^1 v^2_x + v^2 v^2_y + p_y = 0, \quad v^1_x + v^2_y = 0.$$

Il y a trois indéterminées différentielles v^1, v^2 (composantes de la vitesse) et la pression p. Elles dépendent de trois variables indépendantes x, y (variables d'espace) et le temps t. Pour un certain classement compatible avec l'ordre total, l'algorithme *Rosenfeld–Gröbner* produit l'unique chaîne différentielle régulière C suivante :

$$p_{xx} + 2\, v^2_x\, v^1_y + 2\, (v^2_y)^2 + p_{yy}, \quad v^1_t + v^2\, v^1_y + p_x - v^2_y\, v^1, \quad v^1_x + v^2_y, \quad v^2_t + v^1\, v^2_x + v^2\, v^2_y + p_y.$$

[3] On sait par exemple qu'il existe deux représentations courantes des éléments de $\mathbb{Z}/n\mathbb{Z}$: soit les entiers de 0 à $n-1$ soit les entiers de $-(n-1)/2$ à $(n-1)/2$. Le choix de la représentation dépend des besoins de l'algorithme qui l'utilise. Pour l'algorithme *PARDI*, nous avons choisi une représentation des éléments de $R_{<v}/\mathfrak{A}_{<v}$ qui simplifie les divisions exactes.

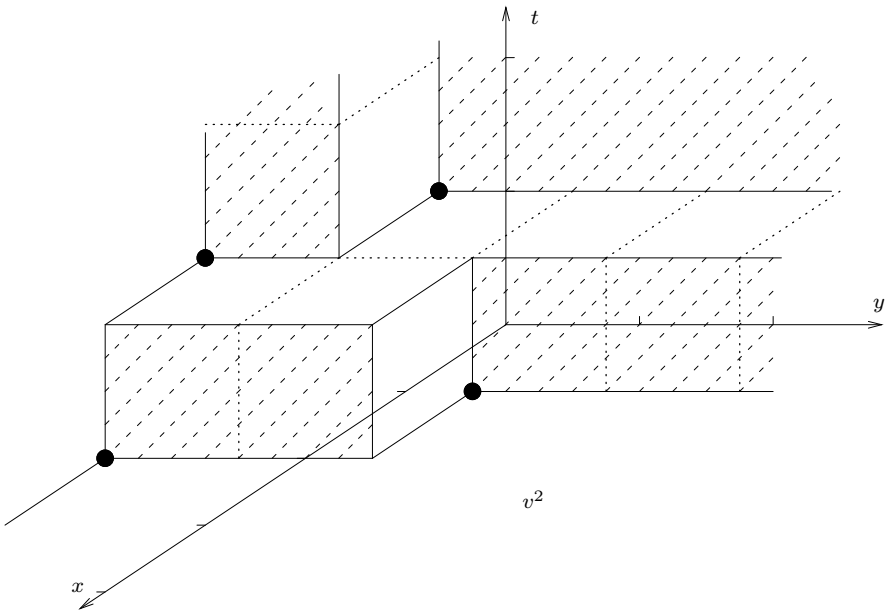
Cette chaîne définit un idéal différentiel premier $\mathfrak{p}$ (il s'agit d'un système orthonomique et cohérent). Voir en section 6.1.1. Pour le classement d'élimination par blocs suivant

$$(p,\, v^1) \gg \mathrm{degrevlex}(v^2)$$

avec $t > x > y$, les implantations de *PARDI* ont permis de calculer une chaîne différentielle régulière $\overline{C}$ de $\mathfrak{p}$. Cette chaîne est trop grosse pour être écrite ici (le fichier fait 600 Ko). Elle comporte sept équations dépendant de cinquante dérivées différentes des trois indéterminées différentielles. Voici le rang de la chaîne

$$\{p_x,\, p_y,\, v^1,\, v^2_{xxxxt},\, v^2_{xxxtt},\, v^2_{xxytt},\, v^2_{xxxyyt}\}.$$

L'escalier correspondant à l'indéterminée différentielle v^2 est le suivant :



Ce calcul correspond davantage à un « challenge calculatoire » qu'à une véritable application. Il avait été mené partiellement par Jean–François Pommaret dans [82] et par moi dans ma thèse [7]. Un problème calculatoire toujours ouvert consiste à éliminer les deux composantes de la vitesse.

Chapitre 7

Fondements algébriques des systèmes triangulaires

Ce chapitre expose les preuves des théorèmes qui fondent la théorie des systèmes triangulaires. On commence par deux théorèmes non différentiels : le théorème de Macaulay et le lemme de Lazard et on continue par deux théorèmes différentiels : le lemme de Rosenfeld et le « lifting » du lemme de Lazard dans le contexte différentiel.

C'est à l'occasion de la controverse au sujet de la preuve du « lemme de Lazard » [9, Lemma 2] que Sally Morrison a mis en évidence [72] l'importance du théorème de Macaulay pour les systèmes triangulaires. La preuve de Sally Morrison est publiée dans [73]. C'est historiquement la première preuve complète du lemme de Lazard. La controverse ne concernait que les idéaux de la forme $(A) : S_A^\infty$ qui apparaissent naturellement dans le cas différentiel mais le problème soulevé concernait tout autant les idéaux de la forme $(A) : I_A^\infty$ dans le cas algébrique. La preuve de [2, Theorem 5.1] souffre de ce problème et n'a été corrigée que dans la thèse de Philippe Aubry [1]. Ce dernier n'utilise pas explicitement le théorème de Macaulay mais les propriétés des « suites régulières » dans les anneaux « de Cohen–Macaulay » qui sont les anneaux dans lesquels le théorème de Macaulay s'applique !

Quel est le problème ? Notons $t_1, \ldots, t_m$ les indéterminées dont un système triangulaire A dépend et qui ne sont pas des indéterminées principales de A. Les preuves de [2, Theorem 5.1] et [9, Lemma 2] utilisent implicitement le fait suivant : tout polynôme non nul qui ne dépend que de $t_1, \ldots, t_m$ est régulier modulo l'idéal défini par A. Cette hypothèse est vraie mais ne saurait être considérée comme évidente.

Dans le cas algébrique, mentionnons le résultat d'équidimensionnalité de Shang–Ching Chou et Xiao–Shan Gao [21] malheureusement insuffisant (il ne résout pas la problème des « premiers immergés »). Dans le cas différentiel, mentionnons une jolie preuve élémentaire de François Ollivier [79], reprise dans [10], qui évite l'emploi du théorème de Macaulay mais ne s'applique qu'aux idéaux de la forme $(A) : S_A^\infty$. La formulation ci–dessous, qui traite en même temps les idéaux de la forme $(A) : I_A^\infty$ et $(A) : S_A^\infty$ est de moi. Je l'ai publiée avec François Lemaire et Marc Moreno Maza dans [13].

Pour la partie différentielle, disons que l'appellation « lemme de Rosenfeld » laisse injustement dans l'ombre un théorème d'Abraham Seidenberg [95, Theorem VI] qui contient déjà

sous une forme fort technique les idées clarifiées et un peu généralisées du lemme d'Azriel Rosenfeld [87]. Dans son livre Ellis Robert Kolchin généralise le lemme de Rosenfeld mais le « découpe » en deux parties [56, chapter III, section 8, page 136 ; remarques page 167], ce qui a pour effet de masquer le fait que les hypothèses de Rosenfeld sont algorithmiques. Le « lifting » du lemme de Lazard dans le contexte différentiel comporte deux parties. La première (la radicalité des idéaux différentiels de la forme $[A] : H_A^\infty$) est publiée dans [9] par Daniel Lazard, François Ollivier, Michel Petitot et moi. La seconde a semble–t–il été trouvée indépendemment par Évelyne Hubert [50] et les auteurs cités ci–dessus dans [10].

7.1 L'équidimensionnalité

Dans cette section, on démontre le théorème 3 qui constitue une version généralisée du point clef 1, page 73, qu'on a admis au chapitre 5. On considère un système triangulaire $A = \{p_1, \ldots, p_n\}$ de l'anneau $R = K[x_1, \ldots, x_n, t_1, \ldots, t_m]$. Chaque polynôme p_i admet x_i pour indéterminée principale. On note $\mathfrak{A}$ l'idéal $(A) : h^\infty$ de R où h désigne soit le produit des initiaux des éléments de A soit le produit des séparants des éléments de A. L'idéal $\mathfrak{A}$ peut fort bien être trivial (prendre $A = \{x_1,\, x_1\, x_2\}$). Nous supposons que ce n'est pas le cas. On note $R_0 = K(t_1,\, \ldots,\, t_m)[x_1,\, \ldots,\, x_n]$ l'anneau obtenu en « faisant passer les indéterminées non principales dans le corps des coefficients » et $\mathfrak{A}_0$ l'idéal $(A) : h^\infty$ de R_0. On M la famille multiplicative formée des éléments non nuls de $K[t_1, \ldots, t_m]$. L'anneau R_0 est égal à l'anneau localisé $M^{-1}\, R$. Tout élément de $R_0/\mathfrak{A}_0$, qui est isomorphe à $(M/\mathfrak{A})^{-1}(R/\mathfrak{A})$, est de la forme a/b avec $a \in R/\mathfrak{A}$ et $b \in M/\mathfrak{A}$.

Théorème 3 *(reformulation du point clef 1)*
Décider de la nullité ou de la régularité dans $R/\mathfrak{A}$ est strictement équivalent à décider de la nullité ou de la régularité dans $R_0/\mathfrak{A}_0$.
Plus précisément, un élément a de $R/\mathfrak{A}$ est nul (resp. régulier) si et seulement si tout élément a/b de $R_0/\mathfrak{A}_0$ est nul (resp. régulier).

Proposition 35 *Pour démontrer le théorème 3, il suffit de démontrer que tout élément de $M/\mathfrak{A}$ est régulier.*

Preuve C'est une proposition très classique. Si tout élément de $M/\mathfrak{A}$ est régulier alors $R_0/\mathfrak{A}_0$ est un sous–anneau de l'anneau total des fractions de $R/\mathfrak{A}$ [108, chapter IV, paragraph 9]. La proposition découle alors de [108, chapter I, paragraph 19, Corollary 1]. □

On rappelle le célèbre théorème de Lasker–Nœther [108, chapter IV, Theorems 4 et 6] ou [101, sections 15.4 et 15.5].

Théorème 4 *(théorème de Lasker–Nœther)*
Dans un anneau nœthérien, tout idéal est une intersection finie d'idéaux primaires. Toute représentation d'un idéal comme intersection d'idéaux primaires peut être minimalisée en supprimant d'une part les idéaux primaires redondants et en regroupant d'autre part les

idéaux primaires dont l'intersection est elle–même un idéal primaire. La décomposition minimale ainsi obtenue d'un idéal n'est pas définie de façon unique. Par contre, le nombre de composantes et les radicaux des composantes primaires (qu'on appelle les idéaux premiers « associés » à l'idéal) le sont.

L'anneau R est nœthérien. En combinant le théorème de Lasker–Nœther et la proposition 35, on obtient la proposition suivante.

Proposition 36 *Pour démontrer le théorème 3, il suffit de démontrer qu'aucun idéal premier associé à $\mathfrak{A}$ ne rencontre M.*

Preuve D'après [108, chapter IV, paragraph 6, Corollary 3], si M ne rencontre aucun des idéaux premiers associés à $\mathfrak{A}$ alors tout élément de $M/\mathfrak{A}$ est régulier. Le théorème 3 découle alors de la proposition 35. □

Définition 12 *La dimension* $\dim \mathfrak{p}$ *d'un idéal premier $\mathfrak{p}$ d'un anneau de polynômes R à coefficients dans un corps K est le degré de transcendance du corps des fractions de $R/\mathfrak{p}$ sur K. La dimension* $\dim \mathfrak{B}$ *d'un idéal $\mathfrak{B}$ quelconque de R est le maximum des dimensions des idéaux premiers associés à $\mathfrak{B}$.*

Le reste de cette section est entièrement consacré à la démonstration du théorème suivant dont le théorème 3 est un corollaire. Cette reformulation du théorème 3 est souvent utile pour rédiger des preuves.

Théorème 5 *Les idéaux premiers associés à $\mathfrak{A}$ sont de dimension m et ne rencontrent pas M.*

Pour pouvoir appliquer le théorème de Macaulay (ce qu'on cherche à faire), il faut se débarrasser de la saturation par h. On utilise pour cela le truc de Rabinowitsch [101, section 16.5]. On introduit une nouvelle indéterminée x_{n+1} et un nouveau polynôme $p_{n+1} = h\,x_{n+1} - 1$. On note $A' = A \cup \{p_{n+1}\}$ le système triangulaire de l'anneau $R' = R[x_{n+1}]$ obtenu en adjoignant p_{n+1} à A. On note $\mathfrak{A}'$ l'idéal (A') de R'. Considérons les deux morphismes d'anneaux canoniques suivants :

$$R \xrightarrow{\ \phi\ } h^{-1}\,R \simeq R'/(p_{n+1}) \xleftarrow{\ \pi\ } R'.$$

L'isomorphisme $h^{-1}R \simeq R'/(p_{n+1})$ est classique [37, Exercise 2.2, page 79] : chaque élément de R est mis en correspondance avec lui–même, x_{n+1} correspond avec h^{-1}. Le morphisme ϕ est la localisation en h. Le morphisme π est le passage au quotient par l'idéal (p_{n+1}). Si $\mathfrak{B}$ est un idéal de R, on note $h^{-1}\mathfrak{B}$ ou $(\phi\mathfrak{B})$ l'idéal de $h^{-1}R$ engendré par $\phi\mathfrak{B}$. Si $\mathfrak{B}'$ est un idéal de R' alors $\pi\mathfrak{B}'$ est un idéal de $\pi R' = R'/(p_{n+1})$.

Lemme 3 *L'idéal $\mathfrak{A}'$ n'est pas trivial. Si l'intersection $\mathfrak{q}'_1 \cap \cdots \cap \mathfrak{q}'_r$ est une décomposition primaire minimale de l'idéal $\mathfrak{A}'$ alors $\phi^{-1}(\pi\mathfrak{q}'_1) \cap \cdots \cap \phi^{-1}(\pi\mathfrak{q}'_r)$ est une décomposition primaire minimale de l'idéal $\mathfrak{A}$.*

Preuve On utilise la notation des « extensions » et des « contractions » définie dans [108, chapter IV, paragraph 8] vis–à–vis du morphisme ϕ de telle sorte que $(\phi\mathfrak{A}) = \mathfrak{A}^e$. L'idéal $\pi\mathfrak{A}'$ est égal à l'idéal $(\phi\mathfrak{A})$ puisque ces deux idéaux admettent une même famille génératrice : A. D'après [108, chapter IV, Theorem 15 (a)] on a $\mathfrak{A} = \mathfrak{A}^{ec}$ puisque $\mathfrak{A} = \mathfrak{A} : h^\infty$. Comme $\mathfrak{A}$ n'est pas trivial, les idéaux $\mathfrak{A}^e$ et $\mathfrak{A}'$ ne le sont donc pas non plus.

Considérons maintenant une décomposition primaire minimale $\mathfrak{q}'_1 \cap \cdots \cap \mathfrak{q}'_r$ de l'idéal $\mathfrak{A}'$. D'après [108, chapter IV, paragraph 5, Remark concerning passage to a residue class ring], $\pi\mathfrak{q}'_1 \cap \cdots \cap \pi\mathfrak{q}'_r$ est une décomposition primaire minimale de $\pi\mathfrak{A}' = \mathfrak{A}^e$. Comme $\mathfrak{A} = \mathfrak{A}^{ec}$, d'après [108, chapter IV, Theorem 15 (b) et les commentaires qui précèdent], les idéaux premiers associés à $\mathfrak{A}$ ne rencontrent pas M. D'après [108, chapter IV, Theorem 17], l'intersection $\phi^{-1}(\pi\mathfrak{q}'_1) \cap \cdots \cap \phi^{-1}(\pi\mathfrak{q}'_r)$ est une décomposition primaire minimale de $\mathfrak{A}$. □

Proposition 37 *Pour démontrer le théorème 5, il suffit de montrer que les idéaux premiers associés à $\mathfrak{A}'$ sont de dimension m et ne rencontrent pas M.*

Preuve Soit $\mathfrak{p}'$ un idéal premier associé à $\mathfrak{A}'$ et $\mathfrak{p} = \phi^{-1}(\pi\mathfrak{p}')$ l'idéal premier associé à $\mathfrak{A}$ qui lui correspond d'après le lemme 3. Soit a un élément du sous–anneau R de R'. Alors $a \in \mathfrak{p}'$ si et seulement si $a/1 \in \pi\mathfrak{p}'$ et $a/1 \in \pi\mathfrak{p}'$ si et seulement si $a \in \mathfrak{p}$. Par conséquent, si $\mathfrak{p}'$ ne rencontre pas M alors $\mathfrak{p}$ non plus et $\dim \mathfrak{p} \geq m$. Si de plus $\dim \mathfrak{p}' = m$ alors $x_1, \ldots, x_n$ dépendent forcément algébriquement de $t_1, \ldots, t_m$ modulo $\mathfrak{p}'$. Donc $x_1, \ldots, x_n$ dépendent algébriquement de $t_1, \ldots, t_m$ modulo $\mathfrak{p}$ et $\dim \mathfrak{p} \leq m$. En combinant les deux inégalités, on conclut que $\dim \mathfrak{p} = m$. □

On distingue deux types d'idéaux premiers associés à un idéal $\mathfrak{A}$: les premiers « isolés » ou « minimaux » d'une part et les premiers « immergés » d'autre part. Les premiers minimaux sont appelés ainsi parce qu'ils sont minimaux pour la relation d'inclusion dans la famille des idéaux premiers qui contiennent $\mathfrak{A}$. Un premier immergé est donc un premier associé qui contient un premier minimal (dans le contexte des idéaux de polynômes, la variété algébrique d'un premier immergé est incluse (immergée) dans celle du premier minimal qu'il contient).

On comprend qu'on arrive assez bien à cerner les premiers minimaux (dans le contexte des idéaux de polynômes en tous cas) : comme ils correspondent aux composantes irréductibles de la variété algébrique de l'idéal $\mathfrak{A}$ [108, chapter VII, paragraph 3, Corollary 3 to Hilbert's Nullstellensatz], on peut les « capter » par des raisonnements sur les solutions de l'idéal. Par contre les idéaux premiers immergés sont beaucoup plus difficiles à cerner : ils n'ont pas de sens géométrique aussi simple — à ma connaissance. Voir toutefois l'interprétation géométrique exposée dans [37, section 3.8].

Pour le problème qui nous concerne, le lemme 4 règle facilement le cas des idéaux premiers minimaux sur $\mathfrak{A}$. Pour le problème posé par les premiers immergés, on utilise un théorème « dur » : le théorème de Macaulay ... qui prouve qu'il n'y en a pas !

On rappelle le théorème de l'idéal principal [108, chapter VII, Theorem 22].

Théorème 6 *(théorème de l'idéal principal)*
Si un idéal $\mathfrak{A}$ d'un anneau $R = K[x_1, \ldots, x_n]$ admet une famille génératrice formée de h éléments ($1 \leq h \leq n$) alors $\dim \mathfrak{A} \geq n - h$.

Revenons à l'étude de l'idéal $\mathfrak{A}'$ de R'.

Lemme 4 *La dimension de l'idéal $\mathfrak{A}'$ est m. De plus, aucun idéal premier associé à $\mathfrak{A}'$ de dimension m (et donc minimal sur $\mathfrak{A}'$) ne rencontre M.*

Preuve Ce lemme est prouvé dans [21] dans le cas où h est le produit des initiaux de A. Considérons un idéal premier $\mathfrak{p}'$ associé à $\mathfrak{A}'$.

Considérons d'abord le cas où h est le produit des initiaux des éléments de A. Alors aucun de ces initiaux n'appartient à $\mathfrak{p}'$ (sinon $\mathfrak{p}'$, qui contient $h\,x_{n+1}-1$, contiendrait 1). Donc les quantités $x_1,\ldots,x_{n+1}$ dépendent algébriquement de $t_1,\ldots,t_m$ sur K dans $R'/\mathfrak{p}'$ (les polynômes de A' ne peuvent pas dégénérer du tout modulo $\mathfrak{p}'$).

Considérons maintenant le cas où h est le produit des séparants des éléments de A. Soit $p_\ell = a_d\,x_\ell^d + \cdots + a_1\,x_\ell + a_0$ un quelconque élément de A'. Comme son séparant $s_\ell = d\,a_d\,x_\ell^{d-1} + \cdots + a_1$ n'appartient pas à $\mathfrak{p}'$, au moins un des coefficients $a_d,\ldots,a_1$ n'appartient pas à cet idéal. Donc les quantités $x_1,\ldots,x_{n+1}$ dépendent algébriquement de $t_1,\ldots,t_m$ sur K dans $R'/\mathfrak{p}'$ (les polynômes de A' ne peuvent pas dégénérer complètement modulo $\mathfrak{p}'$).

Dans les deux cas, $x_1,\ldots,x_{n+1}$ dépendent algébriquement de $t_1,\ldots,t_m$ sur K dans $R'/\mathfrak{p}'$. On en conclut *primo* que $\dim\mathfrak{p}' \le m$ et donc que $\dim\mathfrak{A}' \le m$ et *secundo* que si $\dim\mathfrak{p}' = m$ alors $\mathfrak{p}' \cap M = \emptyset$.

L'idéal $\mathfrak{A}'$ admet une famille génératrice formée de $n+1$ éléments dans un anneau de polynômes en $n+m+1$ indéterminées. D'après le théorème de l'idéal principal $\dim\mathfrak{A}' \ge m$. En combinant les deux inégalités, on conclut que $\dim\mathfrak{A}' = m$. □

La proposition suivante, combinée à la proposition 37, conclut la démonstration du théorème 3.

Proposition 38 *Les idéaux premiers associés à $\mathfrak{A}'$ sont de dimension m et ne rencontrent pas M.*

Preuve L'idéal $\mathfrak{A}'$ admet une famille génératrice formée de $n+1$ éléments dans un anneau de polynômes en $n+m+1$ indéterminées. D'après le lemme 4, sa dimension est m. Le théorème de Macaulay peut donc s'appliquer : tous ses idéaux premiers associés sont de dimension m. D'après le lemme 4, aucun de ses idéaux premiers associés ne rencontre M. □

J'ai retranscrit la preuve du théorème de Macaulay telle qu'elle est donnée dans [108, chapter VII, Theorem 26], augmentée de quelques commentaires. C'est un beau morceau d'algèbre commutative.

Théorème 7 *(théorème de Macaulay)*

Si un idéal $\mathfrak{A}$ d'un anneau $R = K[x_1,\ldots,x_n]$ admet une famille génératrice formée de h éléments ($1 \le h \le n$) et si $\dim\mathfrak{A} = n-h$ alors tous les premiers associés à $\mathfrak{A}$ sont de dimension $n-h$.

Preuve On considère une famille génératrice $\{p_1, \ldots, p_h\}$ de $\mathfrak{A}$. La preuve est une récurrence sur h.

La base $h = 0$ est triviale.

Soit $\mathfrak{p}$ un premier associé à $\mathfrak{A}$, de dimension d.

Comme $\dim \mathfrak{A} = n - h$ on a nécessairement $d \leq n - h$.

Renumérotons les indéterminées de telle sorte que $\mathfrak{p} \cap K[x_1, \ldots, x_d] = (0)$. Notons M la famille multiplicative formée par les éléments non nuls de $K[x_1, \ldots, x_d]$ et plaçons–nous dans l'anneau de polynômes $M^{-1}R = K(x_1, \ldots, x_d)[x_{d+1}, \ldots, x_n]$.

Les premiers associés à $M^{-1}\mathfrak{A}$ sont de la forme $M^{-1}\mathfrak{p}'$ où $\mathfrak{p}'$ est associé à $\mathfrak{A}$ et ne rencontre pas M. La dimension de ces idéaux–là chute exactement de d et donc $\dim M^{-1}\mathfrak{A} \leq n-d-h$. Par ailleurs $\dim M^{-1}\mathfrak{A} \geq n - d - h$ puisque cet idéal est engendré par h éléments dans un anneau de polynômes en $n-d$ indéterminées (théorème de l'idéal principal). Par conséquent $\dim M^{-1}\mathfrak{A} = n - d - h$.

On voit également que l'idéal $M^{-1}\mathfrak{p}$ est associé à $M^{-1}\mathfrak{A}$ et a pour dimension zéro.

Il suffit donc, en revenant aux notations du théorème, de montrer que si $n - h > 0$ et $\dim \mathfrak{p} = 0$ alors $\mathfrak{p}$ n'est pas associé à $\mathfrak{A}$.

On considère l'idéal $\mathfrak{B} = (p_1, \ldots, p_{h-1})$.

J'affirme que $\dim \mathfrak{B} = n-h+1$. Preuve. D'après le théorème de l'idéal principal, $\dim \mathfrak{B} \geq n - h + 1$. Supposons $\dim \mathfrak{B} > n - h + 1$. Alors $\mathfrak{B}$ aurait un premier associé $\mathfrak{p}'$ tel que $\dim \mathfrak{p}' > n-h+1$ et $(\mathfrak{p}', p_h)$ aurait un premier associé $\mathfrak{p}''$ tel que $\dim \mathfrak{p}'' > n-h$ (théorème de l'idéal principal à nouveau). C'est impossible puisque $\mathfrak{A} \subset \mathfrak{p}''$ et $\dim \mathfrak{A} = n-h$. L'affirmation est donc prouvée.

L'hypothèse de récurrence s'applique une première fois : tous les premiers associés à $\mathfrak{B}$ ont pour dimension $n - h + 1$. Notons–les $\mathfrak{p}_1, \ldots, \mathfrak{p}_r$ et rajoutons à cette liste les premiers $\mathfrak{p}_{r+1}, \ldots, \mathfrak{p}_{r'}$ associés à $\mathfrak{A}$ qui sont exactement de dimension $n-h$. Aucun idéal de cette liste n'est maximal (on a supposé $n - h > 0$).

Admettons qu'on puisse construire un polynôme $y_t = x_t + \phi(x_1, \ldots, x_{t-1})$ (pour un certain $1 \leq t \leq n$) tel que y_t soit transcendant sur K modulo $\mathfrak{p}_i$ (pour $1 \leq i \leq r'$).

Comme $\dim \mathfrak{p} = 0$ on voit que y_t est algébrique sur K modulo $\mathfrak{p}$ et on note f le polynôme minimal de y_t modulo $\mathfrak{p}$. On voit que $f \in \mathfrak{p}$ et que $f \notin \mathfrak{p}_i$ puisque y_t est transcendant sur K modulo $\mathfrak{p}_i$ (pour $1 \leq i \leq r'$).

On considère un élément $a \in R$ tel que $a\,\mathfrak{p} \subset \mathfrak{A}$. J'affirme qu'il suffit de montrer que $a \in \mathfrak{A}$ pour conclure la preuve du théorème de Macaulay. Preuve. Notons $\mathfrak{q}_j$ les composantes d'une décomposition primaire minimale de $\mathfrak{A}$ (pour $1 \leq j \leq s$). Supposons que $\mathfrak{p}$ soit associé à $\mathfrak{A}$ et à $\mathfrak{q}_1$. Alors il existe un exposant ρ tel que $\mathfrak{p}^\rho \subset \mathfrak{q}_1$ (c'est une propriété des idéaux primaires dans les anneaux nœthériens). On peut alors prendre $a \in \mathfrak{p}^{\rho-1} \cap \mathfrak{q}_2 \cap \cdots \cap \mathfrak{q}_s$ et donc $a \notin \mathfrak{A}$ tel que $a\,\mathfrak{p} \subset \mathfrak{A}$. L'affirmation est prouvée.

On a $a\,f \in \mathfrak{A} = (\mathfrak{B}, p_h)$ donc il existe $b \in R$ tel que $a\,f + b\,p_h \in \mathfrak{B}$ et $b\,p_h \in (\mathfrak{B}, f)$.

J'affirme que p_h n'appartient à aucun des premiers associés à $(\mathfrak{B}, f)$. Preuve. Dans l'anneau de polynômes $R/(f) = K(\overline{y}_t)[x_1, \ldots, x_{t-1}, x_{t+1}, \ldots, x_n]$, considérons l'idéal $(\mathfrak{B}, f)/(f)$. Il est engendré par $h-1$ éléments. Sa dimension est $n-h$ car f a été choisi en dehors des premiers associés à $\mathfrak{B}$ dont on a prouvé qu'ils ont tous pour dimension $n - h + 1$. Comme

$R/(f)$ est un anneau de polynômes en $n-1$ indéterminées au–dessus d'un corps, l'hypothèse de récurrence s'applique une seconde fois : tous les premiers associés à $(\mathfrak{B},\, f)/(f)$ ont pour dimension $n-h$.

Les premiers associés à $(\mathfrak{B},\, f)$ sont donc eux–aussi de dimension $n-h$.

Si un premier associé à $(\mathfrak{B},\, f)$ contenait p_h, il serait aussi associé à $\mathfrak{A}$ (puisque $\dim \mathfrak{A} = n-h$). C'est impossible puisqu'il contient f et qu'on a choisi f en dehors des premiers associés à $\mathfrak{A}$ de dimension $n-h$.

L'affirmation est donc prouvée.

Maintenant $b\,p_h \in (\mathfrak{B},\, f)$. Comme p_h n'appartient à aucun des premiers associés à cet idéal on a $b \in (\mathfrak{B},\, f)$ (propriété des idéaux primaires). Il existe donc $c \in R$ tel que $b - c\,f \in \mathfrak{B}$. Cette relation, combinée avec $a\,f - b\,p_h \in \mathfrak{B}$, montre que $(a - c\,p_h)\,f \in \mathfrak{B}$. Comme f a été choisi en dehors des premiers associés à $\mathfrak{B}$ on conclut que $a - c\,p_h \in \mathfrak{B}$ et donc que $a \in (\mathfrak{B},\, p_h) = \mathfrak{A}$.

La preuve du théorème de Macaulay est terminée.

Il reste simplement à montrer la construction du polynôme y_t. On se donne une famille d'idéaux premiers non maximaux $\mathfrak{p}_1, \ldots, \mathfrak{p}_{r'}$ de R. On montre qu'il existe un indice $1 \leq t \leq n$ et un polynôme $\phi(x_1, \ldots, x_{t-1})$ tel que $y_t = x_t + \phi(x_1, \ldots, x_{t-1})$ soit transcendant sur K modulo chaque $\mathfrak{p}_j$.

On forme un tableau à double entrée. Les indices de ligne sont les x_i. Les indices de colonne sont les $\mathfrak{p}_j$. Aux coordonnées $(x_i,\, \mathfrak{p}_j)$, on place un T ou un A suivant que x_i est transcendant ou algébrique sur K modulo $\mathfrak{p}_j$. On procède à des permutations de colonne (on renumérote les $\mathfrak{p}_j$). On commence par placer le plus à gauche possible tous les $\mathfrak{p}_j$ tels que x_1 est transcendant sur k modulo $\mathfrak{p}_j$. La première ligne ressemble alors à

	$\mathfrak{p}_1$	$\cdots$	$\mathfrak{p}_{r_1-1}$	$\mathfrak{p}_{r_1}$	$\cdots$	$\mathfrak{p}_{r'}$
x_1	T	$\cdots$	T	A	$\cdots$	A

On considère ensuite les $\mathfrak{p}_j$ qui restent (ceux d'indice supérieur ou égal à r_1). On place le plus à gauche ceux tels que x_2 est transcendant sur k modulo $\mathfrak{p}_j$. Les deux premières lignes ressemblent alors à

	$\mathfrak{p}_1$	$\cdots$	$\mathfrak{p}_{r_1-1}$	$\mathfrak{p}_{r_1}$	$\cdots$	$\mathfrak{p}_{r_2-1}$	$\mathfrak{p}_{r_2}$	$\cdots$	$\mathfrak{p}_{r'}$
x_1	T	$\cdots$	T	A	$\cdots$	A	A	$\cdots$	A
x_2				T	$\cdots$	T	A	$\cdots$	A

En continuant ainsi, on finit par atteindre (à un certain indice $1 \leq t \leq n$) le bord droit du tableau avec des T (nécessairement puisque les idéaux sont supposés non maximaux). Sur K, modulo $\mathfrak{p}_{r_t}, \ldots, \mathfrak{p}_{r'}$ on a x_t transcendant, $x_1, \ldots, x_{t-1}$ algébriques et donc $x_t + \phi(x_1, \ldots, x_{t-1})$ transcendant quel que soit le polynôme $\phi(x_1, \ldots, x_{t-1})$.

Considérons maintenant l'un des idéaux $\mathfrak{p}_j$ pour $r_{t-1} \leq j < r_t$. On a x_{t-1} transcendant sur K modulo $\mathfrak{p}_j$.

J'affirme qu'il existe au plus un exposant a tel que $x_t - x_{t-1}^a$ est algébrique sur K modulo $\mathfrak{p}_j$. Preuve. Si un autre tel exposant b existait, la différence $x_{t-1}^a - x_{t-1}^b$ (et donc x_{t-1}) serait algébrique sur K modulo $\mathfrak{p}_j$. L'affirmation est donc prouvée.

Il existe donc un exposant c tel que $x_t - x_{t-1}^c$ est transcendant sur K modulo tous les idéaux $\mathfrak{p}_{r_{t-1}}, \ldots, \mathfrak{p}_{r_t-1}$. Ce polynôme est donc transcendant sur K modulo $\mathfrak{p}_{r_{t-1}}, \ldots, \mathfrak{p}_{r'}$.

En continuant de proche en proche, on forme le polynôme ϕ désiré. □

7.2 Le lemme de Lazard

Daniel Lazard m'a communiqué le « lemme de Lazard » avec un schéma de preuve quinze jours avant la soutenance de ma thèse [7]. Je n'ai pas eu le temps de l'y insérer. Il a été publié pour la première fois dans [9] avec une preuve incomplète. En plus des résultats cités dans l'introduction de ce chapitre, mentionnons les preuves dues à Josef Schicho et Ziming Li [91], Évelyne Hubert [50] et Brahim Sadik [90]. C'est Ziming Li qui a remarqué le premier que le lemme de Lazard permettait de mener certains calculs « en dimension zéro » (communication orale lors d'une série de séminaires organisée en 1995 au City College de New York par William Sit et Raymond Hobbler).

On adapte les notations fixées au début de la section 7.1. On considère un système triangulaire $A = \{p_1, \ldots, p_n\}$ de l'anneau $R = K[x_1, \ldots, x_n, t_1, \ldots, t_m]$. Chaque polynôme p_i admet x_i pour indéterminée principale. On ne s'intéresse dans cette section qu'à l'idéal $\mathfrak{A} = (A) : S_A^\infty$. On suppose ici aussi qu'il n'est pas trivial. On note $R_0 = K(t_1, \ldots, t_m)[x_1, \ldots, x_n]$ l'anneau obtenu en « faisant passer les indéterminées non principales dans le corps des coefficients » et $\mathfrak{A}_0$ l'idéal $(A) : S_A^\infty$ de R_0. On note M la famille multiplicative de R formée des éléments non nuls de $K[t_1, \ldots, t_m]$.

Théorème 8 *(lemme de Lazard)*
L'idéal $\mathfrak{A}$ est radical.
Les idéaux premiers minimaux sur $\mathfrak{A}$ sont de dimension m et ne rencontrent pas M.

Proposition 39 *Pour démontrer le théorème 8, il suffit de démontrer que $\mathfrak{A}_0$ est radical.*

Preuve La seconde affirmation du théorème est un corollaire du théorème 5. Supposons l'idéal $\mathfrak{A}_0$ radical. Alors l'anneau $R_0/\mathfrak{A}_0$ ne contient aucun élément nilpotent[1] d'après [108, chapter IV, Theorem 10 et Corollary]. Par conséquent, $R/\mathfrak{A}$ n'en contient pas non plus d'après le théorème 3 (parce que si $a \in R/\mathfrak{A}$ est non nul alors son image $a/1$ dans $R_0/\mathfrak{A}_0$ est non nulle aussi ; si une puissance a^d de a était nulle alors son image $(a/1)^d$ le serait aussi et l'anneau $R_0/\mathfrak{A}_0$ contiendrait des éléments nilpotents). Donc $\mathfrak{A}$ est radical et la proposition est prouvée. □

On démontre que $R_0/\mathfrak{A}_0$ est isomorphe à un produit de corps. Comme un produit de corps ne contient pas d'élément nilpotent, l'idéal $\mathfrak{A}_0$ est radical et le lemme de Lazard est prouvé.

[1] Un élément nilpotent d'un anneau R est un élément non nul de R dont une puissance est nulle.

Rappelons que si $R_1, \ldots, R_k$ sont des anneaux, alors le produit cartésien $S = R_1 \times \cdots \times R_k$ peut être muni d'une structure d'anneau, appelé « anneau produit ». Les éléments de S sont des k–uplets. Le zéro de S est $(0, \ldots, 0)$. L'élément un de S est $(1, \ldots, 1)$. L'addition et la multiplication de deux éléments de S se font composante par composante. On voit donc que si les anneaux R_i ne comportent aucun élément nilpotent alors S n'en contient pas non plus. C'est en particulier le cas si les anneaux R_i sont des corps. Le théorème suivant est une généralisation du théorème des restes chinois [37, Exercise 2.6, page 79]. La démonstration est aussi facile que dans le cas des entiers. On le trouve aussi mais sous une forme un peu différente dans [108, chapter III, paragraph 13, Theorem 32]. Il y est formulé en termes de sommes directes d'anneaux au lieu de produits (les deux notions coïncident lorsque les sommes et les produits sont finis). Les règles pour additionner et multiplier les éléments d'une somme directe sont données dans [108, chapter III, paragraph 13, Theorem 30].

Lemme 5 *(théorème des restes chinois)*
Si $\mathfrak{A}_1, \ldots \mathfrak{A}_k$ sont des idéaux de R tels que $R_i + R_j = R$ dès que $i \neq j$ alors l'anneau $R/(\mathfrak{A}_1 \cap \cdots \cap \mathfrak{A}_k)$ est isomorphe à l'anneau produit $(R/\mathfrak{A}_1) \times \cdots \times (R/\mathfrak{A}_k)$.

La proposition ci–dessous conclut la démonstration du lemme de Lazard. Le schéma de peuve est le schéma original de Daniel Lazard.

Proposition 40 *L'anneau $R_0/\mathfrak{A}_0$ est isomorphe à un produit de corps.*

Preuve L'anneau $R_0/\mathfrak{A}_0$ peut se construire incrémentalement : il est isomorphe à l'anneau S_n défini par

$$S_0 = K(t_1, \ldots, t_m), \quad S_i = S_{i-1}[x_i]/(p_i) : s_i^\infty.$$

La démonstration est une récurrence sur n.

La base $n = 0$ est triviale.

Supposons que S_{n-1} soit un produit de corps $K_1 \times \cdots \times K_r$. Alors S_n est isomorphe au produit $(1 \leq j \leq r)$ des anneaux $K_j[x_n]/(p_n) : s_n^\infty$. La seule chose à vérifier (mais c'est immédiat) c'est que l'image du séparant s_n de p_n dans $K_j[x_n]$ est égale au séparant de l'image de p_n dans cet anneau.

Donc, dans $K_j[x_n]$, l'idéal $(p_n) : s_n^\infty$ est engendré par le produit des facteurs irréductibles simples de p_n. Il est donc l'intersection des idéaux maximaux $\mathfrak{m}_\ell$ engendrés par ces facteurs. D'après le théorème des restes chinois, $K_j[x_n]/(p_n) : s_n^\infty$ est isomorphe au produit des corps $K_j[x_n]/\mathfrak{m}_\ell$. D'après les axiomes des produits (l'associativité) l'anneau S_n est un produit de corps. □

7.3 Le lemme de Rosenfeld

Le lemme de Rosenfeld est publié en 1959 dans [87]. Azriel Rosenfeld améliore ainsi (selon ses propres dires) un théorème dont la première version est due à Abraham Seidenberg [95, Theorem 6]. Il existe plusieurs formulations du lemme. Je donne ci–dessous une des plus simples que je connaisse, très proche du texte de 1959.

On note $R = K\{U\}$ un anneau de polynômes différentiels muni de plusieurs dérivations $\delta_1, \ldots, \delta_m$ supposées commuter entre elles. On suppose fixé un classement quelconque sur l'ensemble ΘU. On considère un système A partiellement autoréduit et différentiellement triangulaire (aucun polynôme de A ne comporte une dérivée d'une des dérivées dominantes de A et ces dernières sont distinctes deux–à–deux).

Soit $\{p_1, p_2\}$ une « paire critique » de A c'est–à–dire une paire de polynômes de A dont les dérivées dominantes $\theta_1 u$ et $\theta_2 u$ sont des dérivées d'une même indéterminée différentielle $u \in U$. On note $\theta_{12} u$ la plus petite dérivée commune de $\theta_1 u$ et de $\theta_2 u$. En raison des hypothèses de triangularité faites sur A on a $\theta_{12} u \neq \theta_1 u$ et $\theta_{12} u \neq \theta_2 u$. On définit le Δ–polynôme engendré par la paire critique par la formule suivante, où s_1 et s_2 désignent les séparants de p_1 et de p_2 et où θ_{12}/θ_i désigne l'opérateur de dérivation $\varphi_i \in \Theta$ tel que $\theta_{12} = \theta_i \varphi_i$:

$$\Delta(p_1, p_2) \underset{\text{def}}{=} s_1 \frac{\theta_{12}}{\theta_2} p_2 - s_2 \frac{\theta_{12}}{\theta_1} p_1.$$

Le Δ–polynôme est construit de telle sorte que sa dérivée dominante est strictement inférieure à θ_{12}.

Définition 13 *(paire critique résolue)*

Une paire critique $\{p_1, p_2\}$ *de* A *est dite* résolue *s'il existe un produit de puissances* h *d'initiaux et de séparants de* A *tel que* $h\,\Delta(p_1, p_2)$ *appartienne à l'idéal engendré par l'ensemble* $(\Theta A)_{<\theta_{12} u}$ *des éléments de* ΘA *de dérivée dominante strictement inférieure à* $\theta_{12}\, u$.

Proposition 41 *(critère algorithmique)*

Soit $\{p_1, p_2\}$ *une paire critique de* A. *Si le reste complet de* $\Delta(p_1, p_2)$ *par* A *pour la réduction de Ritt est nul alors la paire critique* $\{p_1, p_2\}$ *est résolue.*

Théorème 9 *(lemme de Rosenfeld)*

Soit A *un système partiellement autoréduit et différentiellement triangulaire d'un anneau* $R = K\{U\}$.

Si toutes les paires critiques de A *sont résolues (propriété dite de « cohérence ») alors tout polynôme différentiel appartenant à l'idéal différentiel* $[A] : H_A^\infty$, *partiellement réduit par rapport à* A, *appartient à l'idéal non différentiel* $(A) : H_A^\infty$.

Preuve Par induction transfinie.

Soient $A = \{p_1, \ldots, p_n\}$ et $f \in [A] : H_A^\infty$ un polynôme différentiel partiellement réduit par rapport à A. Il existe un produit de puissances h d'initiaux et de séparants de A et un nombre fini de polynômes différentiels $b_{\varphi,i}$ non nuls tels que

$$h\, f = \underbrace{\sum_{\varphi \in \Theta} \sum_{i=1}^{n} b_{\varphi,i}\, \varphi\, p_i}_{(\mathscr{F})}.$$

On suppose $f \notin (A) : H_A^\infty$ et on cherche une contradiction. Dans la formule $(\mathscr{F})$ apparaissent donc des dérivées propres des dérivées dominantes de A. Notons $v(\mathscr{F})$ la plus petite d'entre elles pour le classement. Et parmi toutes les formules $(\mathscr{F})$ possibles, on en choisit une telle que $v(\mathscr{F})$ soit minimale. Cette dérivée minimale existe parce que les classements sont des bons ordres. On cherche à construire une autre formule $(\mathscr{F}')$ telle que $v(\mathscr{F}') < v(\mathscr{F})$. Cette contradiction avec l'hypothèse de minimalité prouvera le théorème.

On note $v(\mathscr{F}) = \theta u$ et on suppose que θu est une dérivée propre des dérivées dominantes $\theta_1 u, \ldots, \theta_k u$ des polynômes différentiels $p_1, \ldots, p_k \in A$ en renumérotant les éléments de A si nécessaire. On note $(\theta/\theta_1)\, p_1 = s_1\, \theta u + r_1$. On applique la substitution suivante sur la formule $(\mathscr{F})$:

$$\theta u \longrightarrow \frac{(\theta/\theta_1)\, p_1 - r_1}{s_1}$$

et on multiplie le résultat par une puissance appropriée α du séparant s_1 pour chasser les dénominateurs des fractions rationnelles. On note $\gamma_j = \theta / \operatorname{ppcm}(\theta_1, \theta_j)$ pour $2 \leq j \leq k$. On obtient une formule qui peut s'écrire comme suit, où c, d_j et les $e_{\varphi,j}$ désignent des polynômes en nombre fini et dont les lignes (7.2) et (7.3) ne font figurer que des dérivées strictement inférieures à $v(\mathscr{F})$:

$$\begin{aligned} s_1^\alpha\, h\, f &= c\, \frac{\theta}{\theta_1}\, p_1 && (7.1)\\ &+ \sum_{j=2}^{k} d_j\, \Delta(\gamma_j\, p_1,\, \gamma_j\, p_j) && (7.2)\\ &+ \sum_{\varphi \in \Theta} \sum_{j=1}^{n} e_{\varphi,j}\, \varphi\, p_j. && (7.3) \end{aligned}$$

La dérivée dominante du polynôme $s_i^\alpha\, h\, f$ est elle–aussi strictement inférieure à $v(\mathscr{F})$. La dérivée $v(\mathscr{F})$ ne figure donc que comme dérivée dominante du polynôme $(\theta/\theta_1)\, p_1$. Par conséquent le polynôme c est nécessairement nul.

Si A est un système différentiel ordinaire, la somme (7.2) est vide et la ligne (7.3) fournit la formule $(\mathscr{F}')$ recherchée.

Supposons que A soit un système aux dérivées partielles. Toutes les paires critiques $\{p_1, p_j\}$ sont résolues par hypothèse. D'après le lemme ci–dessous, toutes les paires critiques $\{\gamma_j\, p_1, \gamma_j\, p_j\}$ le sont aussi. Quitte à multiplier à nouveau les deux membres de la formule par une puissance appropriée d'initiaux et de séparants de A on conclut que la somme (7.2) appartient à l'idéal engendré par des éléments de ΘA de dérivée dominante strictement inférieure à $v(\mathscr{F})$. La somme des deux lignes (7.2) et (7.3) fournit donc la formule $(\mathscr{F}')$ recherchée. □

Lemme 6 *Soient $\{p_1, p_2\}$ une paire critique résolue de A et $\gamma \in \Theta$ un opérateur de dérivation quelconque.*

La paire critique $\{\gamma\, p_1, \gamma\, p_2\}$ est résolue.

Preuve Par récurrence sur l'ordre de γ. On remarque pour commencer que

$$\Delta(\gamma\, p_1,\, \gamma\, p_2) = s_1\, \frac{\gamma\, \theta_{12}}{\theta_2}\, p_2 - s_2\, \frac{\gamma\, \theta_{12}}{\theta_1}\, p_1.$$

Base de la récurrence. Si l'ordre est nul alors $\{\gamma\, p_1,\, \gamma\, p_2\} = \{p_1,\, p_2\}$ qui est résolue.

Cas général. On décompose $\gamma = \delta\, \lambda$ avec δ opérateur d'ordre 1. On suppose par récurrence que $\{\lambda\, p_1,\, \lambda\, p_2\}$ est résolue. On veut montrer que $\{\gamma\, p_1,\, \gamma\, p_2\}$ l'est aussi. On note $\varphi = \lambda\, \theta_{12}$ et $\theta = \delta\, \varphi = \gamma\, \theta_{12}$. D'après l'hypothèse de récurrence il existe un produit de puissances h d'initiaux et de séparants de A tel que $h\, \Delta(\lambda\, p_1,\, \lambda\, p_2)$ appartienne à l'idéal engendré par l'ensemble $(\Theta A)_{<\varphi u}$ des éléments de ΘA de dérivée dominante strictement inférieure à $\varphi\, u$. Dérivons cette expression et multiplions–la à nouveau par h. On obtient une somme $(\delta h)\, h\, \Delta(\lambda\, p_1,\, \lambda\, p_2) + h^2\, \delta\, \Delta(\lambda\, p_1,\, \lambda\, p_2)$ dont le premier terme appartient à l'idéal engendré par $(\Theta A)_{<\varphi u}$. Comme $(\Theta A)_{<\varphi u}$ est inclus dans $(\Theta A)_{<\theta u}$ on conclut que le deuxième terme de la somme appartient à l'idéal engendré par $(\Theta A)_{<\theta u}$. Développons ce deuxième terme :

$$\begin{aligned} h^2 \delta \Delta(\lambda\, p_1,\, \lambda\, p_2) &= h^2 \delta \Big\{ s_1\, \frac{\varphi}{\theta_2} p_2 - s_2\, \frac{\varphi}{\theta_1} p_1 \Big\} && (7.4) \\ &= h^2 \Big\{ (\delta s_1) \frac{\varphi}{\theta_2} p_2 - (\delta s_2) \frac{\varphi}{\theta_1} p_1 \Big\} && (7.5) \\ &+ h^2 \Big\{ s_1\, \frac{\theta}{\theta_2} p_2 - s_2\, \frac{\theta}{\theta_1} p_1 \Big\}. && (7.6) \end{aligned}$$

Les polynômes $\varphi/\theta_i\, p_i$ figurant ligne 7.5 appartiennent à l'idéal engendré par $(\Theta A)_{\leq \varphi u}$, lui–même inclus dans l'idéal engendré par $(\Theta A)_{<\theta u}$. Par conséquent, l'expression figurant ligne 7.6, qui n'est autre que $h^2\, \Delta(\gamma\, p_1,\, \gamma\, p_2)$, appartient elle–aussi à cet idéal. La paire critique $\{\gamma\, p_1,\, \gamma\, p_2\}$ est donc résolue. □

7.3.1 La version de Seidenberg

Je donne ci–dessous une autre formulation possible du lemme de Rosenfeld, plus proche de la version initiale de Seidenberg. Tout d'abord, il faut affuter la définition des paires critiques résolues. La définition ci–dessous est plus restrictive que la précédente mais reste compatible avec le critère algorithmique ! Si v est une dérivée quelconque, on note $(\Theta A)_{\leq v}$ l'ensemble des éléments de ΘA de dérivée dominante inférieure ou égale à v, on note $(H_A)_{\leq v}$ l'ensemble des initiaux des éléments de $(\Theta A)_{\leq v}$ et $R_{\leq v}$ l'anneau des polynômes différentiels de dérivée dominante inférieure ou égale à v.

Définition 14 *(paire critique résolue, autre formulation)*

Une paire critique $\{p_1,\, p_2\}$ *de* A *est dite* résolue *s'il existe un produit de puissances* h *d'éléments de* $(H_A)_{<\theta_{12} u}$ *tel que* $h\, \Delta(p_1,\, p_2)$ *appartienne à l'idéal engendré par l'ensemble* $(\Theta A)_{<\theta_{12} u}$.

Théorème 10 *(lemme de Rosenfeld, autre formulation)*

Soient A un système différentiellement triangulaire (mais non nécessairement partiellement autoréduit) d'un anneau $R = K\{U\}$ et v une dérivée quelconque de ΘU.

Si toutes les paires critiques de A sont résolues alors

$$[A] : H_A^\infty \cap R_{\leq v} = ((\Theta A)_{\leq v}) : (H_A)_{\leq v}^\infty.$$

7.4 Le « lifting » du lemme de Lazard

C'est une conséquence des lemmes de Lazard et de Rosenfeld qui est publiée dans [10]. On l'appuie sur la version de Rosenfeld du lemme de Rosenfeld. La jolie preuve du deuxième point du théorème est due à Michel Petitot. Ce théorème implique que la décomposition en chaînes différentielles régulières d'un idéal différentiel présenté par un système A satisfaisant les hypothèses ci–dessous est un problème non différentiel. Ce résultat–là est dû à Évelyne Hubert dans [50].

Théorème 11 *(lifting du lemme de Lazard)*

Soient A un système partiellement autoréduit, différentiellement triangulaire et cohérent d'un anneau $R = K\{U\}$ et R_0 l'anneau des polynômes différentiels partiellement réduits par rapport à A.

L'idéal différentiel $[A] : H_A^\infty$ est radical.

Il y a bijection entre les idéaux différentiels premiers $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ minimaux sur $[A] : H_A^\infty$ et les idéaux premiers $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$ minimaux sur $(A) : H_A^\infty$. La bijection est donnée par $\mathfrak{b}_i = \mathfrak{p}_i \cap R_0$.

Preuve On suppose qu'une puissance p^k d'un polynôme différentiel p appartient à l'idéal $[A] : H_A^\infty$. On montre que p appartient à cet idéal. Soit $\bar{p}$ le reste partiel de p par A pour la réduction de Ritt. D'après le lemme de Rosenfeld, $\bar{p}^k$ appartient à l'idéal $(A) : H_A^\infty$. D'après le lemme de Lazard, $\bar{p}$ aussi appartient à cet idéal. Les polynômes p et $\bar{p}$ sont liés par une relation de la forme suivante, où h désigne un produit de puissances de séparants de A :

$$h\, p \equiv \bar{p} \mod [A].$$

Comme $\bar{p}$ appartient à $[A] : H_A^\infty$, le polynôme p aussi et la radicalité de l'idéal différentiel $[A] : H_A^\infty$ est prouvée.

Le deuxième point du théorème. Les idéaux $\mathfrak{b}_i$ sont premiers et leur intersection est égale à $(A) : H_A^\infty$. Il suffit donc de montrer qu'aucun d'eux n'est redondant. On suppose que $\mathfrak{b}_1$ est redondant et on cherche une contradiction en montrant que $\mathfrak{p}_1$ l'est alors lui–aussi. Soient p un polynôme différentiel appartenant à l'intersection $\mathfrak{p}_2 \cap \cdots \cap \mathfrak{p}_k$ et $\bar{p}$ son reste partiel par A pour la réduction de Ritt. Ce reste partiel $\bar{p}$ appartient à l'intersection $\mathfrak{b}_2 \cap \cdots \cap \mathfrak{b}_k$ et donc aussi à l'idéal $(A) : H_A^\infty$ puisqu'on a supposé $\mathfrak{b}_1$ redondant. Les polynômes p et $\bar{p}$ sont liés par une relation de la forme suivante, où h désigne un produit de puissances de séparants de A :

$$h\, p \equiv \bar{p} \mod [A].$$

Comme $\bar{p}$ appartient à $[A] : H_A^\infty$, le polynôme p aussi et l'idéal différentiel premier $\mathfrak{p}_1$ est redondant. □

Troisième partie

Les mécanismes de scindages et de complétion

Chapitre 8

Étude d'un système lent–rapide

Comme dans les parties précédentes, on commence par montrer une application des méthodes qui sont approfondies ensuite. Il s'agit ici de l'algorithme *Rosenfeld–Gröbner* qui est fondé entre autres sur une technique de discussion de cas, ou de scindages. L'application présentée concerne les équations différentielles ordinaires.

On étudie dans ce chapitre un article de biologie [103]. L'une des étapes décrite dans l'article et menée interactivement en MATHEMATICA par ses auteurs, aurait pu être menée automatiquement en utilisant une méthode d'élimination en algèbre différentielle. La plus grande partie du travail sur lequel je m'appuie pour rédiger ce chapitre est due à Natacha Skrzypczak [98] sous la direction de Michel Petitot.

L'article [103] étudie un modèle d'horloge circadienne, c'est–à–dire d'horloge de période approximativement égale à 24 heures. Cette horloge, qui est mise en œuvre par des réactions biochimiques au cœur d'une cellule, est modélisée par un système de 9 équations différentielles ordinaires en 9 indéterminées différentielles, dépendant de paramètres. Dans la première partie de l'article, les auteurs cherchent quelles conditions les paramètres doivent satisfaire pour que le système différentiel ait un comportement oscillant, c'est–à–dire se comporte comme une horloge. Voici leur idée : il est possible d'approximer le système initial de 9 équations en 9 inconnues par un système de 2 équations en 2 inconnues et les valeurs des paramètres pour lesquelles les deux systèmes oscillent devraient approximativement correspondre (ils le vérifient expérimentalement). Pourquoi un système de 2 équations en 2 inconnues ? Parce que dans ce contexte, des théorèmes particuliers s'appliquent et notamment le théorème de Poincaré–Bendixson [45, Theorem 16.1] qui fournit les conditions recherchées. Dans la deuxième partie de l'article, les auteurs montrent que même lorsque les paramètres ont des valeurs telles que le système ne devrait pas osciller, la présence de bruit peut le faire osciller quand même.

Que dit le théorème de Poincaré–Bendixson et comment l'applique–t–on ici ? Informellement, le théorème dit que si les trajectoires d'un système différentiel autonome en deux indéterminées différentielles restent confinées dans une région bornée et si le système n'admet aucun point fixe attractif dans cette région alors la région contient des cycles limites. Les modèles différentiels de systèmes biologiques sont toujours autonomes (c'est–à–dire que les équations ne dépendent pas explicitement du temps). Les trajectoires sont confinées parce

qu'elles décrivent des phénomènes biologiques et ne peuvent donc pas tendre vers l'infini. Pourquoi s'intéresser à des cycles limites ? Parce que la présence d'un cycle limite dans l'espace des phases (c'est–à–dire en projetant les trajectoires des deux variables sur le plan perpendiculaire à l'axe du temps) correspond à un comportement oscillant des deux variables — ce qu'on cherche. Le système considéré admet un unique point fixe (pour des valeurs des variables ayant un sens biologique). On étudie sa stabilité en linéarisant le système différentiel en son voisinage et en étudiant le signe des parties réelles des valeurs propres de la matrice M des coefficients du système linéarisé : le point est instable si et seulement si l'une des deux valeurs propres a une partie réelle positive. Les auteurs négligent le cas de deux valeurs propres réelles de signe contraire (cas d'un point fixe de type « col ») et je ne sais pas pourquoi. Si on excepte ce cas, les parties réelles des valeurs propres de la matrice M ont même signe, qui est égal au signe de la trace de la matrice, c'est–à–dire de la somme des éléments diagonaux de M. Il suffit donc de discuter du signe de cette trace, qui dépend des paramètres du modèle.

8.1 Le modèle initial

Le modèle comporte deux gènes : un activateur $\mathscr{A}$ et un répresseur $\mathscr{R}$ qui sont transcrits en ARN messagers M_A et M_R. Les ARN messagers sont ensuite eux–mêmes traduits en protéines A et R. La protéine A se fixe sur le promoteur du gène $\mathscr{A}$. Ce faisant, elle accélère la transcription de $\mathscr{A}$ en ARN messager. Le protéine A se fixe aussi sur le promoteur du gène $\mathscr{R}$. Ce faisant, elle accélère aussi la transcriprion de $\mathscr{R}$ en ARN messager. Les deux protéines A et R réagissent entre elles et produisent un complexe C. On voit qu'en produisant de la protéine A, le gène $\mathscr{A}$ a tendance à accelérer la réaction alors qu'en produisant de la protéine R qui capture A pour former le complexe C, le gène $\mathscr{R}$ a tendance à freiner la réaction. Les 9 variables du modèle sont les suivantes :

- M_A désigne la concentration d'ARN messager produit par $\mathscr{A}$;
- M_R désigne la concentration d'ARN messager produit par $\mathscr{R}$;
- A, R et C désignent les concentrations de protéines A, R et C.

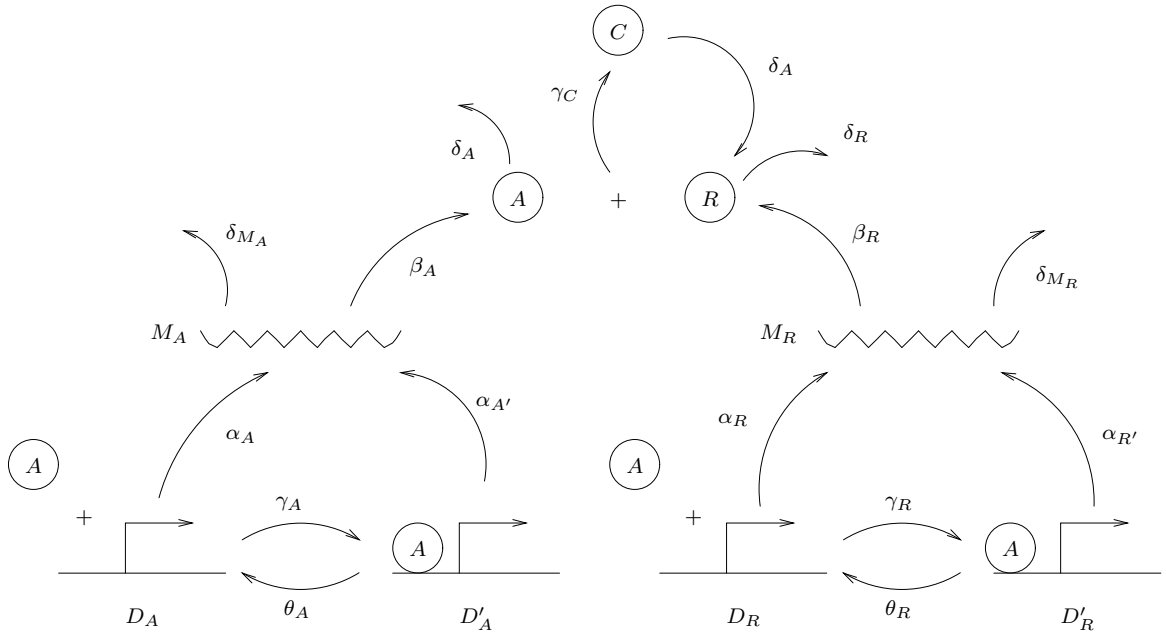
Pour chacun des gènes $\mathscr{A}$ et $\mathscr{R}$ on introduit deux variables afin de distinguer la situation où la protéine A n'est pas fixée sur le promoteur du gène de la situation où la protéine A est fixée sur le promoteur. On obtient donc quatre variables. Ces quatre variables ne sont pas des concentrations puisque, pour le type d'organisme extrêmement simple considéré, on peut penser qu'il n'y a qu'un seul exemplaire de chacun des gènes dans la cellule. Il faut plutôt les voir comme des espérances.

- D_A représente l'état du gène $\mathscr{A}$ sur le promoteur duquel la protéine A n'est pas fixée ;
  D'_A représente l'état du gène $\mathscr{A}$ sur le promoteur duquel la protéine A est fixée ;
- D_R représente l'état du gène $\mathscr{R}$ sur le promoteur duquel la protéine A n'est pas fixée ;
  D'_R représente l'état du gène $\mathscr{R}$ sur le promoteur duquel la protéine A est fixée ;

Voici les équations du modèle ainsi qu'une représentation graphique.

$$
\begin{aligned}
\dot{D}_A &= \theta_A\, D'_A - \gamma_A\, A\, D_A,\\
\dot{D}_R &= \theta_R\, D'_R - \gamma_R\, A\, D_R,\\
\dot{D}'_A &= \gamma_A\, A\, D_A - \theta_A\, D'_A,\\
\dot{D}'_R &= \gamma_R\, A\, D_R - \theta_R\, D'_R,\\
\dot{M}_A &= \alpha_{A'}\, D'_A + \alpha_A\, D_A - \delta_{M_A}\, M_A,\\
\dot{M}_R &= \alpha_{R'}\, D'_R + \alpha_R\, D_R - \delta_{M_R}\, M_R,\\
\dot{A} &= \beta_A\, M_A + \theta_A\, D'_A + \theta_R\, D'_R - \gamma_A\, A\, D_A - \gamma_R\, A\, D_R - \gamma_C\, A\, R - \delta_A\, A,\\
\dot{R} &= \beta_R\, M_R - \gamma_C\, A\, R + \delta_A\, C - \delta_R\, R,\\
\dot{C} &= \gamma_C\, A\, R - \delta_A\, C.
\end{aligned}
$$

Les lettres grecques désignent des constantes. Les équations sont polynomiales de degré deux. Les non linéarités sont dues à la loi d'action de masse.



Comment déduit–on un système d'équations différentielles d'un tel modèle ? Il suffit de considérer les échanges (les flèches) les uns après les autres comme on l'a fait au chapitre 4. Chaque échange ajoute un terme positif dans le membre droit de l'équation qui définit la cible de l'échange ainsi qu'un terme négatif dans le membre droit des équations qui définissent la ou les sources. La plupart des échanges (ceux à une source et une cible) sont linéaires. Ils sont notés

$$X \xrightarrow[\alpha]{} Y.$$

Lorsque la source X est une protéine, on obtient

$$\dot{X} = -\alpha\, X, \quad \dot{Y} = \alpha\, X.$$

Lorsque la source est de l'ADN ou de l'ARN messager, les équations obtenues sont un peu différentes dans la mesure où l'ADN et l'ARN ne sont pas consommés par la réaction. On

obtient alors seulement

$$\dot{Y} = \alpha\, X.$$

Certaines flèches n'ont pas de cible. Elles indiquent que la source se dégrade. L'échange est supposé linéaire. Les protéines peuvent se dégrader. L'ARN messager aussi. Les autres réactions font intervenir deux réactants. Elles sont notées

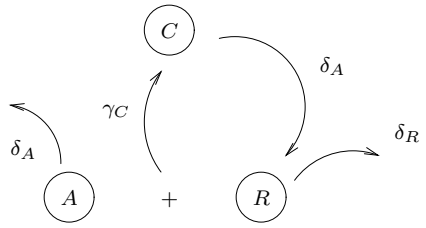
$$X + Z \xrightarrow[\alpha]{} Y.$$

Les équations se construisent alors en utilisant la loi d'action de masse qui dit en substance que sur un petit intervalle de temps, la quantité de Y produit est proportionnelle au produit des concentrations des réactants X et Z. La réaction est faible lorsque l'un des deux réactants vient à manquer. Elle est forte lorsque les deux concentrations sont élevées. On obtient les équations suivantes :

$$\dot{X} = -\alpha\, X\, Z, \quad \dot{Z} = -\alpha\, X\, Z, \quad \dot{Y} = \alpha\, X\, Z.$$

Appliquons maintenant ces considérations générales sur l'exemple.

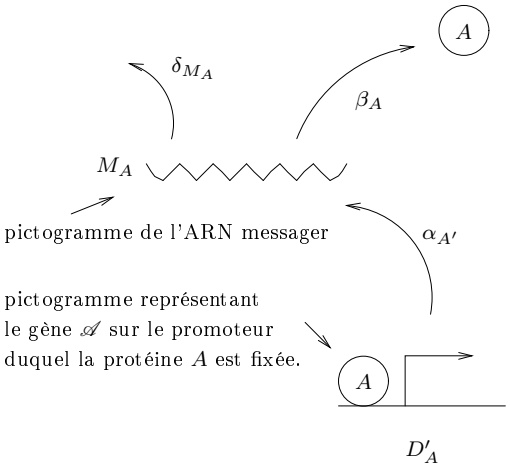
La figure suivante représente les deux protéines A et C réagissant pour fournir du complexe C suivant la loi d'action de masse (constante γ_C). Le complexe peut se casser et produit alors de la protéine R suivant un échange linéaire (constante δ_A). Les deux protéines A et R peuvent aussi se dégrader (constantes δ_A et δ_R).



$$\begin{aligned}
\dot{A} &= -\delta_A\, A - \gamma_C\, A\, R,\\
\dot{R} &= -\delta_R\, R - \gamma_C\, A\, R + \delta_A\, C,\\
\dot{C} &= \gamma_C\, A\, R - \delta_A\, C.
\end{aligned}$$

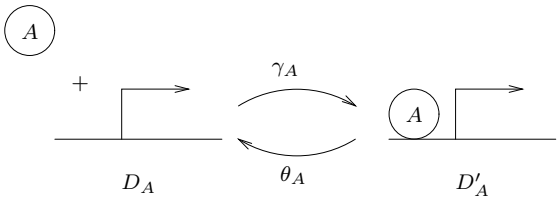
La figure suivante montre la contribution du gène $\mathscr{A}$ sur le promoteur duquel la protéine A est fixée dans la production d'ARN messager. Cette transcription ne détruit pas le gène $\mathscr{A}$. Elle est assimilée à un échange linéaire de constante $\alpha_{A'}$. L'ARN messager est traduit en protéine A. Cette traduction ne modifie pas la concentration d'ARN messager. Elle est assimilée à un échange linéaire de constante β_A. Parallèlement, l'ARN messager peut se

dégrader (constante δ_{M_A}).



$$\begin{aligned} \dot{D}'_A &= 0, \\ \dot{M}_A &= \alpha_{A'}\, D'_A - \delta_{M_A}\, M_A, \\ \dot{A} &= \beta_A\, M_A \end{aligned}$$

La figure suivante décrit la fixation de la protéine A sur le promoteur du gène $\mathscr{A}$. La protéine A peut se fixer sur le promoteur du gène $\mathscr{A}$. Cette réaction « produit » du gène $\mathscr{A}$ sur le promoteur duquel la protéine A est fixée. Elle suit la loi d'action de masse (constante γ_A). La protéine A peut aussi se détacher du promoteur de $\mathscr{A}$. Cette réaction « produit » du gène $\mathscr{A}$ sur le promoteur duquel la protéine A n'est pas fixée suivant une loi linéaire de constante θ_A.



$$\begin{aligned} \dot{D}_A &= -\gamma_A\, A\, D_A + \theta_A\, D'_A, \\ \dot{D}'_A &= \gamma_A\, A\, D_A - \theta_A\, D'_A. \end{aligned}$$

8.2 Réduction exacte du modèle

On commence par réduire le modèle initial en cherchant des lois de conservation et en particulier, sur ce type de modèle, des lois linéaires. Ici on trouve deux lois de conservation linéaires :

$$D_A + D'_A = D_{A_0} = \textit{constante}, \qquad D_R + D'_R = D_{R_0} = \textit{constante}.$$

Comme l'ont remarqué Michel Petitot et Natacha Skrzypczak, ce type de loi, qui survient très naturellement dans ce genre de modèle, peut s'obtenir symboliquement très facilement. On commence par nommer chaque terme figurant dans le membre droit des équations par une inconnue. Par exemple, les équations définissant D_A et D'_A

$$\begin{aligned} \dot{D}_A &= \theta_A\, D'_A - \gamma_A\, A\, D_A, \\ \dot{D}'_A &= \gamma_A\, A\, D_A - \theta_A\, D'_A \end{aligned}$$

peuvent se noter comme suit, en introduisant deux inconnues v_1 et v_2 :

$$\begin{aligned} \dot{D}_A &= v_1 - v_2, \\ \dot{D}'_A &= v_2 - v_1. \end{aligned}$$

Il suffit ensuite de procéder à une élimination linéaire (à un pivot de Gauss par exemple) sur le système ci–dessus en adoptant l'ordre sur les variables suivant : $(v_1, v_2) \gg (\dot{D}_A, \dot{D}'_A)$ pour obtenir la relation

$$\dot{D}_A + \dot{D}'_A = 0$$

qu'on traduit en

$$D_A + D'_A = constante.$$

Chacune des lois de conservation ainsi trouvées permet de supprimer une des équations différentielles du modèle. On obtient ainsi un nouveau système de 7 équations différentielles en 7 indéterminées différentielles, strictement équivalent au système initial. Le système ci–dessous a été obtenu de cette manière en éliminant D'_A et D'_R.

$$\begin{aligned} \dot{D}_A &= \theta_A(D_{A_0} - D_A) - \gamma_A\, D_A\, A, \\ \dot{D}_R &= \theta_R\,(D_{R_0} - D_R) - \gamma_R\, D_R\, A, \\ \dot{M}_A &= \alpha'_A\,(D_{A_0} - D_A) + \alpha_A\, D_A - \delta_{M_A}\, M_A, \\ \dot{M}_R &= \alpha'_R\,(D_{R_0} - D_R) + \alpha_R\, D_R - \delta_{M_R}\, M_R, \\ \dot{A} &= \beta_A\, M_A + \theta_A\,(D_{A_0} - D_A) + \theta_R\,(D_{R_0} - D_R) \\ &\quad -A\,(\gamma_A\, D_A + \gamma_R\, D_R + \gamma_C\, R + \delta_A), \\ \dot{R} &= \beta_R\, M_R - \gamma_C\, A\, R + \delta_A\, C - \delta_R\, R, \\ \dot{C} &= \gamma_C\, A\, R - \delta_A\, C. \end{aligned}$$

8.3 Le modèle en BLAD après réduction exacte

Le système différentiel ci–dessus forme une chaîne différentielle régulière pour tout classement compatible avec l'ordre total sur les indéterminées différentielles :

$$(variables) \gg (paramètres).$$

Le programme C suivant commence par définir un tel classement. Le système est alors affecté à une chaîne différentielle régulière, nommée *equadiffs*. On remarque que, bien que les chaînes régulières soient constituées de polynômes, les parseurs autorisent l'entrée de fractions rationnelles, dans la mesure où les dénominateurs des fractions sont les initiaux des polynômes.

La boucle finale montre comment on peut utiliser l'algorithme de forme normale modulo une chaîne différentielle régulière pour spécialiser les équations différentielles en les valeurs des paramètres. Les valeurs attribuées ici sont celles considérées par les auteurs de [103] en deuxième page. On utilise une chaîne régulière *params* pour lier les paramètres et les valeurs numériques. Pour spécialiser une équation différentielle, il suffit de calculer sa forme normale modulo *params*.

```
/* File natural_ordering.c */

#include "bad.h"

int main ()
{
    struct bad_regchain params, equadiffs;
    struct bap_ratfrac_mpz Q;
    bav_Iordering r;
    struct ba0_mark M;
    int i;

    bad_restart (0,0);
    ba0_record (&M);
/*
   The natural ordering involves an orderly block for Ma, Da, Dr, Mr, A, R, C
   and another block for the parameters.
*/
    ba0_sscanf2
        ("ordering (derivations = [t], \
                    blocks = [[Ma, Da, Dr, Mr, A, R, C], \
                        [alphaA, alphaA_, alphaR, alphaR_, betaA, betaR, \
                         deltaMa, deltaMr, deltaA, deltaR, \
                         gammaA, gammaR, gammaC, thetaA, thetaR, Da0, Dr0]])",
         "%ordering", &r);
    bav_R_push_ordering (r);
/*
   The parameters values are stored in a regular chain
*/
    bad_init_regchain (&params);
    ba0_sscanf2
        ("regchain (\
                [alphaA - 50, alphaA_ - 500, alphaR - 1/100, alphaR_ - 50,\
                 betaA - 50, betaR - 5, \
                 deltaMa - 10, deltaMr - 1/2, deltaA - 1, \
                 deltaR - 1/5, \
                 gammaA - 1, gammaR - 1, gammaC - 2,\
                 thetaA - 50, thetaR - 100,\
                 Da0 - 1, Dr0 - 1], \
                [prime, autoreduced, primitive, squarefree, normalized])",
        "%regchain", &params);
/*
   The 7 ODE initial system forms a regular differential chain.
*/
    bad_init_regchain (&equadiffs);
```

```
    ba0_sscanf2
        ("regchain (\
               [Da[t] - (thetaA*(Da0 - Da) - gammaA*Da*A),\
                Dr[t] - (thetaR*(Dr0 - Dr) - gammaR*Dr*A),\
                Ma[t] - (alphaA_*(Da0 - Da) + alphaA*Da - deltaMa*Ma),\
                A[t] - (betaA*Ma + thetaA*(Da0 - Da) + thetaR*(Dr0 - Dr)\
                        - A*(gammaA*Da + gammaR*Dr + gammaC*R + deltaA)),\
                Mr[t] - (alphaR_*(Dr0 - Dr) + alphaR*Dr - deltaMr*Mr),\
                R[t] - (betaR*Mr - gammaC*A*R + deltaA*C - deltaR*R),\
                C[t] - (gammaC*A*R - deltaA*C)],\
               [prime, differential, autoreduced, primitive, squarefree, \
                                                  coherent, normalized])",
        "%regchain", &equadiffs);
/*
   To specialize the ODE system with the parameters values, it suffices
   to use a normal form algorithm.
*/
    bap_init_ratfrac_mpz (&Q);
    for (i = 0; i < equadiffs.decision_system.size; i++)
    {   bad_normal_form_polynom_mod_regchain
            (&Q, equadiffs.decision_system.tab [i],
                                                  &params, (bap_polynom_mpz*)0);
        ba0_printf ("%Qz\n", &Q);
    }
/*
   Exit
*/
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu en exécutant le programme.

```
$ ./natural_ordering
parameters = regchain ([Dr0 - 1, Da0 - 1, thetaR - 100, thetaA - 50, gammaC - 2
, gammaR - 1, gammaA - 1, 5*deltaR - 1, deltaA - 1, 2*deltaMr - 1, deltaMa - 10
, betaR - 5, betaA - 50, alphaR_ - 50, 100*alphaR - 1, alphaA_ - 500, alphaA -
50], [prime, autoreduced, primitive, squarefree, normalized])
system = regchain ([C[t] - A*R*gammaC + C*deltaA, R[t] - Mr*betaR + A*R*gammaC
 + R*deltaR - C*deltaA, A[t] - Ma*betaA + Da*A*gammaA + Da*thetaA + Dr*A*gammaR
 + Dr*thetaR + A*R*gammaC + A*deltaA - thetaA*Da0 - thetaR*Dr0, Mr[t] - Dr*
alphaR + Dr*alphaR_ + Mr*deltaMr - alphaR_*Dr0, Dr[t] + Dr*A*gammaR + Dr*thetaR
 - thetaR*Dr0, Da[t] + Da*A*gammaA + Da*thetaA - thetaA*Da0, Ma[t] + Ma*deltaMa
 - Da*alphaA + Da*alphaA_ - alphaA_*Da0], [differential, prime, autoreduced,
```

```
primitive, squarefree, coherent, normalized])
C[t] - 2*A*R + C
5*R[t] - 25*Mr + 10*A*R + R - 5*C
A[t] - 50*Ma + Da*A + 50*Da + Dr*A + 100*Dr + 2*A*R + A - 150
100*Mr[t] + 4999*Dr + 50*Mr - 5000
Dr[t] + Dr*A + 100*Dr - 100
Da[t] + Da*A + 50*Da - 50
Ma[t] + 10*Ma + 450*Da - 500
```

8.4 Approximation du modèle

On cherche à approximer un système de 7 équations différentielles en 7 variables par un système de 2 équations en 2 variables. Pour éliminer 5 variables, l'idée consiste à séparer ces dernières en variables « lentes » et « rapides ». Voici l'idée, énoncée informellement. Considérons un système différentiel de la forme suivante, où ε désigne une petite constante positive :

$$\dot{x} = f(x, y), \quad \varepsilon\, \dot{y} = g(x, y).$$

En un point $(x, y) \in \mathbb{R}^2$ quelconque et en particulier au voisinage des conditions initiales, la vitesse $\dot{y}$ est grande : la variable y est rapide et tend très rapidement vers une région de $\mathbb{R}^2$ où $g(x, y) \simeq 0$. Face à un tel système, il est donc raisonnable de penser que, après une première phase transitoire, le système peut être approché par le système suivant :

$$\dot{x} = f(x, y), \quad 0 = g(x, y).$$

De tels systèmes, qui mélangent équations différentielles et équations purement algébriques ne sont pas simples à intégrer numériquement. En particulier, les intégrateurs numériques ne peuvent généralement pas garantir que les trajectoires des variables du systèmes restent sur la variété $g(x, y) = 0$. De même, lorsque le système comporte plusieurs équations algébriques $g_i = 0$, il peut exister entre les variables du système des relations « cachées » conséquences des g_i qui doivent être satisfaites. L'élimination différentielle permet en quelque sorte de simplifier de tels systèmes afin de supprimer ces relations cachées.

Les auteurs de l'article [103] ont choisi de considérer que 5 des 7 variables du système sont rapides et que les 2 autres (R et C) sont lentes, ce qui les a conduit à s'intéresser au système différentiel–algébrique suivant :

$$\begin{aligned}
0 &= \theta_A(D_{A_0} - D_A) - \gamma_A\, D_A\, A, \\
0 &= \theta_R\,(D_{R_0} - D_R) - \gamma_R\, D_R\, A, \\
0 &= \alpha'_A\,(D_{A_0} - D_A) + \alpha_A\, D_A - \delta_{M_A}\, M_A, \\
0 &= \alpha'_R\,(D_{R_0} - D_R) + \alpha_R\, D_R - \delta_{M_R}\, M_R, \\
0 &= \beta_A\, M_A + \theta_A\,(D_{A_0} - D_A) + \theta_R\,(D_{R_0} - D_R) \\
&\quad -A\,(\gamma_A\, D_A + \gamma_R\, D_R + \gamma_C\, R + \delta_A), \\
\dot{R} &= \beta_R\, M_R - \gamma_C\, A\, R + \delta_A\, C - \delta_R\, R, \\
\dot{C} &= \gamma_C\, A\, R - \delta_A\, C.
\end{aligned}$$

Ils ont ensuite procédé à une élimination différentielle interactivement en MATHEMATICA pour obtenir le modèle réduit qu'ils cherchaient. Les bibliothèques *BLAD* et en particulier l'algorithme *Rosenfeld–Gröbner* permettent d'automatiser ce calcul. Le programme C suivant effectue cette simplification. C'est une extension du précédent.

```
/* File differential_elimination.c */

#include "bad.h"

int main ()
{
    struct bad_regchain params, equadiffs;
    struct bad_intersectof_regchain ideal;
    bap_ratfrac_mpz Q;
    struct bap_tableof_polynom_mpz eqns, ineqns;
    struct bap_polynom_mpz poly;
    struct bav_tableof_variable fast_vars;
    bav_Iordering r, s;
    bav_variable v;
    struct ba0_mark M;
    int i;

    bad_restart (0,0);
    ba0_record (&M);
    ba0_init_table ((ba0_table)&fast_vars);
/*
   Ordering w.r.t. which the input system is a regular chain.
*/
    ba0_sscanf2
        ("ordering (derivations = [t], \
                    blocks = [[Ma, Da, Dr, Mr, A, R, C], \
                        [alphaA, alphaA_, alphaR, alphaR_, betaA, betaR, \
                         deltaMa, deltaMr, deltaA, deltaR, \
                         gammaA, gammaR, gammaC, thetaA, thetaR, Da0, Dr0]])",
         "%ordering", &r);
    bav_R_push_ordering (r);
/*
   The parameters values. They are stored in a regular chain.
*/
    bad_init_regchain (&params);
    ba0_sscanf2
        ("regchain (\
                [alphaA - 50, alphaA_ - 500, alphaR - 1/100, alphaR_ - 50,\
                 betaA - 50, betaR - 5, \
                 deltaMa - 10, deltaMr - 1/2, deltaA - 1, \
                 deltaR - 1/5, \
```

```
                gammaA - 1, gammaR - 1, gammaC - 2,\
                thetaA - 50, thetaR - 100,\
                Da0 - 1, Dr0 - 1], \
               [prime, autoreduced, primitive, squarefree, normalized])",
        "%regchain", &params);
    ba0_printf ("parameters = %regchain\n", &params);
/*
   The input system.
*/
    bad_init_regchain (&equadiffs);
    ba0_sscanf2
        ("regchain (\
               [Da[t] - (thetaA*(Da0 - Da) - gammaA*Da*A),\
                Dr[t] - (thetaR*(Dr0 - Dr) - gammaR*Dr*A),\
                Ma[t] - (alphaA_*(Da0 - Da) + alphaA*Da - deltaMa*Ma),\
                A[t] - (betaA*Ma + thetaA*(Da0 - Da) + thetaR*(Dr0 - Dr)\
                        - A*(gammaA*Da + gammaR*Dr + gammaC*R + deltaA)),\
                Mr[t] - (alphaR_*(Dr0 - Dr) + alphaR*Dr - deltaMr*Mr),\
                R[t] - (betaR*Mr - gammaC*A*R + deltaA*C - deltaR*R),\
                C[t] - (gammaC*A*R - deltaA*C)],\
               [prime, differential, autoreduced, primitive, squarefree, \
                                                 coherent, normalized])",
        "%regchain", &equadiffs);
    ba0_printf ("system = %regchain\n", &equadiffs);
/*
   The ranking w.r.t. which computations must be performed.
   The fast variables (assumed to be steady).
*/
    ba0_sscanf2
        ("ordering (derivations = [t], \
                    blocks = [[Ma, Da, Dr, Mr], [A, R, C], \
                        [alphaA, alphaA_, alphaR, alphaR_, betaA, betaR, \
                         deltaMa, deltaMr, deltaA, deltaR, \
                         gammaA, gammaR, gammaC, thetaA, thetaR, Da0, Dr0]])",
         "%ordering", &s);
    ba0_sscanf2 ("[Da, Dr, Ma, Mr, A]", "%t[%v]", &fast_vars);
/*
   Construction of the table of the equations to be processed.
   The parameters values are substituted to the parameters.
   Every differential equation x' = f(x) s.t. x is a fast variable
        is replaced by f(x) = 0
   All equations are converted from ordering r to ordering s.
*/
    ba0_init_table ((ba0_table)&eqns);
    ba0_realloc2_table ((ba0_table)&eqns, equadiffs.decision_system.size,
```

```
                        (ba0_new_function*)bap_new_polynom_mpz);
    bap_init_polynom_mpz (&poly);
    for (i = 0; i < equadiffs.decision_system.size; i++)
    {   Q = bap_new_ratfrac_mpz ();
        v = bap_leader_polynom_mpz (equadiffs.decision_system.tab [i]);
        v = bav_order_zero_variable (v);
        if (ba0_member_table (v, (ba0_table)&fast_vars))
        {   bap_reductum_polynom_mpz
                                (&poly, equadiffs.decision_system.tab [i]);
            bav_R_push_ordering (s);
            bap_sort_polynom_mpz (&poly, &poly);
        } else
        {   bav_R_push_ordering (s);
            bap_sort_polynom_mpz (&poly, equadiffs.decision_system.tab [i]);
        }
        bad_normal_form_polynom_mod_regchain
                                (Q, &poly, &params, (bap_polynom_mpz*)0);
        bap_set_polynom_mpz (eqns.tab [eqns.size++], &Q->numer);
        bav_R_pull_ordering ();
    }
    ba0_printf ("equations to process = %t[%Az]\n", &eqns);
/*
   Call to Rosenfeld_Groebner
*/
    bav_R_push_ordering (s);
    bad_reduction_strategy = bad_gcd_prem_and_factor_reduction_strategy;
    bad_init_intersectof_regchain (&ideal);
    ba0_sscanf2
        ("intersectof_regchain ([], [differential, autoreduced, primitive, \
        squarefree, coherent, normalized])", "%intersectof_regchain", &ideal);
    ba0_init_table ((ba0_table)&ineqns);
    bad_Rosenfeld_Groebner (&ideal, &eqns, &ineqns, (bav_tableof_variable)0);
    ba0_printf
        ("result of differential elimination = %intersectof_regchain\n",
         &ideal);
/*
   Exit from the program
*/
    bav_R_pull_ordering ();
    bav_R_pull_ordering ();
    ba0_restore (&M);
    bad_terminate (ba0_init_level);
    return 0;
}
```

Pour procéder à l'élimination différentielle, on crée un classement s qui élimine les va-

riables rapides. Ce devrait être

$$(\textit{variables rapides}) \gg (\textit{variables lentes}) \gg (\textit{paramètres}).$$

Toutefois, pour reproduire exactement le résultat obtenu dans [103] et pour éviter de faire grossir inutilement les données, on garde la variable A dans le deuxième bloc, c'est–à–dire parmi les variables lentes. Remarquer que le classement s n'est pas empilé, de telle sorte que le classement courant est toujours le classement r défini en section 8.3.

Les 5 variables rapides sont enregistrées dans un tableau de variables nommé *fast_ vars*. Le tableau *eqns* des équations du système différentiel–algébrique est ensuite rempli. L'algorithme de forme normale est utilisé pour spécialiser les équations en les valeurs des paramètres. On applique le traitement suivant à tout élément p du tableau *equadiffs* : si la dérivée dominante de p est la dérivée d'une variable rapide alors le « reductum » de p est enregistré dans la table sinon, c'est p lui–même qui est enregistré. On rappelle que le reductum d'un polynôme $\dot{x} + f(x, y)$ de dérivée dominante $\dot{x}$ est le polynôme $f(x, y)$.

Les appels à *bap_ sort_ polynom_ mpz* permettent de réordonner les monômes qui constituent les équations en fonction du classement s. La variable *ideal* est initialisée. Elle est destinée à recevoir le résultat de l'élimination. Un choix heuristique est effectué sur la façon de procéder aux réductions de Ritt. Le classement s est empilé et devient le classement courant. L'élimination différentielle est effectuée par un appel à *Rosenfeld–Gröbner*.

Exécutons ce programme. Il se trouve qu'une seule chaîne différentielle régulière est produite : l'exemple n'a pas produit de scindage. Les trois premières équations de la chaîne sont celles données par les auteurs de [103] au bas de la deuxième page. Les équations sont ici spécialisées.

```
$ ./differential_elimination
equations to process = [C[t] - 2*A*R + C, -25*Mr + 5*R[t] + 10*A*R + R - 5*C, -
50*Ma + Da*A + 50*Da + Dr*A + 100*Dr + 2*A*R + A - 150, 4999*Dr + 50*Mr - 5000
, Dr*A + 100*Dr - 100, Da*A + 50*Da - 50, 10*Ma + 450*Da - 500]
result of differential elimination = intersectof_regchain ([regchain ([2*A^2*R
 + A^2 + 100*A*R - 2450*A - 12500, C[t] - 2*A*R + C, 1000*R[t]*R + 24250*R[t]
 + 2000*A*R^2 + 38502*A*R - 4999*A + 200*R^2 - 1000*R*C + 4750*R - 24250*C +
622450, 5000*Mr*R + 121250*Mr - 9998*A*R - 4999*A - 100*R + 622450, 100*Dr*R +
2425*Dr + 2*A*R + A - 100*R - 2550, 2250*Da + 2*A*R + A - 2500, 50*Ma - 2*A*R
 - A], [differential, autoreduced, primitive, squarefree, coherent, normalized]
)], [differential, autoreduced, primitive, squarefree, coherent, normalized])
```

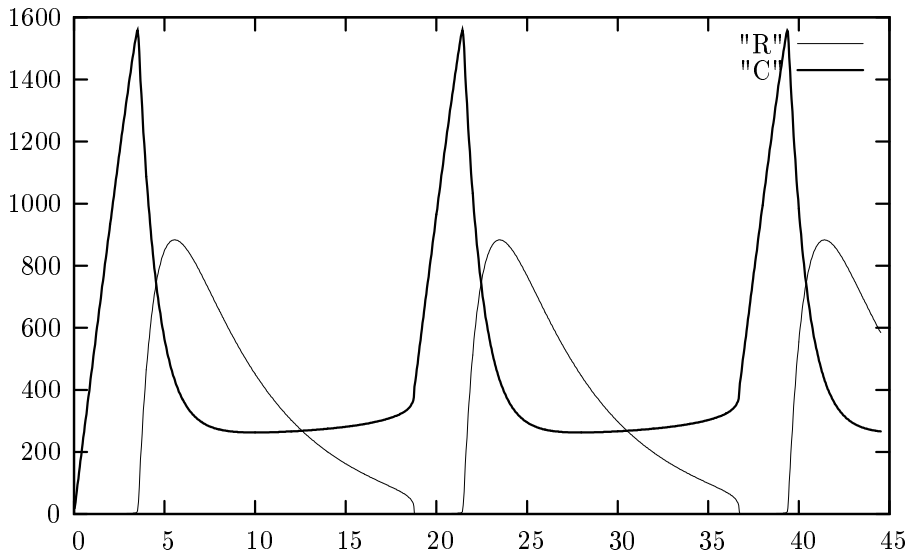
Peut–on effectuer l'élimination différentielle sur le système générique, c'est–à–dire avec des paramètres symboliques ? Il y a en fait une difficulté. Si on applique *Rosenfeld–Gröbner* sur le système générique on va traiter les paramètres comme de simples indéterminées différentielles et, même si on ajoute des équations supplémentaires signifiant que les paramètres sont des constantes, *Rosenfeld–Gröbner* risque de se lancer dans des calculs compliqués et de discuter de la nature du résultat de l'élimination en fonction de l'annulation ou de la non annulation de diverses relations algébriques entre les paramètres.

Ces discussions de cas sont sans objet. Les paramètres sont attachés à des valeurs qui ne sont pas très précises et qui peuvent varier expérimentalement. Il faudrait pouvoir signifier à l'algorithme de ne considérer séparément aucun cas où une relation algébrique entre les paramètres serait satisfaite. Dans le paquetage *diffalg* de MAPLE, c'est possible en « faisant entrer les paramètres dans le corps des coefficients » des équations. En *BLAD*, il suffit d'enregistrer les paramètres dans un tableau de de passer ce tableau en quatrième paramètre à la fonction *bad_Rosenfeld_Groebner*. Cette fonctionnalité est prévue dans l'interface de la fonction. Elle n'est pas encore implantée dans la version 1.4 des bibliothèques.

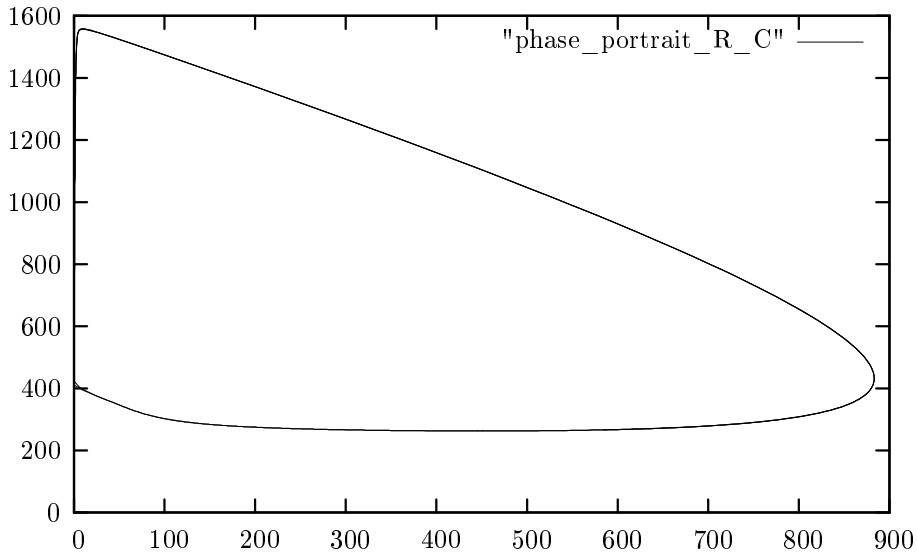
8.5 Simulation numérique

On trouve ci–dessous plusieurs courbes obtenues par simulation numérique à partir du résultat de l'élimination différentielle. De telles routines de simulation numérique ne figurent pas dans les bibliothèques *BLAD* bien que l'inclusion d'un intégrateur numérique général pour chaînes différentielles régulières soit un de mes objectifs importants à court terme.

Certains des calculs qui ont produit les courbes ci–dessous ont été réalisés avec la *Gnu Scientific Library*. J'ai programmé une variante de l'intégrateur *RadauIIA* conçu par Ernst Hairer (avec un pas fixe) [46, page 72] qui se comporte expérimentalement plutôt bien face à des systèmes « raides ». Les systèmes obtenus après élimination différentielle sur des systèmes mélangeant équations différentielles et algébriques semblent souvent raides. Les courbes ont été tracées avec *gnuplot*. Voici $R(t)$ et $C(t)$ en fonction de t.



Voici le cycle limite obtenu à partir des courbes précédentes. La fonction $R(t)$ est en abscisse. La fonction $C(t)$ est en ordonnée.



8.5.1 Difficultés de l'intégration numérique

La simulation numérique précédente présente une difficulté : il faut résoudre à chaque pas une équation implicite en la variable A. Le système à intégrer est en effet de la forme

$$f_1(A,\, R) = 0, \quad \dot{C} = f_2(A,\, R,\, C), \quad \dot{R} = f_3(A,\, R,\, C)$$

où f_1, f_2 et f_3 sont des fonctions polynômes. Pour intégrer numériquement les équations, il faut pouvoir les évaluer or f_1 est implicite. Il faut alors « suivre » la racine réelle de f_1 initialement choisie à chaque modification de R. J'ai procédé de la façon suivante. Étant donné un couple de réels $(R,\, C)$, l'algorithme calcule toutes les valeurs réelles possibles de A en utilisant un algorithme d'isolation de racines réelles en une variable [22, 88]. La bonne racine est ici la plus grande des deux. Le triplet $(A,\, R,\, C)$ étant connu, il ne reste plus qu'à calculer $f_2(A,\, R,\, C)$ et $f_3(A,\, R,\, C)$.

Sur cet exemple, on dispose d'une autre façon de contourner la difficulté : il suffit de modifier légèrement le classement et de ranger les variables lentes de la façon $R > A > C$ au lieu de $A > R > C$. On obtient ainsi un système à intégrer numériquement de la forme

$$R = g_1(A), \quad \dot{C} = g_2(A,\, R,\, C), \quad \dot{A} = g_3(A,\, R,\, C)$$

où g_1, g_2 et g_3 sont des fonctions polynômes. Voici les détails obtenus avec *BLAD*.

```
$ ./differential_elimination
result of differential elimination = intersectof_regchain ([regchain ([2*R*A^2
 + 100*R*A + A^2 - 2450*A - 12500, C[t]*A + 50*C[t] + A^2 + A*C - 2450*A + 50*C
 - 12500, 12500*A[t]*A^3 + 1375000*A[t]*A^2 + 15625000*A[t]*A + 312500000*A[t]
 + 10*A^6 + 10*A^5*C - 17999*A^5 + 2000*A^4*C - 3252200*A^4 + 125000*A^3*C -
129115000*A^3 + 2500000*A^2*C - 638875000*A^2 - 62500000*A, Mr*A + 100*Mr - 100
```

```
*A - 2, Dr*A + 100*Dr - 100, Da*A + 50*Da - 50, Ma*A + 50*Ma - 50*A - 250], [
differential, autoreduced, primitive, squarefree, coherent, normalized]),
regchain ([3700*C^2 - 2015100*C + 1842117993, 80282*A - 1850*C + 905185], [
differential, autoreduced, primitive, squarefree, coherent, normalized])], [
differential, autoreduced, primitive, squarefree, coherent, normalized])
```

Le résultat obtenu peut surprendre certains lecteurs. On lit en effet régulièrement que, pour approximer un système différentiel comme nous l'avons fait, « il suffit de poser que les dérivées des variables rapides sont nulles ». Ce type d'affirmation conduit à des contradictions. En effet, la variable A fait partie des variables rapides. Si on pose sa dérivée nulle, comment se fait–il qu'on trouve une équation $\dot{A} = g_3(A, R, C)$? Réponse : on ne pose pas la dérivée des variables rapides égale à zéro ; on simplifie les équations différentielles qui définissent les variables lentes sur la variété algébrique obtenue en posant nuls les membres droits des équations différentielles qui définissent les variables rapides !

Chapitre 9

Simplification de systèmes différentiels

Ce chapitre décrit l'algorithme *Rosenfeld–Gröbner*. On commence par traiter à la main deux exemples : un exemple différentiel ordinaire et un exemple aux dérivées partielles. On donne ensuite le schéma de principe de l'algorithme. Dans la dernière partie du chapitre, on détaille un peu l'implantation faite en *BLAD*. On décrit en particulier un mécanisme qui évite d'engendrer des paires critiques inutiles dans le cas des systèmes aux dérivées partielles et un mécanisme qui évite d'engendrer des discussions de cas inutiles lors du traitement de certains sous–problèmes purement algébriques. Dans ce domaine, des progrès supplémentaires significatifs peuvent certainement être encore apportés, en s'appuyant sur les résultats de Marc Moreno Maza [70].

L'algorithme *Rosenfeld–Gröbner* prend en entrée un système Σ de polynômes différentiels et un classement arbitraires. Il produit en sortie une famille finie de chaînes différentielles régulières $C_1, \ldots, C_t$. La famille produite en sortie est équivalente au système initial dans le sens où toute solution analytique (sur un ouvert $\mathscr{D}$ qu'on ne précise pas) de Σ est une solution analytique de l'un des idéaux différentiels $[C_i] : H_{C_i}^\infty$ représentés par les chaînes et réciproquement. Dit autrement en utilisant la correspondance entre idéaux et variétés définie par le théorème des zéros, la famille produite est équivalente au système initial dans le sens où :

$$\sqrt{[\Sigma]} = [C_1] : H_{C_1}^\infty \cap \cdots \cap [C_t] : H_{C_t}^\infty.$$

La décomposition calculée n'est pas minimale. Rendre minimale une telle décomposition est un problème ouvert pour les systèmes différentiels. Schématiquement, l'algorithme *Rosenfeld–Gröbner* comporte deux étapes : une étape différentielle puis une étape purement algébrique. L'étape différentielle consiste à représenter Σ par une famille de « systèmes différentiels réguliers » $A_i = 0,\ S_i \neq 0$ qui sont des systèmes qui ne sont pas encore des chaînes différentielles régulières mais auxquels les lemmes de Lazard et de Rosenfeld s'appliquent déjà. La seconde étape consiste à transformer chacun des systèmes différentiels réguliers en une famille finie de chaînes différentielles régulières.

La première étape applique une idée élémentaire : parmi les solutions de l'équation $a\,x +$

$b = 0$, il y a celles qui annulent a et celles qui n'annulent pas a. En d'autres termes

$$a\,x + b = 0 \qquad \Leftrightarrow \qquad \left[a = 0,\ b = 0 \quad \text{ou} \quad x = -\frac{b}{a},\ a \neq 0\right].$$

À cette idée s'ajoute un problème particulier à régler dans le cas des systèmes aux dérivées partielles : la « cohérence ».

Il semble qu'Abraham Seidenberg [95] ait été le premier à appliquer les méthodes décrites ci–dessus (scindages, résolution du problème de cohérence, correspondance entre idéaux et variétés grâce au théorème des zéros) aux systèmes de polynômes différentiels mais Seidenberg ne disposait pas en 1956 des méthodes (bases de Gröbner, chaînes régulières) permettant d'effectuer les traitements purement algébriques nécessaires sur les systèmes différentiels réguliers. Il se limitait aux purs classements d'élimination. Surtout, il ne cherchait pas à calculer une représentation du radical de l'idéal différentiel engendré par un système afin de pouvoir, par exemple, en étudier les solutions. Il cherchait un algorithme de principe, prenant en paramètre un système et un polynôme, décidant si le polynôme appartient au radical de l'idéal défini par le système, c'est–à–dire un algorithme à résultat booléen.

Avant lui, Joseph Fels Ritt a proposé [86] un algorithme pour représenter le radical d'un idéal différentiel présenté par une famille de polynômes sous la forme d'une intersections d'idéaux présentés par des ensembles caractéristiques (ou des chaînes différentielles régulières) définissant des idéaux différentiels premiers : Ritt ne dispose pas du théorème d'équidimensionnalité des idéaux définis par des chaînes régulières qui lui permettraient de considérer le cas général. De plus, son algorithme ne s'applique qu'aux systèmes différentiels ordinaires (Ritt ne dispose pas non plus des théorèmes de Seidenberg et de Rosenfeld) et procède à des factorisations complètes sur des tours d'extensions algébriques du corps des coefficients des équations. Cette opération est algorithmique [99] mais très coûteuse.

Après Seidenberg, Ellis Robert Kolchin a généralisé [56] les méthodes de Ritt mais y a introduit des étapes non algorithmiques. On ne peut donc plus vraiment parler d'algorithme.

Les idées de Seidenberg ont été reprises ensuite par de très nombreux auteurs, en particulier par Wu Wen–Tsün [100] dans le contexte différentiel ordinaire mais sans faire la correspondance entre les variétés et les idéaux et surtout sans procéder à la simplification purement algébrique finale des systèmes produits à la fin de l'étape différentielle. Au début des années 1990, Elizabeth Mansfield a réalisé [67] un algorithme de simplification de systèmes différentiels polynomiaux qui s'applique aux systèmes aux dérivées partielles. Elle a utilisé une implantation en MAPLE de son logiciel *diffgrob* pour traiter de nombreux exemples et a étendu les fonctionnalités de son paquetage en dehors du cadre de l'algèbre différentielle (son paquetage permet de traiter des fonctions composées par exemple). Les bases théoriques de *diffgrob* ne sont pas aussi rigoureuses que celles de *Rosenfeld–Gröbner*. Un autre algorithme de simplification de systèmes différentiels a été proposé [84] par Gregory J. Reid, Allan D. Wittkopf et Alan Boulton. Voir aussi l'article [83] de Gregory J. Reid, P. Lin et Allan Wittkopf pour une application plus spécifique aux problèmes différentiels–algébriques. L'algorithme *rif* présente la particularité de séparer les systèmes différentiels, d'une part en une partie différentielle explicite (les dérivées dominantes des polynômes apparaissent linéairement), d'autre part en une variété différentiable sur laquelle les équations

différentielles sont définies. Dans le cas de systèmes polynomiaux, la variété différentiable est présentée via une variété algébrique formée de polynômes différentiels d'ordre zéro. La théorie de *rif* a été conçue pour le contexte des systèmes différentiels analytiques qui est plus général que le contexte des systèmes polynomiaux. Dans ce contexte, on ne dispose d'aucun analogue satisfaisant du lemme de Rosenfeld.

Dans ma thèse [7], j'ai repris l'idée de Seidenberg et en quelque sorte terminé le travail amorcé en montrant d'une part qu'on pouvait utiliser l'algorithme de bases de Gröbner pour effectuer la simplification purement algébrique finale et d'autre part, dans le cadre des systèmes aux dérivées partielles, comment le lemme de Rosenfeld [87] pouvait être appliqué. Ce résultat, augmenté du lemme de Lazard, a ensuite été publié par Daniel Lazard, Michel Petitot, François Ollivier et moi dans [9].

En 1995–1996, j'ai conçu et réalisé la première version du paquetage *diffalg* de MAPLE lors de mon stage post–doctoral au *Symbolic Computation Group* de l'université de Waterloo (Ontario, Canada). La première implantation de *Rosenfeld–Gröbner* qui y figure est assez sophistiquée. Elle est décrite dans l'article [10]. Évelyne Hubert, Allan D. Wittkopf et François Lemaire ont amélioré *diffalg* et l'ont interfacé avec les autres solveurs d'équations différentielles de MAPLE durant les années qui ont suivi.

Entre 1995 et 2000, un important travail a été effectué sur la théorie et l'algorithmique de la seconde partie de l'algorithme. C'est Évelyne Hubert qui la première a montré [49] que cette seconde partie est bien un problème purement algébrique. François Lemaire et moi avons montré [12] comment le principe de l'algorithme *lexTriangular* pouvait être appliqué pour éviter tout recours aux bases de Gröbner. L'algorithme *reg_ characteristic* ainsi obtenu est décrit dans le chapitre 6.

De nombreuses variantes de *Rosenfeld–Gröbner* ont été publiées. Ziming Li et Dongming Wang ont présenté dans [63] une méthode très proche fondée sur les méthodes de calcul d'ensembles triangulaires du deuxième auteur. Une autre méthode, fondée sur de nombreux calculs de bases de Gröbner, a été développée par l'équipe d'Abdelillah Kandri Rody dans la thèse d'Hamid Maârouf [66] et l'article [16] de Driss Bouziane, Abdelillah Kandri Rody et Hamid Maârouf.

9.1 *Rosenfeld–Gröbner* sur deux exemples

Dans cette section, on traite à la main deux exemples pour illustrer les grands principes de l'algorithme.

Définition 15 *(système différentiel régulier)*
Un système différentiel régulier *est un système $A = 0,\ S \neq 0$ d'équations et d'inéquations différentielles polynomiales d'un anneau de polynômes différentiels R tel que*

1. *A est partiellement autoréduit et triangulaire,*
2. *S contient les initiaux et les séparants des éléments de A et ne comporte que des polynômes différentiels partiellement réduits par rapport à A,*
3. *A est « cohérent ».*

La définition de la cohérence est donnée section 7.3, page 117. Les systèmes différentiels réguliers sont des systèmes auxquels les théorèmes décrits au chapitre 7 s'appliquent. On peut résumer leurs propriétés les plus utiles par le théorème suivant.

Théorème 12 *Soient $A = 0$, $S \neq 0$ un système différentiel régulier et p un polynôme différentiel.*

1. *L'idéal $(A) : S^\infty$ est radical.*
2. *L'idéal différentiel $[A] : S^\infty$ est radical.*
3. *Le polynôme p appartient à $[A] : S^\infty$ si et seulement si le reste partiel de p par A pour la réduction de Ritt appartient à $(A) : S^\infty$.*
4. *Le polynôme p s'annule sur toute solution du système si et seulement si p appartient à $[A] : S^\infty$.*

9.1.1 Un exemple différentiel ordinaire

La condition de cohérence est trivialement vérifiée par de tels systèmes. On considère le système suivant de l'anneau de polynômes différentiels $R = \mathbb{Q}\{u, v\}$ muni d'une seule dérivation δ_x.

$$(\Sigma_1) \qquad u_{xx} + v = 0, \quad u_x^2 + v = 0.$$

On choisit un classement $u \gg v$ qui élimine u et ses dérivées. Il n'y a qu'un seul tel classement. Les dérivées dominantes des équations sont respectivement u_{xx} et u_x. La première équation n'est pas partiellement réduite par rapport à la seconde puisqu'elle comporte une dérivée de la dérivée dominante de la seconde. Pour la réduire, on applique l'algorithme de réduction de Ritt, ce qui revient en pratique à procéder ainsi : on dérive une fois la deuxième équation

$$2\,u_x\,u_{xx} + v_x = 0$$

et on remplace u_{xx} par $-v_x/(2\,u_x)$ dans la première, ce qui donne

$$-\frac{v_x}{2\,u_x} + v = 0.$$

On remplace ensuite la première équation par l'équation réduite ou, plus précisément, par son numérateur puisqu'on s'intéresse aux solutions du système. On pose que $u_x \neq 0$ et on considère séparément les solutions de (Σ_1) qui annulent u_x. On obtient un « scindage[1] » de (Σ_1) en deux systèmes

$$(\Sigma_2) \qquad u_{xx} + v = 0, \quad u_x^2 + v = 0, \quad u_x = 0$$

[1] On a déjà rencontré des scindages au chapitre 5 mais il s'agissait de scindages de complètement différents de ceux pratiqués ci–dessus. Dans sa thèse [69], Marc Moreno Maza qualifie les scindages pratiqués ci–dessus de « scindages de première espèce » et ceux décrits au chapitre 5 de « scindages de deuxième espèce ». Les scindages de première espèce sont des scindages qui permettent de séparer les solutions qui annulent un initial ou un séparant de celles qui ne l'annulent pas. Les scindages de deuxième espèce (beaucoup plus rares et plus intéressants) surviennent lorsqu'on découvre qu'une équation se factorise.

et

$$(\Sigma_3) \qquad 2\,v\,u_x - v_x = 0, \quad u_x^2 + v = 0, \quad u_x \neq 0.$$

Considérons (Σ_2). En reportant la troisième équation dans la deuxième et sa dérivée première dans la première équation, on obtient $v = 0$. Ce système se simplifie donc en un système différentiel régulier

$$(\Sigma_4) \qquad u_x = 0, \quad v = 0$$

dont les solutions sont $u(x) = c$ et $v(x) = 0$ où c désigne une constante arbitraire. Le système (Σ_4) est une chaîne différentielle régulière. Considérons le système (Σ_3) laissé en suspens. Les deux premières équations ont même dérivée dominante. Le système n'est donc pas triangulaire. Pour le rendre triangulaire, on applique l'algorithme de réduction de Ritt, ce qui revient à procéder comme suit : on remplace u_x par $v_x/(2\,v)$ dans la deuxième équation, ce qui donne :

$$\left(\frac{v_x}{2\,v}\right)^2 + v = 0.$$

Comme précédemment, on remplace la deuxième équation par le numérateur de l'équation réduite sous réserve que $v \neq 0$ et on considère séparément les solutions de (Σ_3) qui annulent v. On obtient un scindage de (Σ_3) en deux systèmes

$$(\Sigma_5) \qquad 2\,v\,u_x - v_x = 0, \quad u_x^2 + v = 0, \quad v = 0, \quad u_x \neq 0$$

et

$$(\Sigma_6) \qquad 2\,v\,u_x - v_x = 0, \quad v_x^2 + 4\,v^3 = 0, \quad u_x \neq 0, \quad v \neq 0.$$

Considérons le système (Σ_5). L'équation $v = 0$ réduit à zéro (par la réduction de Ritt) la première équation. Elle permet aussi de simplifier la deuxième équation. On obtient un système

$$(\Sigma_7) \qquad u_x^2 = 0, \quad v = 0, \quad u_x \neq 0.$$

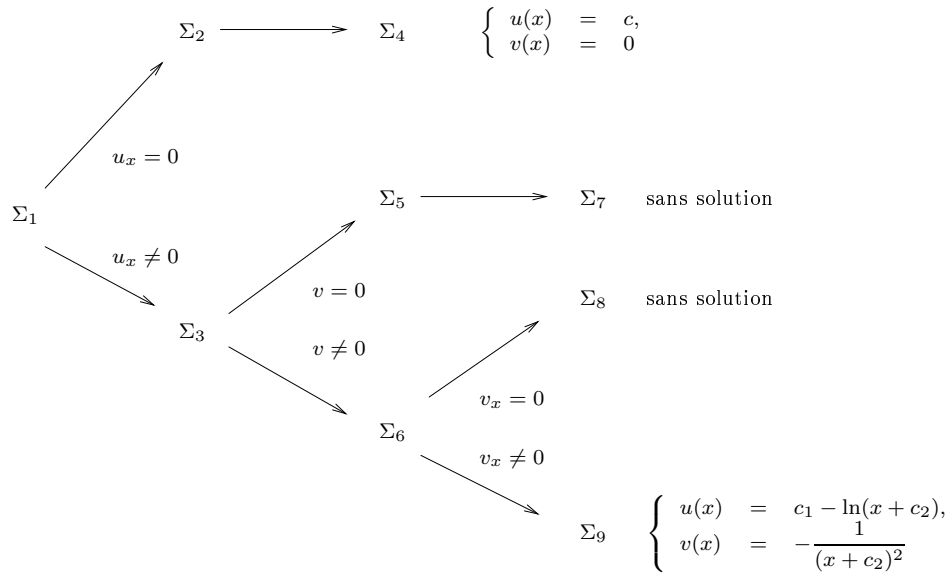
Le système (Σ_7) est un système différentiel régulier. L'algorithme *reg_ characteristic* peut lui être appliqué ce qui démontre, par un calcul de pgcd entre l'équation $u_x^2 = 0$ et l'inéquation $u_x \neq 0$, que le système n'a aucune solution. Revenons à (Σ_6). Ce système n'est pas encore un système différentiel régulier parce que le séparant $2\,v_x$ de la deuxième équation ne fait pas partie des inéquations. Il suffit de procéder à un scindage supplémentaire et de considérer séparément les solutions de (Σ_6) qui annulent v_x de celles qui ne l'annulent pas. On obtient deux systèmes

$$(\Sigma_8) \qquad 2\,v\,u_x - v_x = 0, \quad v_x^2 + 4\,v^3 = 0, \quad v_x = 0, \quad u_x \neq 0, \quad v \neq 0.$$

$$(\Sigma_9) \qquad 2\,v\,u_x - v_x = 0, \quad v_x^2 + 4\,v^3 = 0, \quad v_x \neq 0, \quad u_x \neq 0, \quad v \neq 0.$$

Des raisonnements similaires à ceux tenus pour (Σ_7) s'appliquent au système (Σ_8) qui n'a aucune solution. Le système (Σ_9) est un système différentiel régulier. L'ensemble de ses équations forme même une chaîne différentielle régulière. On peut donc supprimer l'inéquation $u_x \neq 0$ qui ne fait pas partie des initiaux et des séparants de la chaîne. Les solutions de

(Σ_9) sont $u(x) = c_1 - \ln(x + c_2)$ et $v(x) = -1/(x + c_2)^2$ où c_1 et c_2 sont deux constantes arbitraires. Voici résumé, l'arbre des scindages effectués ci–dessus.



Toute solution du système (Σ_1) est solution de (Σ_4) ou de (Σ_9). Réciproquement, les solutions de (Σ_4) et de (Σ_9) sont solutions de (Σ_1). En utilisant le théorème 12, page 142, on obtient

$$\sqrt{[u_{xx} + v,\, u_x^2 + v]} = [u_x,\, v] \cap [2\, v\, u_x - v_x,\; v_x^2 + 4\, v^3] : (v\, v_x)^\infty.$$

En utilisant les propriétés des chaînes différentielles régulières, on obtient aussi : un polynôme différentiel p appartient au radical de l'idéal différentiel engendré par (Σ_1) si et seulement si il est réduit à zéro pour la réduction de Ritt, à la fois par (Σ_4) et par (Σ_9). C'est le cas par exemple du polynôme $v_{xx} + 6\, v^2$.

9.1.2 Un exemple aux dérivées partielles

Considérons le système $\{f_1,\, f_2,\, f_3\}$ suivant de l'anneau de polynômes différentiels $\mathbb{Q}\{u, v\}$ muni de deux dérivations δ_x et δ_y.

$$(\Sigma_1) \qquad u_y^2 - 4\, u = 0, \quad u_x - v_x\, u = 0, \quad v_y = 0.$$

On choisit le classement compatible avec l'ordre total et de Riquier suivant :

$$\cdots > u_{xx} > u_{xy} > u_{yy} > v_{xx} > v_{xy} > v_{yy} > u_x > u_y > v_x > v_y > u > v.$$

Les dérivées dominantes des trois équations sont donc respectivement u_y, u_x et v_y. Le système est triangulaire et partiellement autoréduit. Est–il cohérent ? Les deux premières équations forment une paire critique $\{f_1,\, f_2\}$. Pour former le Δ–polynôme, on calcule la dérivée par δ_x de la première équation

$$\delta_x\, f_1 = 2\, u_y\, u_{xy} - 4\, u_x,$$

et la dérivée par δ_y de la seconde, qu'on multiplie par le séparant $2\,u_y$ de la première

$$2\,u_y\,\delta_y\,f_2 = 2\,u_y(u_{xy} - v_{xy}\,u - v_x\,u_y).$$

On obtient le Δ–polynôme par soustraction :

$$\Delta(f_1,\, f_2) = 2\,u\,u_y\,v_{xy} + 2\,u_y^2\,v_x - 4\,u_x.$$

Pour réduire ce Δ–polynôme par (Σ_1) pour la réduction de Ritt, il suffit, de calculer son pseudo–reste successivement par $\delta_x\,f_3$, f_1 et f_2. On obtient une quatrième équation $f_4 = u\,v_x = 0$ qu'on rajoute au système :

$$(\Sigma_2) \qquad u_y^2 - 4\,u = 0, \quad u_x - v_x\,u = 0, \quad v_y = 0, \quad u\,v_x = 0.$$

L'ajout de cette quatrième équation a pour effet de résoudre la paire critique $\{f_1,\, f_2\}$ et d'engendrer une nouvelle paire critique $\{f_3,\, f_4\}$. Avant de former le Δ–polynôme, on procède à un scindage sur l'initial de f_4. On considère séparément les solutions de (Σ_2) qui annulent u de celles qui ne l'annulent pas. On obtient deux systèmes :

$$(\Sigma_3) \qquad u_y^2 - 4\,u = 0, \quad u_x - v_x\,u = 0, \quad v_y = 0, \quad u\,v_x = 0, \quad u = 0$$

et

$$(\Sigma_4) \qquad u_y^2 - 4\,u = 0, \quad u_x = 0, \quad v_y = 0, \quad v_x = 0, \quad u \neq 0.$$

Le système (Σ_3) se simplifie considérablement en un système

$$(\Sigma_5) \qquad v_y = 0, \quad u = 0$$

qui constitue un système différentiel régulier et même une chaîne différentielle régulière. Ses solutions sont $u(x,\, y) = 0$ et $v(x,\, y) = \varphi(x)$ où $\varphi(x)$ désigne une fonction arbitraire de x.

Considérons le système (Σ_4). La paire critique $\{f_1,\, f_2\}$ est résolue. La paire critique $\{f_3,\, f_4\}$ l'est aussi puisque $\Delta(f_3,\, f_4) = 0$. Ce système est donc cohérent. Ce n'est pas encore un système différentiel régulier parce que le séparant u_y de f_1 ne fait pas partie des inéquations. On procède alors à un scindage de (Σ_4) en deux systèmes :

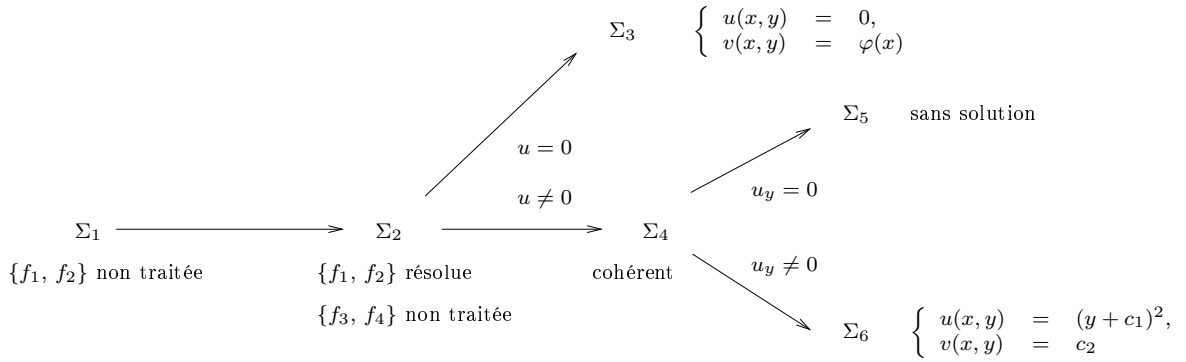
$$(\Sigma_5) \qquad u_y^2 - 4\,u = 0, \quad u_x = 0, \quad v_y = 0, \quad v_x = 0, \quad u_y = 0, \quad u \neq 0$$

et

$$(\Sigma_6) \qquad u_y^2 - 4\,u = 0, \quad u_x = 0, \quad v_y = 0, \quad v_x = 0, \quad u_y \neq 0, \quad u \neq 0.$$

Le système (Σ_5) n'a aucune solution : la nouvelle équation $u_y = 0$ permet de simplifier la première et d'obtenir $u = 0$, ce qui est incompatible avec l'inéquation $u \neq 0$. Le système (Σ_6) est un système différentiel régulier. L'ensemble de ses équations forme même une chaîne différentielle régulière. L'algorithme *reg_ characteristic* permet de prouver que l'inéquation $u \neq 0$, qui ne fait pas partie des initiaux et des séparants de la chaîne, est régulière modulo l'idéal défini par la chaîne. On la supprime donc. Les solutions de (Σ_6) sont $u(x,\, y) = (y+c_1)^2$

et $v(x,\,y) = c_2$ où c_1 et c_2 sont deux constantes arbitraires. Voici résumé l'arbre des scindages effectués ci–dessus.



Comme dans le cas différentiel ordinaire, toute solution de (Σ_1) est solution de (Σ_3) ou de (Σ_6). Réciproquement, toute solution de (Σ_3) ou de (Σ_6) est solution de (Σ_1). En utilisant le théorème 12, page 142, on obtient :

$$\sqrt{[u_y^2 - 4\,u,\, u_x - v_x\,u,\, v_y]} = [u,\, v_y] \cap [u_y^2 - 4\,u,\, u_x,\, v_y,\, v_x] : (u_y)^\infty.$$

En appliquant les propriétés des chaînes différentielles régulières, on obtient aussi : un polynôme différentiel p appartient au radical de l'idéal différentiel engendré par (Σ_1) si et seulement si il est réduit à zéro pour la réduction de Ritt à la fois par (Σ_3) et par (Σ_6). On voit que ni u ni v_x n'appartiennent à cet idéal alors que leur produit si. Le radical de l'idéal différentiel engendré par (Σ_1) n'est donc pas premier.

9.2 Le problème de l'inclusion des composantes

Soient A et B deux chaînes différentielles régulières. Il est très facile de décider si elles définissent le même idéal différentiel. Les deux chaînes doivent avoir le même rang et il suffit de tester que chacune des deux réduit tous les polynômes de l'autre à zéro par la réduction de Ritt. Il est même démontré dans [10] qu'en imposant quelques conditions sur les chaînes, il suffit de tester qu'elles sont syntaxiquement égales.

Mais même en supposant, pour simplifier, que les idéaux différentiels qu'elles définissent soient premiers, on ne connaît pas d'algorithme qui décide de l'inclusion entre les deux idéaux différentiels. On ne dispose que d'un critère : si A est réduit à zéro par B et si les initiaux et séparants de A sont réguliers modulo l'idéal différentiel $[B] : H_B^\infty$ alors on a $[A] : H_A^\infty \subset [B] : H_B^\infty$. Si par contre A est réduit à zéro par B et l'un des initiaux ou séparants de A est nul ou divise zéro modulo $[B] : H_B^\infty$ alors on ne connaît aucun moyen de conclure dans le cas général.

Un exemple illustre un peu la difficulté dans le cas d'une seule équation, où on dispose justement d'un algorithme de décision fondé sur le [56, Low Power Theorem] (voir aussi [49]). Bien que les deux polynômes différentiels soient très ressemblants, on a $[u_x^2 - 4\,u] : u_x^\infty \not\subset [u]$

mais $[u_x^2 - 4\,u^3] : u_x^\infty \subset [u]$. Dans le premier cas en effet, en dérivant une fois le polynôme différentiel $u_x^2 - 4\,u$ et en divisant par u_x, on trouve que $u_{xx} - 2 \in [u_x^2 - 4\,u] : u_x^\infty$ alors que $u_{xx} - 2 \notin [u]$. Une telle factorisation ne survient pas si on dérive une fois $u_x^2 - 4\,u^3$. Cela ne suffit bien sûr pas à prouver l'inclusion : une telle simplification pourrait s'appliquer à un polynôme différentiel obtenu en dérivant plus qu'une fois $u_x^2 - 4\,u^3$. Le [56, Low Power Theorem], qui permet de prouver que cela ne se produit pas est un théorème difficile, dont le preuve occupe plus de cinquante pages du livre d'Ellis Robert Kolchin.

9.3 Extension de la notion de paire critique

On étend pour les besoins des algorithmes la notion de paire critique à des situations non différentiellement triangulaires. Soit $\{p_1, p_2\}$ une « paire critique » c'est–à–dire une paire de polynômes dont les dérivées dominantes $\theta_1\,u$ et $\theta_2\,u$ sont des dérivées d'une même indéterminée différentielle. On note $\theta_{12}\,u$ la plus petite dérivée commune de $\theta_1\,u$ et de $\theta_2\,u$. Considérons d'abord le cas où $\theta_{12}\,u \neq \theta_1\,u$ et $\theta_{12}\,u \neq \theta_2\,u$. On retrouve la situation décrite en section 7.3, page 117. Ces paires critiques–là sont appelées des « paires normales ». On définit le Δ–polynôme engendré par la paire critique par la formule suivante, où s_1 et s_2 désignent les séparants de p_1 et de p_2 et où θ_{12}/θ_i désigne l'opérateur de dérivation $\varphi_i \in \Theta$ tel que $\theta_{12} = \theta_i\,\varphi_i$:

$$\Delta(p_1,\,p_2) \underset{\text{def}}{=} s_1\,\frac{\theta_{12}}{\theta_2}\,p_2 - s_2\,\frac{\theta_{12}}{\theta_1}\,p_1.$$

Considérons maintenant le cas où l'une des deux dérivées dominantes est une dérivée de l'autre. Mettons que le rang de p_2 soit supérieur à celui de p_1 (le cas de l'égalité ne se produit pas). Le Δ–polynôme engendré par la paire critique est alors défini par la formule suivante :

$$\Delta(p_1,\,p_2) \underset{\text{def}}{=} \text{prem}\left(p_2,\,\frac{\theta_2}{\theta_1}\,p_1\right).$$

Le calcul du Δ–polynôme correspondant à une réduction, on appelle de telles paires critiques des « paires de réduction ».

9.4 Pseudo–code de *Rosenfeld–Gröbner*

La fonction suivante reçoit en paramètre un système d'équations et d'inéquations polynomiales différentielles $A_0 = 0,\ S_0 \neq 0$ ainsi qu'un classement $\mathscr{O}$. Elle retourne une liste de chaînes différentielles régulières $C_1, \ldots, C_t$ telle que

$$\sqrt{[A_0] : S_0^\infty} = [C_1] : H_{C_1}^\infty \cap \cdots \cap [C_t] : H_{C_t}^\infty.$$

Un système en cours de traitement est représenté par un « quadruplet » $\langle A,\,D,\,P,\,S\rangle$ où A est l'ensemble des équations « déjà traitées », D est l'ensemble des paires critiques à traiter, P est l'ensemble des équations à traiter et S est l'ensemble des inéquations.

La fonction s'appuie sur deux sous–fonctions : *reg_ characteristic* déjà décrite au chapitre 6 et *compléter* décrite juste après.

Les scindages sont gérés au moyen d'une liste *AFaire* de quadruplets à traiter et une liste *Fait* de chaînes différentielles régulières déjà calculées. Les deux rajouts de quadruplets à la liste *AFaire* effectués juste avant l'appel à *compléter* correspondent à deux scindages. Le premier correspond au cas où l'initial du polynôme p est nul. Le second correspond au cas où l'initial est non nul et le séparant est nul. L'appel à *compléter* traite le cas où ni l'initial ni le séparant ne sont nuls.

L'expression « extraire un élément e d'une liste L » signifie qu'on affecte à e l'un des éléments de L (par exemple le premier) puis qu'on retire cet élément de la liste.

fonction *Rosenfeld–Gröbner* $(A_0, S_0, \mathscr{O})$
début
  AFaire $:= [\langle \emptyset, \emptyset, A_0, S_0 \rangle]$
  Fait $:= []$
  tant que *AFaire* $\neq []$ faire
    Extraire un quadruplet $\langle A, D, P, S \rangle$ de *AFaire*
    si $D = P = \emptyset$ alors
      Réduire partiellement les éléments de A les uns par rapport aux autres
      si cette opération n'a pas modifié le rang de A alors
        Réduire partiellement les éléments de S par rapport à A
        Rajouter à *Fait* les chaînes différentielles régulières obtenues
        en appliquant *reg_ characteristic* au système $A = 0,\ S \neq 0$
      fin si
    sinon
      si $P \neq \emptyset$ alors
        Extraire un polynôme p de P
      sinon
        Extraire une paire critique $\{p_1, p_2\}$ de D
        $p := \Delta(p_1, p_2)$
      fin si
      $p :=$ reste_complet(p, A)
      si $p = 0$ alors
        Rajouter le quadruplet $\langle A, D, P, S \rangle$ à la liste *AFaire*
      sinon
        Soient v^d, i et s le rang, l'initial et le séparant de p
        $q_i := p - i\, v^d$
        $q_s := d\, p - v\, s$
        Rajouter $\langle A, D, P \cup \{i, q_i\}, S \rangle$ à la liste *AFaire*
        Rajouter $\langle A, D, P \cup \{s, q_s\}, S \cup \{i\} \rangle$ à la liste *AFaire*
        Rajouter *compléter* $(\langle A, D, P, S \rangle, p)$ à la liste *AFaire*
      fin si
    fin si

fait
retourner *Fait*
fin

La fonction *compléter* insère la nouvelle équation dans la liste A des équations déjà traitées. Comme on souhaite maintenir cette liste différentiellement triangulaire, on en retire tous les polynômes p_i dont la dérivée dominante est une dérivée de la dérivée dominante de p. Ces équations supprimées de A ne sont pas perdues : on les retrouve dans la liste D des paires critiques à l'intérieur de paires de réduction.

fonction *compléter* $(\langle A, D, P, S\rangle, p)$
début
$\bar{A} :=$ $\{p\}$ union l'ensemble des éléments de A dont la dérivée dominante n'est pas une dérivée de la dérivée dominante de p
$\bar{D} :=$ D union l'ensemble de toutes les paires critiques qu'il est possible de former entre p et un élément de A
$\bar{P} := P$
$\bar{S} :=$ S union l'ensemble formé de l'initial et du séparant de p
retourner $\langle \bar{A}, \bar{D}, \bar{P}, \bar{S}\rangle$
fin

Voici quelques propriétés maintenues invariantes pour tous les quadruplets manipulés par l'algorithme :

1. A est différentiellement triangulaire ;
2. S contient les initiaux et les séparants des éléments de A ;
3. les paires critiques de D sont soit des paires de réduction soit des paires normales.

À tout quadruplet on peut associer l'idéal différentiel :

$$\sqrt{[A \cup \Delta(D) \cup P]} : S^\infty$$

où $\Delta(D)$ désigne l'ensemble des Δ–polynômes engendrés par les paires critiques de D (on peut en fait se restreindre aux Δ–polynômes engendrés par des paires de réduction, nous y reviendrons ultérieurement). Voici le principal invariant de boucle de l'algorithme : l'idéal $\sqrt{[A_0] : S_0^\infty}$ est l'intersection des idéaux différentiels définis par les quadruplets de la liste *AFaire* et des idéaux différentiels définis par les chaînes différentielles régulières de la liste *Fait*.

Proposition 42 *La fonction* Rosenfeld–Gröbner *termine quelles que soient les valeurs de ses paramètres.*

Preuve La fonction construit un arbre de scindages dont les nœuds sont des quadruplets. Il suffit de montrer que chaque branche est de longueur finie. Considérons une branche quelconque. Les quadruplets qui y figurent ont été engendrés soit par un appel à la fonction

compléter soit par un scindage sur l'initial ou le séparant du polynôme p soit suite à la réduction à zéro de p.

Il n'y a que la fonction *compléter* qui modifie la liste A. Elle insère dans cette liste différentiellement triangulaire un polynôme complètement réduit par rapport à A. Les arguments développés en section 2.4, page 37 montrent qu'elle ne peut être appelée qu'un nombre fini de fois. Cette fonction est aussi la seule à rajouter des paires critiques dans la liste D. Le long de cette branche, l'ensemble de toutes les paires critiques stockées au moins une fois dans D est donc fini et seul un nombre fini de Δ–polynômes est calculé. Pour simplifier la suite de la démonstration, on peut donc tricher un peu, supposer que D n'existe pas et que tous ces Δ–polynômes sont présents depuis le début des calculs dans la liste P.

Les deux autres opérations susceptibles d'engendrer un quadruplet consistent à extraire un polynôme de la liste P puis à le remplacer par au plus deux polynômes de rang plus petit que lui. Ces deux opérations–là aussi ne peuvent être effectuées qu'un nombre fini de fois.

Toutes les branches de l'arbre de scindages sont donc de longueur finie et l'algorithme s'arrête dans tous les cas. □

9.5 Optimisation du mécanisme de complétion

On cherche à éviter au maximum d'engendrer des paires critiques dont les Δ–polynômes se réduisent à zéro. Dans le contexte des bases de Gröbner, on dispose des « critères de Buchberger » [17, 43] et plus récemment du résultat quasi–optimal de Jean–Charles Faugère [40] dans un contexte restreint. J'ai publié avec Daniel Lazard, François Ollivier et Michel Petitot dans [10] un analogue des critères de Buchberger dans le contexte différentiel mais je ne trouve pas ces analogues très satisfaisants : ils sont compliqués à prouver et sont plus limités que les critères de Buchberger. Plus récemment, j'ai mis au point dans [5] un nouveau critère beaucoup plus simple, complémentaire de ceux publiés dans [10]. En fait, la version abstraite de ce nouveau critère est très connue. La seule difficulté consiste à bien comprendre les rôles respectifs des paires critiques normales et des paires de réduction dans un algorithme tel que l'algorithme précédent.

Considérons un appel *compléter* $(\langle A,\, D,\, P,\, S\rangle, p)$ et intéressons–nous aux nouvelles paires critiques insérées dans D. Après l'appel, les paires critiques normales de D contiennent des éléments qui sont encore présents dans A alors que les paires de réduction de D sont de la forme $\{p_i,\, p\}$ où p_i est un polynôme qui est présent dans A au moment de l'appel mais qui ne l'est plus à la fin. On voit donc qu'à la fin de l'appel, si on supprime une paire normale de D, l'idéal différentiel défini par le quadruplet n'est pas modifié. Par contre, si on supprime une paire de réduction de D, l'idéal différentiel défini par le quadruplet est modifié : on perd un des générateurs de l'idéal.

Maintenant, quelles paires critiques normales peut–on se permettre de supprimer ? Ces paires critiques–là sont uniquement destinées à assurer la cohérence de l'ensemble A. Dès qu'un polynôme différentiel est extrait de A, on peut donc supprimer toute paire critique normale dans laquelle il figure.

Proposition 43 *(critère pour supprimer des paires critiques)*

On peut supprimer de D toute paire critique normale dont un élément au moins n'appartient pas à A.

9.6 Sous–problèmes algébriques

Un sous–problème algébrique survient dans *Rosenfeld–Gröbner* lorsque le nouveau polynôme p réduit par rapport à A a même dérivée dominante qu'un polynôme p_i de A.

Ce cas–là n'est pas différencié des autres dans le pseudo–code précédent. Une paire de réduction $\{p_i, p\}$ est créée. Le Δ–polynôme engendré par cette paire n'est autre que le pseudo–reste $\mathrm{prem}(p_i, p)$. En supposant que cette paire soit immédiatement traitée à l'itération suivante de *Rosenfeld–Gröbner*, on voit que l'algorithme commence à émuler une version naïve de l'algorithme d'Euclide. De plus, à chaque étape

1. deux scindages sont effectués : un sur l'initial et un sur le séparant du pseudo–reste courant ;
2. des paires critiques normales engendrées par le pseudo–reste courant et les autres polynômes de A sont stockées dans D (note : ces paires sont supprimées à l'étape suivante si on applique la proposition précédente).

On optimise les calculs en appliquant la même idée que dans *PARDI* : les racines communes de deux polynômes sont les racines de leur pgcd. Il suffit donc de remplacer le polynôme p_i de A par le pgcd de p_i et de p ou plus précisément par le dernier sous–résultant non nul de la suite des sous–résultants définie par les deux polynômes. Pour s'assurer que ce dernier sous–résultant est correct, il suffit de stocker les initiaux de tous les sous–résultants intermédiaires dans la liste S des inéquations. Il faut aussi, pour ne pas perdre de solutions, engendrer à chaque étape un quadruplet considérant le cas où l'initial du sous–résultant courant est nul mais on évite tous les scindages sur les séparants des sous–résultants. On évite aussi d'engendrer des paires critiques normales à chaque calcul de sous–résultant. Enfin, on peut appliquer une méthode sophistiquée [36, 65] pour calculer la suite.

9.7 Réduction de problèmes algébriques et complexité

Il existe deux façons de réduire un problème d'élimination algébrique en un problème d'élimination différentielle. La première façon consiste à regarder les indéterminées du système polynomial à simplifier comme des fonctions constantes plutôt que comme des nombres et à regarder les équations polynomiales comme des équations différentielles d'ordre zéro.

L'application d'un algorithme de décomposition en chaînes différentielles régulières sur un tel système produit les mêmes chaînes que celles que pourrait produire un algorithme de décomposition en chaînes régulières non différentiel.

La seconde façon consiste à coder le système polynomial initial en un système aux dérivées partielles linéaires, en une indéterminée différentielle et à coefficients constants. On illustre le codage sur un exemple : le système S_1 suivant de $\mathbb{Q}[x, y, z]$

$$x^2\,y + 3\,z - 1 = 0, \quad 7\,x + y\,z - 2 = 0$$

se code en le système aux dérivées partielles linéaire S_2 suivant de $R = \mathbb{Q}\{u\}$ muni des dérivations δ_x, δ_y, δ_z

$$u_{xxy} + 3\,u_z - u = 0, \quad 7\,u_x + u_{yz} - 2\,u = 0.$$

Que se passe–t–il si on calcule une base de Gröbner réduite G_1 de l'idéal engendré par S_1 dans $\mathbb{Q}[x, y, z]$ pour un ordre admissible quelconque et si on applique le même procédé de codage sur les polynômes de G_1 ? On obtient une chaîne différentielle régulière C_1 complètement réduite pour le classement sur $\Theta\{u\}$ induit par l'ordre admissible. Cette chaîne est égale à celle qu'on aurait obtenue en appliquant *Rosenfeld–Gröbner* sur S_2 (on remarque que *Rosenfeld–Gröbner* ne produit aucun scindage si on l'applique à un système différentiel linéaire) or Ernst Wilhelm Mayr et Klaus Kühnle ont montré [57] le théorème suivant [104, Theorem 21.40].

Théorème 13 *Le problème du calcul d'une base de Gröbner réduite est expspace–complet.*

On en déduit la proposition suivante qui donne une idée de la complexité du résultat de l'algorithme *Rosenfeld–Gröbner* et non de l'algorithme lui–même, qui est nécessairement pire.

Proposition 44 *Le problème du calcul d'une chaîne différentielle régulière complètement réduite est expspace–dur.*

Comme le font remarquer Joachim von zur Gathen et Jürgen Gerhard, le résultat de Mayr et Kühnle est obtenu en considérant des systèmes de nature plus « combinatoire » que « géométrique » et on est en droit d'espérer que les problèmes différentiels « naturels » sont plus faciles à résoudre que les problèmes « combinatoires ».

Quatrième partie

Les bibliothèques BLAD

Chapitre 10

Conception des bibliothèques BLAD

« Comme souvent à ce stade, la réussite de l'idée dépend au moins autant de l'énergie consacrée à sa réalisation que de ses qualités initiales. »

Anonyme

La conception des bibliothèques *BLAD* remonte à peu près à 2000. J'ai réalisé deux versions en C++ puis deux versions en langage C. La seconde version en C correspond à la version actuelle. Elle fait approximativement quarante mille lignes de code, ce qui est peu, comparativement aux quatre vingt mille de la *Gnu Multiple Precision Library* ou des deux cent mille de la *Gnu Scientific Library*. J'ai développé les bibliothèques sur plusieurs architectures en particulier sur des machines disposant de processeurs INTEL, fonctionnant sous LINUX et sur une station SUN BLADE 100, disposant d'un processeur SPARC de 64 bits, fonctionnant sous SOLARIS 9.

10.1 Premiers choix

10.1.1 Éviter d'écrire un système de calcul formel

J'ai voulu éviter de me perdre dans la tentative d'écriture d'un système de calcul formel, qui est une tâche impossible pour une personne seule. Je suis donc parti d'un projet d'application utilisant l'élimination différentielle (celui décrit dans le chapitre 4) et je n'ai développé à peu de choses près que les outils nécessaires à sa réalisation.

J'ai été fortement impressionné par la conception de la *Gnu Multiple Precision Library* de Torbjörn Granlund. J'ai essayé de réaliser un analogue de cette bibliothèque (qui ne gère « que » les grands nombres) pour les systèmes de polynômes différentiels.

10.1.2 Le choix de la licence

J'ai longtemps hésité entre la *General Public License* (*GPL* en abrégé) et la *Lesser General Public License* (*LGPL* en abrégé). J'ai finalement choisi la seconde qui est moins coercitive. Les deux licences stipulent que les bibliothèques sont « *open source* ». Quelle est la différence

principale entre les deux ? La *GPL* est « contagieuse » alors que le *LGPL* ne l'est pas. En d'autres termes, seul un programme sous *GPL* peut utiliser une bibliothèque sous *GPL* alors que tout programme (même un programme non « *open source* ») peut utiliser une bibliothèque sous *LGPL*. La *Gnu Multiple Precision Library* est protégée par la *LGPL*. Elle est utilisée par le logiciel MAPLE qui n'est pas « *open source* ». La *Gnu Scientific Library* est protégée par la *GPL*. Nos logiciels développés dans le cadre du projet *LÉPISME* l'utilisent. Nous sommes donc obligés de les protéger par la *GPL*. Je rencontre déjà des difficultés à populariser les méthodes d'élimination différentielles. Je n'ai pas voulu en ajouter une supplémentaire en adoptant une licence trop contraignante.

10.1.3 Le choix du langage

J'ai beaucoup hésité sur le langage. Je voulais impérativement un langage assez largement répandu. Je cherchais aussi un langage qui me permette d'écrire une bibliothèque qui puisse être appelée depuis un programme principal écrit en C ou en FORTRAN. Voici quelques langages auxquels j'ai pensé :

ALDOR Un problème de diffusion. Ce langage semblait pourtant tout–à–fait adapté à mon projet. Mais il n'existait qu'une centaine de développeurs ALDOR au monde et un seul compilateur, maintenu par une seule équipe universitaire.

CAML J'ai craint aussi un problème de diffusion.

JAVA Un bon candidat, utilisé par ILOG pour la réalisation de ses solveurs il me semble. Mais, s'il n'est pas trop difficile d'utiliser une bibliothèque C à partir d'une application principale écrite en JAVA, l'inverse paraît plus difficile.

J'ai commencé par tenter des réalisations en C++. J'ai rencontré deux difficultés. La première était liée aux compilateurs : des programmes C++ qui me semblaient pourtant respecter la norme ANSI étaient acceptés par certains compilateurs mais pas par d'autres. Des optimiseurs de code avaient des comportement erratiques : des instructions de lancement d'exceptions (*throw*) n'avaient plus aucun effet lorsqu'on optimisait le code. Il y avait des erreurs dans le mécanisme d'instanciation des *templates* : le compilateur oubliait de générer le code correspondant à certaines fonctions, ce qui provoquait une erreur à l'édition des liens.

La deuxième difficulté était davantage une déception. Le langage C++ fournit des mécanismes en apparence très séduisants pour la réalisation de bibliothèques de calcul formel : héritage, généricité, possibilité de redéfinir les opérateurs arithmétiques pour améliorer la lisibilité du code, possibilité de redéfinir l'affectation et le constructeur de recopie pour implanter un « *garbage collector* » automatique. Ces mécanismes se sont avérés inadaptés.

Une première idée consiste à chercher une solution fondée sur le mécanisme d'héritage. On aboutit assez vite à des héritages multiples, où les structures algébriques (les anneaux de polynômes par exemple) sont implantées par des classes et où les éléments de ces structures (les polynômes) sont implantés par d'autres classes qui contiennent une référence vers la structure à laquelle ils sont mathématiquement censés appartenir. Le résultat est une implantation assez lourde et inefficace à l'exécution. Le code est très volumineux en raison de

tous les raffinements nécessaires à la bonne description des structures mathématiques. Voici deux exemples.

Pour implanter les polynômes en une indéterminée, on est conduit à distinguer ceux pour lesquels on dispose d'un algorithme de division euclidienne des autres. Une solution naturelle consiste à déclarer une classe A pour les polynômes généraux et une classe B qui hérite de A pour les polynômes qui disposent d'une division euclidienne. Ce type d'héritage où la classe B qui hérite ajoute des fonctionnalités mais pas de « champs » (c'est–à–dire des zones de stockage de données) à la classe A dont elle hérite est très fréquent en calcul formel. C'est même le seul type d'héritage dont on ait vraiment besoin dans ce domaine. Mais comme le mécanisme d'héritage C++ est conçu pour gérer le cas général, on est obligé d'écrire un nombre non négligeable de lignes de code pour expliquer que les instances de B doivent être gérées (affectation, constructeur de recopie qui ne sont pas hérités) exactement de la même façon que les instances de A. Tout ce code sans intérêt sur le fond dilue le code des fonctions où il se passe réellement quelque chose. Dans le cas de deux classes, ce n'est pas très gênant mais on peut vite aboutir pour les seuls polynômes en une indéterminée à quelques dizaines de classes.

Même l'anneau $\mathbb{Z}$ des entiers n'est pas simple : pour les applications en algèbre différentielle, il faut le concevoir dès le début comme un anneau *différentiel*, ce qui n'est pas vraiment naturel.

Le système de calcul formel AXIOM illustre parfaitement la lourdeur à laquelle on aboutit lorsqu'on pousse cette logique au bout. Et AXIOM s'appuie sur un langage de programmation spécialement conçu pour gérer ce genre de difficultés (langage dont ALDOR est issu), ce qui n'est pas le cas de C++.

L'idée qui vient ensuite consiste à utiliser le mécanisme des *templates* pour implanter du code générique. On aboutit à une solution plus naturelle avec un code plus rapide. Un gros avantage vient du fait qu'on n'est pas obligé de coder les fonctions qu'on n'utilise pas. Un gros inconvénient : le code qu'on écrit n'est pas compilé tant qu'il n'est pas appelé, ce qui fait qu'on peut être amené à corriger des erreurs de syntaxe dans des fonctions qu'on a écrites plusieurs mois auparavant mais qu'on n'a jamais eu l'occasion de compiler. Ces difficultés sont gérables pour du code simple (des listes génériques par exemple) mais, pour les polynômes en plusieurs indéterminées, on aboutit à quelques dizaines de milliers de lignes de *templates* très pénibles à manipuler.

La possibilité de redéfinir les opérateurs arithmétiques est assez peu utile. D'abord, quand les algorithmes commencent à se sophistiquer, ces opérations se diluent fortement dans le code. Ensuite, pour pouvoir vraiment manipuler les polynômes comme on manipule des *double* (des instructions telles que « `return (a + b)*c` » avec a, b, c polynômes) il faut implanter des mécanismes de *garbage collector* assez coûteux. J'avais implanté un mécanisme à base de compteurs de références qui étaient incrémentés et décrémentés par l'affectation, le constructeur de recopie et le destructeur. Par conséquent, chaque fois qu'un polynôme était passé en paramètre à une fonction, son compteur de référence était incrémenté au moment de l'appel et décrémenté à la fin. J'ai fini par trouver que c'était cher payer le luxe de pouvoir écrire des expressions telles que celle donnée ci–dessus.

Toutes ces réflexions conduisent à programmer en C++ de façon « ascétique » en se forçant à n'exploiter qu'un sous–ensemble assez réduit des mécanismes offerts par le langage. Lorsqu'on développe une bibliothèque sur plusieurs années, il est humainement très difficile de s'astreindre à une telle discipline. Tant qu'à faire, autant programmer en C.

Je ne suis pas le premier à raisonner ainsi. Jean–Charles Faugère et Fabrice Rouillier ont eux–aussi (avant moi) commencé à implanter en C++ leur chaîne logicielle *GB+RS* puis sont revenus au langage C pour des raisons très similaires.

10.2 Structure générale des bibliothèques

Les grands entiers sont gérés par la *Gnu Multiple Precision*. Il y a quatre bibliothèques, construites les unes au–dessus des autres.

ba0 Cette bibliothèque gère la mémoire, les entrées–sorties, les parseurs et le mécanisme des exceptions. Elle gère aussi quelques structures de données simples, dont les tableaux génériques.

bav Cette bibliothèque gère les variables, c'est–à–dire les indéterminées différentielles, les dérivations (variables indépendantes), les dérivées (variables dépendantes) et les classements. Elle gère aussi quelques structures de données simples comme les « termes » c'est–à–dire les produits de puissances de variables.

bap Cette bibliothèque gère les polynômes. Sa principale fonctionnalité est le pgcd de deux polynômes en plusieurs variables et à coefficients entiers. Ce calcul nécessite l'usage de polynômes à coefficients entiers modulaires (entiers machine et grands entiers). La bibliothèque fournit aussi quelques fonctionnalités sur les polynômes à coefficients rationnels et sur les fractions rationnelles.

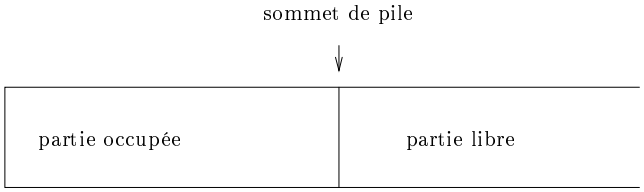
bad Cette bibliothèque gère les chaînes régulières, différentielles ou non. Elle implante *PARDI*, *Rosenfeld–Gröbner* et l'algorithme de forme normale.

Pour faciliter leur portabilité, les bibliothèques *BLAD* sont développées grâce au mécanisme *autoconf+automake+libtool*. Ce mécanisme permet aussi de maintenir facilement la documentation et des batteries de tests. J'utilise *cvs* pour gérer les différentes versions.

10.3 La gestion de la mémoire et le *garbage collector*

Le *garbage collector* n'est pas automatique. Il doit être intégré lors de l'écriture des algorithmes. Deux mécanismes sont fournis : un mécanisme à base de deux piles et le mécanisme inventé par Jean–Charles Faugère [38]. La mémoire est organisée logiquement en plusieurs piles (appelées *stacks*). D'un point de vue interne, chaque pile est une liste chaînée de cellules qui font typiquement quelques dizaines de megaoctets chacune. Ce découpage en cellules est transparent pour les fonctions qui réclament ou restituent de la mémoire. Dans chaque pile, on distingue la partie occupée de la pile de la partie libre. La frontière entre ces deux parties est indiquée par un « sommet de pile ». Ce mécanisme rend très rapides les allocations et les

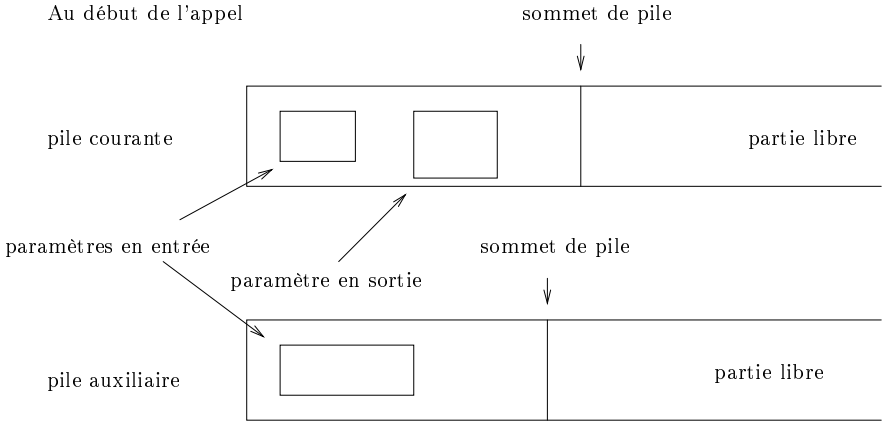
désallocations de mémoire dans la pile : il suffit de déplacer le sommet de pile. Il y a deux « piles d'usage courant » : la pile principale et la pile secondaire. L'appellation est trompeuse parce qu'elles jouent des rôles interchangeables. Il existe d'autres piles ayant des rôles très spécialisés qu'on décrit plus loin.



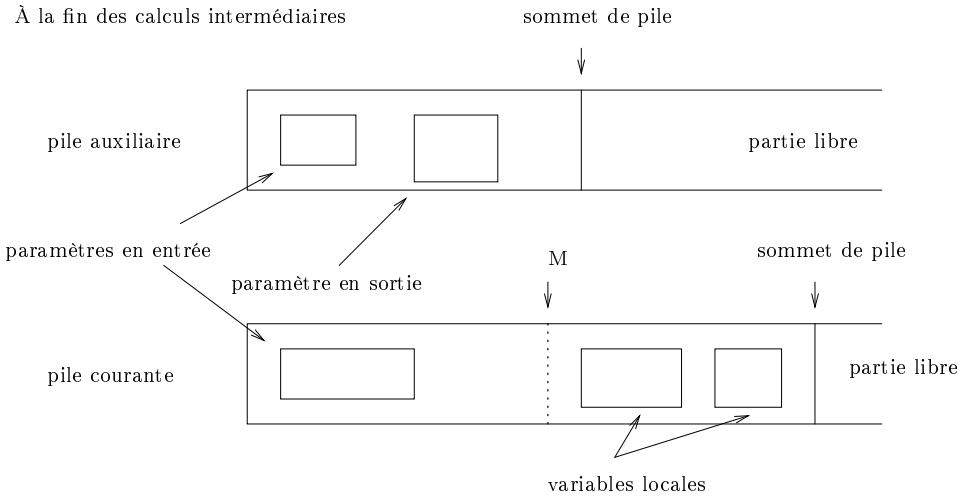
10.3.1 Le mécanisme à deux piles

Le schéma de programmation utilisé dans les bibliothèques *BLAD* est procédural : les procédures reçoivent en argument leurs données (paramètres en entrée) ainsi que les adresses des variables destinées à recevoir leur résultat (paramètres en sortie). À chaque appel à une procédure, l'une des deux piles d'usage courant est la « pile courante », l'autre est la « pile auxiliaire ». La pile courante est la pile dans laquelle se font les allocations mémoires. Les variables et les procédures de la bibliothèque respectent les conventions suivantes :

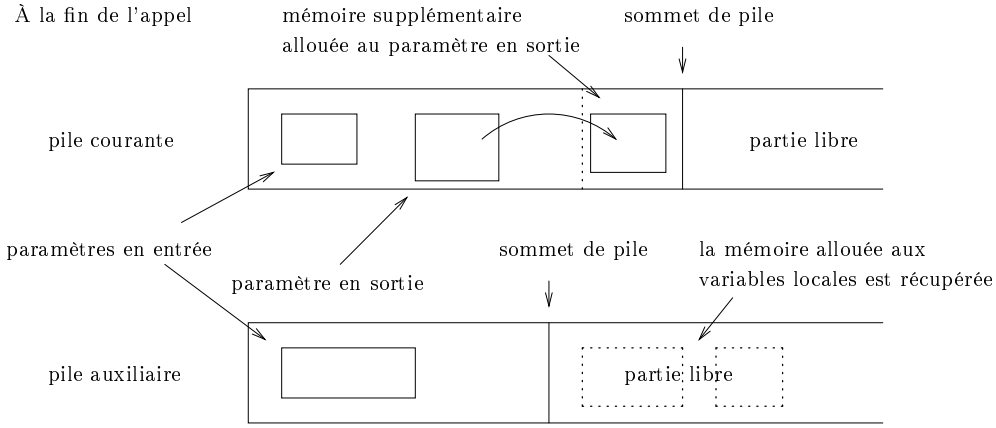
1. aucune structure de données n'est répartie sur les deux piles d'usage courant en même temps ;
2. tout paramètre en sortie d'une fonction ne peut pointer que sur des zones appartenant à la pile courante.



Typiquement, le corps de chaque fonction commence par une instruction qui échange les rôles des deux piles. L'adresse du sommet de la pile courante (l'ancienne pile auxiliaire) est mémorisée dans une variable locale M. De la mémoire est allouée dans la pile courante pour les variables locales à la fonction et les calculs intermédiaires peuvent commencer.



À la fin de ces calculs, une instruction échange à nouveau les rôles des deux piles : on se retrouve dans la situation qui prévalait au début de l'appel. Le résultat de la fonction est stocké dans les paramètres en sortie (cette affectation peut consommer de la place dans la pile courante). Le sommet de la pile auxiliaire est replacé à l'adresse mémorisée dans M, ce qui a pour effet de libérer toute la mémoire attribuée aux variables locales.



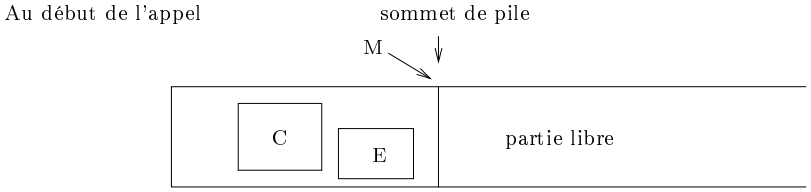
En réfléchissant un peu, on s'aperçoit que ce mécanisme fonctionne récursivement. Il est bien adapté au style de programmation fortement récursif qu'on rencontre dans les algorithmes dédiés aux chaînes régulières. Il présente l'inconvénient d'induire de nombreuses recopies d'objets. Cet inconvénient présente un avantage : les données sont stockées en mémoire dans l'ordre dans lequel elles sont lues, ce qui minimise les sauts de pages mémoire et optimise le fonctionnement du cache de l'ordinateur et de la zone de swap. Par comparaison, les performances de MAPLE, qui disperse complètement ses données en mémoire, s'effondrent totalement dès que la zone de swap est utilisée. Le programme C suivant qui calcule un produit vectoriel illustre le mécanisme à deux piles.

```
#include "ba0.h"

void wedge (ba0_tableof_int_p R, ba0_tableof_int_p A, ba0_tableof_int_p B)
```

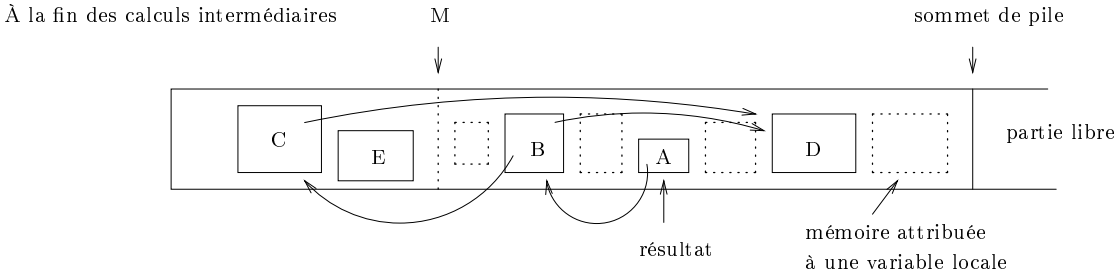
```
{   struct ba0_tableof_int_p T;
    struct ba0_mark M;
                                  // La variable résultat R pointe dans
                                  //    la pile courante
    ba0_push_another_stack ();    // échange les rôles des piles
    ba0_record (&M);              // mémorise le sommet de la pile
                                  //    courante (ancienne pile auxiliaire)
    ba0_init_table ((ba0_table)&T); // variable auxiliaire T allouée (on
                                  //     ne peut pas modifier R qui
                                  //     pourrait être égale à A).
    ba0_realloc_table ((ba0_table)&T, 3);
    T.size = 3;
    T.tab [0] = A->tab [1]*B->tab [2] - A->tab [2]*B->tab [1];
    T.tab [1] = - A->tab [0]*B->tab [2] + A->tab [2]*B->tab [0];
    T.tab [2] = A->tab [0]*B->tab [1] - A->tab [1]*B->tab [0];

    ba0_pull_stack ();            // rend aux deux piles le rôle
                                  //     qu'elles avaient lors de l'appel
                                  // affecte T à R
    ba0_set_table ((ba0_table)R, (ba0_table)&T);
    ba0_restore (&M);             // libère la mémoire allouée à T.
}

int main ()
{   ba0_tableof_int_p R, A, B;

    ba0_restart (0, 0);

    R = (ba0_tableof_int_p)ba0_new_table ();
    A = (ba0_tableof_int_p)ba0_new_table ();
    B = (ba0_tableof_int_p)ba0_new_table ();

    ba0_sscanf2 ("[2, 3, 4]", "%t[%d]", A);
    ba0_sscanf2 ("[3, 1, -2]", "%t[%d]", B);

    wedge (R, A, B);

    ba0_printf ("%t[%d] ^ %t[%d] = %t[%d]\n", A, B, R);
    ba0_terminate (ba0_init_level);
    return 0;
}
```

10.3.2 Le mécanisme de Faugère

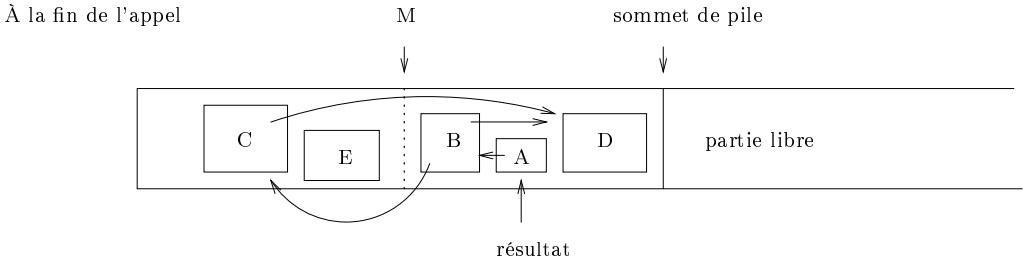
Ce mécanisme n'utilise qu'une seule pile. Il est bien adapté aux algorithmes fortement itératifs qui consomment relativement peu de mémoire à chaque itération. Au début d'un appel de fonction, une instruction mémorise le sommet de pile dans une variable locale M.



À chaque itération, la fonction consomme de la mémoire dans la pile. Entre M et le nouveau sommet de pile on trouve de la mémoire attribuée aux variables locales (qu'on veut récupérer) et de la mémoire attribuée pour stocker le résultat (qu'on veut préserver).



Lorsque le calcul se termine, la fonction passe en paramètre au *garbage collector* la valeur de M et l'adresse de la variable résultat. En général, le résultat est constitué par plusieurs petites zones mémoire qui pointent les unes sur les autres. Le *garbage collector* tasse le contenu de la variable résultat juste au–dessus de M en supprimant les « trous » qui existent entre les petites zones mémoires constitutives du résultat puis il déplace le sommet de pile en M augmenté de la taille du résultat. Les petites zones mémoires constitutives du résultat sont tassées en respectant l'ordre dans lequel elles sont rangées dans la pile. Les petites zones mémoires référencées plusieurs fois ne sont pas dupliquées. Les structures circulaires sont autorisées.



Ce *garbage collector* peut s'implanter par l'algorithme suivant.

1. Parcourir le résultat à partir de sa racine en enregistrant dans un tableau auxiliaire des structures composées de deux champs : l'adresse et la taille des petites zones mémoires rencontrées. Il suffit de stocker les adresses des zones qui sont rangées au–dessus de M.

À la fin de ce parcours, le nombre des petites zones mémoires est connu.

2. Affecter à un deuxième tableau auxiliaire les adresses des structures créées ci–dessus. Trier ce deuxième tableau par ordre croissant des adresses des petites zones mémoires.
3. Tasser les petites zones mémoires au–dessus de M. Parcourir pour cela le deuxième tableau auxiliaire. Enregistrer la nouvelle adresse des petites zones mémoires à la place de l'ancienne dans le premier tableau auxiliaire.
4. Mettre à jour les pointeurs internes des petites zones mémoires entre elles. Il suffit pour cela de parcourir le résultat à partir de sa racine exactement dans le même ordre qu'en 1. Les bonnes adresses s'obtiennent en parcourant séquentiellement le premier tableau auxiliaire.

On détecte facilement si une zone mémoire est référencée plusieurs fois. On évite ainsi facilement de dupliquer une telle zone. L'algorithme n'est pas gêné par les structures circulaires. Tasser les petites zones mémoires en respectant l'ordre dans lequel elles sont rangées dans la pile évite aussi un problème de type « sac à dos » qui pourrait survenir en raison du morcellement des piles en cellules.

Ce *garbage collector* peut être appelé à chaque itération. Pour éviter de gaspiller du temps de calcul, on peut alors implanter une variante qui teste si la pile est proche de la saturation ou pas et qui ne récupère la mémoire que dans le cas où la pile est proche de la saturation. Le programme C suivant montre comment utiliser l'implantation en *BLAD* du mécanisme de Faugère.

```
#include "ba0.h"

void wedge (ba0_tableof_int_p R, ba0_tableof_int_p A, ba0_tableof_int_p B)
{   struct ba0_tableof_int_p T;

    ba0_init_table ((ba0_table)&T);
    ba0_realloc_table ((ba0_table)&T, 3);
    T.size = 3;
    T.tab [0] = A->tab [1]*B->tab [2] - A->tab [2]*B->tab [1];
    T.tab [1] = - A->tab [0]*B->tab [2] + A->tab [2]*B->tab [0];
    T.tab [2] = A->tab [0]*B->tab [1] - A->tab [1]*B->tab [0];
    ba0_set_table ((ba0_table)R, (ba0_table)&T);
}

int main ()
{   struct ba0_mark M;
    ba0_tableof_int_p R, A, B;

    ba0_restart (0, 0);
    ba0_record (&M);                          // enregistre le sommet de pile
                                              //    dans M
    R = (ba0_tableof_int_p)ba0_new_table ();
```

```
    A = (ba0_tableof_int_p)ba0_new_table ();
    B = (ba0_tableof_int_p)ba0_new_table ();

    ba0_sscanf2 ("[2, 3, 4]", "%t[%d]", A);
    ba0_sscanf2 ("[3, 1, -2]", "%t[%d]", B);

    wedge (R, A, B);                          // la mémoire est gaspillée

    ba0_garbage ("%t[%d]", &M, &R);           // appel du garbage collector
                                              // Le contenu de R est tassé
                                              // au-dessus de M.
    ba0_printf ("result = %t[%d]\n", R);
    ba0_terminate (ba0_init_level);
    return 0;
}
```

10.3.3 Avantage du style procédural

C'est le style adopté par Torbjörn Granlund pour la *Gnu Multiple Precision Library*. Le fait que les procédures reçoivent en paramètre l'adresse des variables destinées à recevoir les résultats présente deux avantages sur un style fonctionnel.

Premier avantage : une procédure peut tester si la variable résultat dispose d'assez de mémoire pour recevoir le résultat et donc éviter d'allouer de la mémoire dans beaucoup de cas.

Deuxième avantage : une procédure peut tester si son paramètre en sortie est égal à l'un de ses paramètres en entrée et implanter un code optimisé pour ce cas. Cette situation est vraiment très fréquente : les instructions de la forme « `a = a + b` » sont beaucoup plus courantes que les instructions de la forme « `a = b + c` ».

10.3.4 Le mécanisme de gestion d'exceptions

Il est implanté par un mécanisme fondé sur les fonctions *setjmp* et *longjmp* de la bibliothèque standard du langage C. Une pile de points de traitement d'exceptions est implantée. La seule difficulté dans la gestion des exceptions réside dans le problème de récupérer la mémoire allouée aux variables locales entre le moment où le point de traitement d'exception est posé et le moment où l'exception est levée. C'est facile avec les mécanismes de gestion de la mémoire décrits ci–dessus : il suffit de mémoriser les valeurs des sommets des deux piles d'usage courant lors de la pose du point d'exception et de restaurer ces valeurs au moment où l'exception est levée. Ce mécanisme impose des restrictions sur le style de programmation qui ne se sont pas avérées très contraignantes à l'usage. Un mécanisme de levée d'exception avec valeur de retour récupérée au niveau du point de traitement d'exception est implanté aussi. Il utilise à la fois le mécanisme à deux piles et le mécanisme de Faugère pour préserver la valeur de retour au moment de la restauration des deux sommets de pile.

10.4 Les variables et les classements

Les bibliothèques *BLAD* permettent de manipuler dans le cadre d'une même « suite d'appels » un seul anneau de polynômes différentiels. Cet anneau peut être muni en même temps de plusieurs classements — et plus généralement d'ordres sur les dérivées qui ne sont pas des classements. Mathématiquement, les seuls anneaux de polynômes qu'il est possible de définir sont de la forme

$$\mathbb{Z}[x_1, \ldots, x_m]\{u_1, \ldots, u_n\}$$

où $x_1, \ldots, x_m$ désignent des variables indépendantes et $u_1, \ldots, u_n$ des indéterminées différentielles. L'ensemble des dérivations est l'ensemble des dérivées partielles par rapport aux m variables indépendantes.

Dans les bibliothèques *BLAD*, une « variable » est soit une variable indépendante soit une dérivée d'une des indéterminées différentielles. Elles sont codées par des structures de données qui ne sont pas dupliquées. Le test d'égalité entre deux variables est donc très rapide. Toutes les variables sont stockées dans une pile spéciale. Une variable créée lors d'une suite d'appels n'est pas détruite avant le fin de cette suite.

Plusieurs classements peuvent être définis en même temps. Les comparaisons de variables vis–à–vis des différents classements sont effectuées ainsi : chaque fois qu'un classement est créé, toutes les variables créées reçoivent un numéro d'ordre pour ce classement ; chaque fois qu'une variable est créée, tous les numéros de toutes les variables vis–à–vis de tous les classements sont recalculés. On compare les variables en comparant leur numéro. Ce mécanisme est un peu pénalisant au début de l'exécution d'un programme standard, c'est–à–dire au moment où la plupart des variables sont créées. Il permet d'accélérer les comparaisons de variables ensuite. J'ai fait le pari qu'on ne dépasserait jamais en pratique une ou deux centaines de variables et quelques classements mais si on code un calcul de bases de Gröbner en un calcul de chaîne différentielle régulière de telle sorte que chaque variable de *BLAD* code un terme du système polynomial (voir section 9.7), on risque d'obtenir de piètres performances.

Des ordres sur les variables plus généraux que les classements peuvent être définis. Ils sont utiles dans le cadre de certains algorithmes non différentiels où il est souhaitable de privilégier l'une des variables apparaissant dans un polynôme plutôt que les autres. C'est le cas de certains algorithmes de pgcd de polynômes en plusieurs variables où l'on évalue toutes les variables sauf une de façon à se ramener à un calcul de pgcd de polynômes en une variable et où l'on conclut par une remontée de Hensel. Pour que la remontée de Hensel (point pénalisant de l'algorithme) soit efficace, il faut que le coefficient dominant de la variable privilégiée soit aussi petit que possible. On définit de tels ordres en *BLAD* en partant d'un classement et en précisant les variables qui doivent être considérées comme supérieures à toutes les autres. Voici un exemple :

```
#include "bav.h"

int main ()
{
    struct ba0_mark M;
```

```
    bav_tableof_variable T;
    bav_Iordering r, s, t;

    bav_restart (0, 0);
    ba0_record (&M);
/*
  Les deux premiers ordres sont des classements, pas le dernier.
*/
    ba0_sscanf2
        ("ordering (derivations = [x,y], blocks = [u,v])", "%ordering", &r);
    ba0_sscanf2
        ("ordering (derivations = [x,y], blocks = [v,u])", "%ordering", &s);
    ba0_sscanf2
        ("ordering (derivations = [x,y], blocks = [v,u], varmax = [u[x]])",
         "%ordering", &t);
/*
   Le format "%t[%v]" indique une table de variables.
*/
    bav_R_push_ordering (r);
    T = (bav_tableof_variable)ba0_new_table ();
    ba0_sscanf2 ("[u[x], u[x,y], u, v, v[y], v[x,y]]", "%t[%v]", T);
    bav_sort_tableof_variable (T);
    ba0_printf ("w.r.t. %ordering\nsorted table = %t[%v]\n", r, T);
    bav_R_push_ordering (s);
    bav_sort_tableof_variable (T);
    ba0_printf ("w.r.t. %ordering\nsorted table = %t[%v]\n", s, T);
    bav_R_push_ordering (t);
    bav_sort_tableof_variable (T);
    ba0_printf ("w.r.t. %ordering\nsorted table = %t[%v]\n", t, T);

    bav_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu à l'exécution.

```
$ ./bav_variables
w.r.t. ordering (derivations = [x, y], blocks = [grlexA[u], grlexA[v]])
sorted table = [v, v[y], v[x,y], u, u[x], u[x,y]]
w.r.t. ordering (derivations = [x, y], blocks = [grlexA[v], grlexA[u]])
sorted table = [u, u[x], u[x,y], v, v[y], v[x,y]]
w.r.t. ordering (derivations = [x, y], blocks = [grlexA[v], grlexA[u]],
varmax = [u[x]])
sorted table = [u, u[x,y], v, v[y], v[x,y], u[x]]
```

10.5 Les polynômes

10.5.1 Choix de la représentation

La représentation que j'ai choisie est une variante de la représentation distribuée qui ne pénalise pas (trop) l'accès aux coefficients d'un polynôme vis–à–vis d'un sous–ensemble des variables dont il dépend. J'ai là–aussi beaucoup hésité. Voici des choix types (il y en a d'autres, que je n'ai pas considérés, comme des représentations par « programmes d'évaluation » ou « *straight lines programs* ») :

représentation distribuée. Un polynôme est donné par une liste de termes (c'est–à–dire de produits de puissances de variables) et une liste de coefficients numériques. Elle est avantageuse pour tous les algorithmes qui regardent les polynômes comme des vecteurs de coefficients (algorithme de Buchberger, algorithme *FGLM*, contenu numérique). Elle permet de compresser les termes des polynômes. Elle ne privilégie pas trop l'une des variables par rapport aux autres. Elle est pénalisante pour tous les algorithmes qui ont besoin d'accéder aux coefficients d'un polynôme vis–à–vis d'un sous–ensemble des variables dont il dépend (algorithmes sur les chaînes régulières).

représentation en arbre. C'est la représentation de MAPLE (sauf que MAPLE emploie des graphes acycliques plutôt que des arbres). Un polynôme est donné par un nœud représentant une opération (somme ou produit) et les opérandes sur lesquels elle agit, qui sont d'autres polynômes. Elle présente l'avantage d'être très souple d'emploi et de permettre de généraliser les polynômes à des expressions quelconques. Elle peut être très utile dans le cadre de certains algorithmes où elle permet de conserver les polynômes sous forme factorisée (algorithmes sur les chaînes régulières, implantation de fractions rationnelles, dont les dénominateurs sont souvent des produits). Elle rend inefficaces certains petits algorithmes fréquemment appelés (extraction de la dérivée dominante, du monôme de tête, et même reconnaissance de zéro).

représentation récursive. Un polynôme en plusieurs variables est représenté récursivement à partir de polynômes en une variable. Un polynôme est donc soit un coefficient numérique soit un polynôme en une variable privilégiée et à coefficients dans les polynômes en les variables restantes. Elle est avantageuse pour les algorithmes qui regardent les polynômes justement de cette façon–là (algorithmes sur les chaînes régulières). Elle est incompressible (elle est truffée de pointeurs et de coefficients numériques). Elle est pénalisante pour tous les algorithmes qui regardent les polynômes comme des vecteurs de coefficients. Elle privilégie beaucoup l'une des variables vis–à–vis des autres.

Les trois représentations ont des avantages et des inconvénients. Il est très difficile de savoir au vu des expériences des autres lesquelles sont les plus efficaces. En effet, une fois choisie une représentation, les développeurs de code en calcul formel adoptent rapidement un style de programmation qui privilégie la représentation choisie, ce qui biaise les comparaisons de performance. Ceux qui adoptent une représentation récursive ont tendance à programmer par récurrence sur la variable principale (Marc Moreno Maza en ALDOR), ceux qui adoptent une représentation distribuée ont tendance à programmer itérativement

en énumérant les termes (moi en *BLAD*) et plus d'une fonction MAPLE tire parti du fait que les données ne sont pas fortement structurées (variable privilégiée sélectionnée suivant des critères heuristiques).

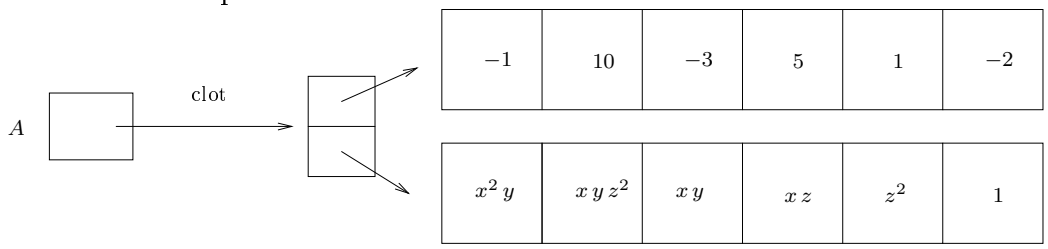
J'ai craint, en choisissant la représentation MAPLE, de retrouver certains défauts que j'avais rencontrés lors du développement de *diffalg*. J'avais déjà plusieurs fois testé la représentation récursive. Je voulais pouvoir compresser les polynômes et je craignais d'être pénalisé dans l'implantation du pgcd de deux polynômes en plusieurs variables par une représentation qui privilégierait trop l'une des variables — et pas forcément la bonne. J'ai donc choisi une variante de la représentation distribuée en essayant d'atténuer le coût des accès aux coefficients d'un polynôme vis–à–vis de sa variable principale.

10.5.2 Réalisation

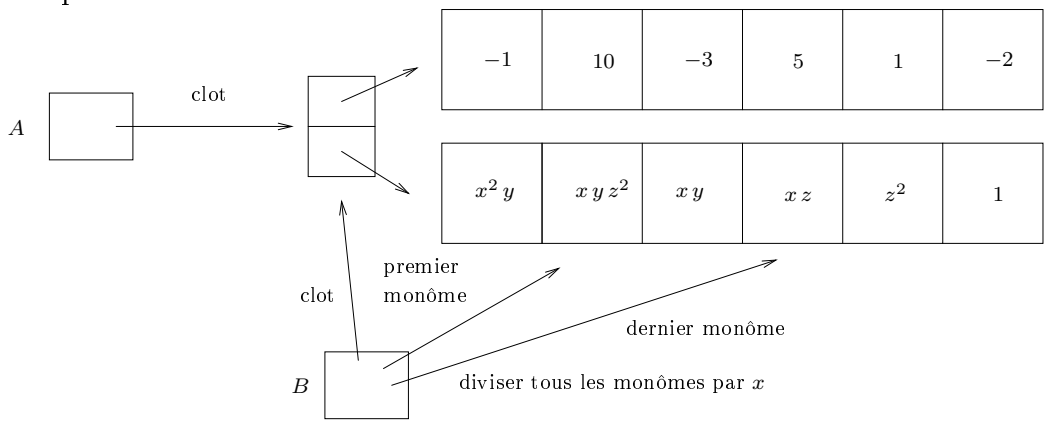
Chaque polynôme en *BLAD* est construit en deux couches. La couche du bas implante une combinaison linéaire ordonnée de termes (une « *clot* »). La couche du haut implante le polynôme proprement dit. Supposons que l'ordre sur les variables soit $x > y > z$ et prenons l'exemple du polynôme

$$A = -x^2\,y + 10\,x\,y\,z^2 - 3\,x\,y + 5\,x\,z + z^2 - 2.$$

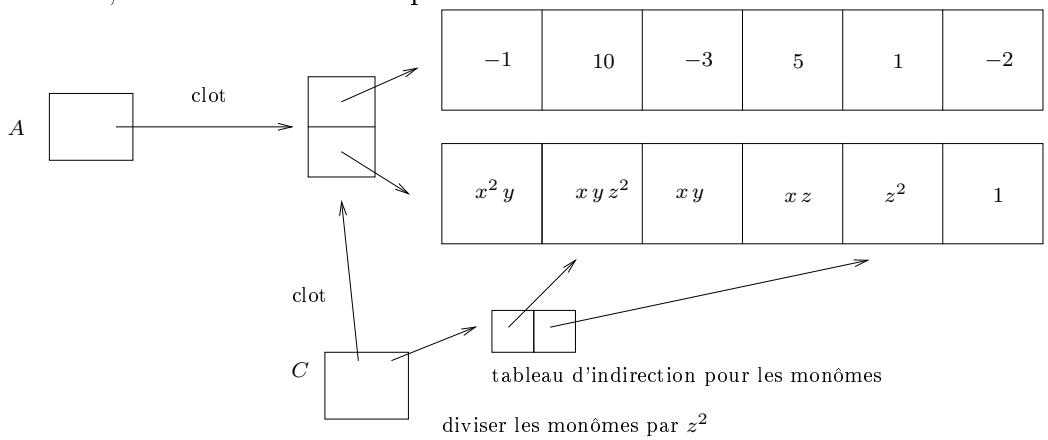
La *clot* de A est le tableau des six termes et des six coefficients suivant. Le polynôme A est essentiellement un pointeur sur sa *clot*.



Affectons maintenant à un polynôme B le coefficient de x dans A. Le polynôme B, qui vaut $10\,y\,z^2 - 3\,y + 5\,z$, est obtenu en *BLAD* sans dupliquer le polynôme A. Essentiellement, il est décrit comme la somme de tous les monômes de la *clot* de A entre les indices 2 et 4, divisés par le terme x.



Affectons maintenant à un polynôme C le coefficient de z^2 dans A. Le polynôme C, qui vaut $10\,x\,y + 1$, est obtenu en $BLAD$ sans dupliquer le polynôme A. Il est décrit comme la somme de certains monômes de la *clot* de A divisés par z^2. Comme ces monômes ne sont pas consécutifs, leur liste est donnée par un tableau d'indirections.



Ce mécanisme peut induire des effets de bord pervers. Le style de programmation est procédural on l'a dit : les procédures reçoivent en paramètre les variables dans lesquelles elles doivent enregistrer leurs résultats et cherchent à récupérer autant que possible la mémoire déjà allouée à ces variables. Que se passerait–il si on demandait d'affecter à C la somme de deux polynômes ? Si on modifie la *clot* de C, les polynômes A et B sont modifiés par effet de bord ! J'ai interdit cette situation en attribuant un booléen *readonly* à tous les polynômes. Les polynômes créés de toute pièce (comme A) peuvent être modifiés : leur booléen vaut *faux*. Par contre les polynômes comme B et C obtenus par sélection de monômes dans un autre polynôme ne peuvent pas être modifiés : leur booléen vaut *vrai*. La règle ci–dessus n'interdit pas tous les effets de bord : si on modifie A, on modifie aussi B et C. On pourrait la renforcer mais ça ne semble pas nécessaire en pratique.

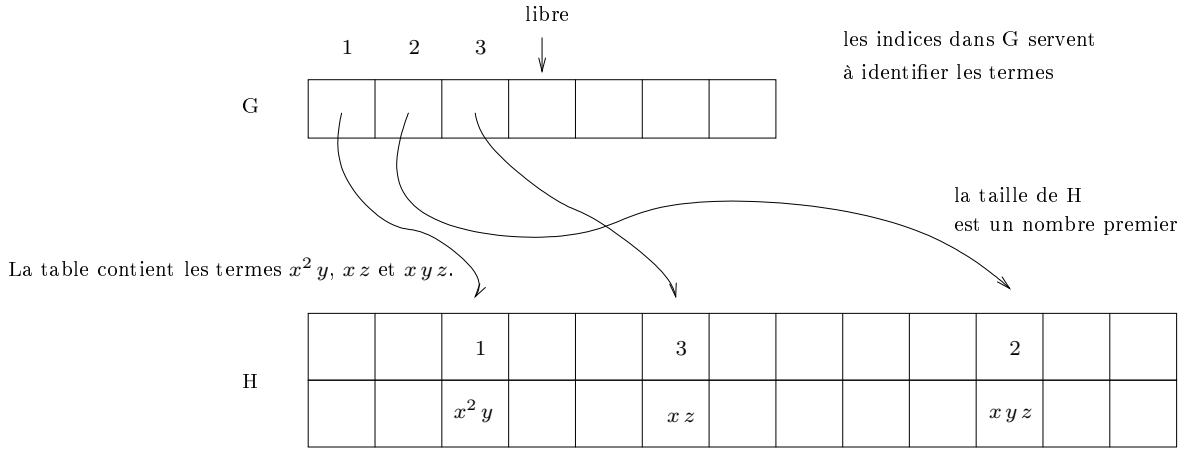
J'ai choisi d'implanter les *clot* sous la forme de deux tableaux (il s'agit en fait de tableaux dynamiques) et non sous la forme d'un tableau de couples (terme, coefficient) pour pouvoir mieux compresser les données. Par exemple, dans le cas des polynômes à coefficients petits entiers modulaires, qui prennent chacun la taille d'un demi–pointeur, j'évite ainsi de perdre de la place pour des raisons d'alignement de mots en mémoire.

Plusieurs codages sont possibles pour les termes. Le codage choisi peut varier d'un polynôme (d'une *clot* plutôt) à l'autre. Il y a le codage simple, le codage compressé et le codage par hachage. Le codage simple n'appelle pas de commentaire.

Le codage compressé, qui est le codage par défaut, est obtenu ainsi : on maintient pour chaque polynôme la liste des variables dont il dépend avec leur degré. J'appelle cette liste le « rang total » du polynôme. Pour la plupart des opérations arithmétiques, il est très facile de déterminer le rang total du résultat à partir des rangs totaux des opérandes. Ce rang total fournit, pour chaque variable, le nombre minimal de bits nécessaire pour coder son degré (c'est la partie entière du logarithme en base deux plus un du degré figurant dans le rang total si je me souviens bien). Dans chaque terme de la *clot*, on réserve, pour chaque variable dont dépend le polynôme, le nombre minimal de bits nécessaires et on code le degré sur ce

nombre de bits.

Lorsqu'on utilise le codage par hachage, les termes sont stockés dans une table de hachage, capable de se redimensionner toute seule, gérée par une technique d'adressage ouvert, en utilisant la variante dite du double hachage [23, chapitre 11]. Cette table est stockée dans une pile spéciale. Les termes stockés dans la table ne sont pas dupliqués, ce qui accélère les tests d'égalité. C'est très intéressant dans le cadre d'algorithmes qui regardent les polynômes comme des vecteurs de coefficients et pour lesquels un terme ne sert qu'à désigner une colonne d'une matrice (algorithme *FGLM*, algorithme de réduction de Jean–Charles Faugère [39], fondés sur des méthodes d'algèbre linéaire). Deux tableaux sont utilisés : G et H. Le tableau G est rempli en incrémentant un indice (noté « libre » dans le graphique ci–dessous). Un terme est identifié par son indice dans G. Les éléments de G sont des pointeurs vers des entrées de H. Le éléments du tableau H contiennent deux champs : le terme et l'identifiant du terme, c'est–à–dire son indice dans G. La taille de H est un nombre premier p, comme il se doit pour la technique du double hachage. Soit à stocker un terme dans la table. On calcule une clef primaire c_1 et une clef secondaire c_2, qui sont des entiers modulo p. On détermine rapidement en parcourant H à partir de l'indice c_1 et en incrémentant de c_2 modulo p si le terme figure déjà dans H. Si c'est le cas, on retourne son identifiant sinon on le stocke dans H et l'indice « libre » de G fournit son identifiant. Pour parcourir l'ensemble de tous les termes créés, il suffit de parcourir G. Si la table H est remplie à plus d'un tiers, on la redimensionne. Pour ventiler dans le nouveau tableau H les termes présents dans l'ancien, il suffit de parcourir G, calculer les nouvelles clefs primaires et secondaires de tous les termes et de les stocker dans le nouveau tableau comme on l'a décrit ci–dessus. Redimensionner H ne modifie pas les identifiants des termes.



10.5.3 Itérateurs de coefficients

Il est impensable de manipuler directement des structures de données aussi compliquées. Les bibliothèques *BLAD* fournissent donc des itérateurs et des créateurs de polynômes. Le programme C suivant montre comment utiliser l'un des deux itérateurs. On y considère un polynôme $A \in \mathbb{Z}[x,\, y,\, z]$. L'ordre sur les variables est $x > y > z$. Dans la première boucle, on

le regarde comme un polynôme de l'anneau $\mathbb{Z}[z][x, y]$ c'est–à–dire comme un polynôme en les variables x et y et à coefficients dans l'anneau $\mathbb{Z}[z]$. Dans la deuxième boucle, on le regarde comme un polynôme de $\mathbb{Z}[x, y][z]$, c'est–à–dire comme un polynôme en z et à coefficients dans $\mathbb{Z}[x, y]$. Pour cela, un deuxième ordre sur les variables est créé (il s'agit de $z > y > x$) et le polynôme A est trié (résultat dans B). Attention au fait que le tri ne modifie pas A. Un tableau d'indirections est créé vers les monômes de la *clot* de A et c'est ce tableau qui est trié vis–à–vis du deuxième ordre sur les variables. Le polynôme B est un polynôme du même type que le polynôme C représenté graphiquement un peu plus haut.

```
#include <bap.h>

int main ()
{   struct bap_polynom_mpz A, B, coeff;
    struct bap_itercoeff_mpz iter;
    struct bav_term term;
    bav_Iordering r;
    bav_variable v;

    bap_restart (0, 0);
    ba0_sscanf2 ("ordering (derivations = [], blocks = [x,y,z])",
        "%ordering", &r);
    bav_R_push_ordering (r);
    bap_init_polynom_mpz (&A);
    ba0_sscanf2 ("x*y*z^2 - 3*x*z + z^2 - y*x^2 - 1", "%Az", &A);
    ba0_sscanf2 ("y", "%v", &v);
    bap_init_polynom_mpz (&coeff);
    bav_init_term (&term);
    ba0_printf ("Polynomial : %Az\n", &A);
    bap_begin_itercoeff_mpz (&iter, &A, v);
    while (! bap_outof_itercoeff_mpz (&iter))
    {   bap_term_itercoeff_mpz (&term, &iter);
        bap_coeff_itercoeff_mpz (&coeff, &iter);
        ba0_printf ("coeff = %Az, term = %term\n", &coeff, &term);
        bap_next_itercoeff_mpz (&iter);
    }
    bap_close_itercoeff_mpz (&iter);
    ba0_sscanf2 ("ordering (derivations = [], blocks = [z,y,x])",
        "%ordering", &r);
    bav_R_push_ordering (r);
    bap_init_readonly_polynom_mpz (&B);
    bap_sort_polynom_mpz (&B, &A);
    ba0_sscanf2 ("z", "%v", &v);
    ba0_printf ("Polynomial : %Az\n", &B);
    bap_begin_itercoeff_mpz (&iter, &B, v);
    while (! bap_outof_itercoeff_mpz (&iter))
```

```
    {   bap_term_itercoeff_mpz (&term, &iter);
        bap_coeff_itercoeff_mpz (&coeff, &iter);
        ba0_printf ("coeff = %Az, term = %term\n", &coeff, &term);
        bap_next_itercoeff_mpz (&iter);
    }
    bap_close_itercoeff_mpz (&iter);
    bap_terminate (ba0_init_level);
    return 0;
}
```

Voici l'affichage obtenu à l'exécution.

```
$ ./bap_iterator
Polynomial : -x^2*y + x*y*z^2 - 3*x*z + z^2 - 1
coeff = -1, term = x^2*y
coeff = z^2, term = x*y
coeff = -3*z, term = x
coeff = z^2 - 1, term = 1
Polynomial : z^2*y*x + z^2 - 3*z*x - y*x^2 - 1
coeff = y*x + 1, term = z^2
coeff = -3*x, term = z
coeff = -y*x^2 - 1, term = 1
```

Il existe un autre type d'itérateur, spécialisé pour énumérer des couples de la forme (coefficient numérique, terme). Il existe aussi un créateur de polynômes qui permet de les construire en énumérant leurs monômes sous la forme de couples (coefficient numérique, terme).

En résumé, je me suis inspiré de mécanismes de manipulation de fichiers dans les langages de programmation traditionnels (fichiers séquentiels et séquentiels indexés). J'ai atteint en *BLAD* des polynômes de près d'un million de monômes. Il me semble assez raisonnable de manipuler un polynôme de cette taille comme on manipulerait un fichier ou même une base de données. J'ai d'ailleurs envisagé de permettre le stockage de certains polynômes directement sur mémoire de masse, de façon transparente pour les procédures qui les manipulent. La représentation choisie pour les polynômes n'est pas destinée à pulvériser des records de vitesse. J'ai plutôt tenté d'économiser la mémoire.

10.5.4 Addition et multiplication

L'addition de deux polynômes est un algorithme simple qui s'apparente à la fusion de deux listes triées. Pour l'addition d'une famille de polynômes, j'ai implanté la structure des « *geobuckets* » préconisée par Thomas Yan dans [107]. L'idée consiste à organiser les sommes intermédiaires de façon à toujours additionner des polynômes de tailles comparables. Concrètement, un geobucket est un tableau G de polynômes vérifiant la propriété suivante : pour tout indice i, la longueur du polynôme G_i est comprise entre 2^i et 2^{i+1}.

Pour la multiplication de deux polynômes, j'ai implanté l'algorithme élémentaire avec un raffinement. Soit à multiplier deux polynômes A et B construits sur des alphabets respectifs X_A et X_B. L'algorithme commence par déterminer les alphabets $Y_{AB} = X_A \cap X_B$, $Y_A = X_A \setminus Y_{AB}$ et $Y_B = X_B \setminus Y_{AB}$. Les polynômes A et B sont réordonnés de façon à être regardés comme des polynômes de $\mathbb{Z}[Y_{AB}][Y_A]$ et de $\mathbb{Z}[Y_{AB}][Y_B]$. L'algorithme de multiplication élémentaire n'est ensuite appliqué que sur les coefficients dans $\mathbb{Z}[Y_{AB}]$ des deux polynômes. Réordonner les deux polynômes n'exige pas de reconstruire les polynômes : les itérateurs de coefficients sont employés pour cela.

Je n'ai pas implanté les algorithmes d'Anatolii Karatsuba et Ofman Yu [55] ou d'Arnold Schönhage et Viktor Strassen [92]. Ces algorithmes, conçus pour des polynômes en une variable peuvent s'appliquer à des polynômes en plusieurs variables : il suffit de regarder ces derniers comme des polynômes en une variable privilégiée et à coefficients dans les polynômes en les autres variables. La représentation récursive des polynômes est mieux adaptée à ce type d'algorithme. La représentation que j'ai adoptée convient aussi : il suffit d'utiliser les itérateurs de coefficients pour émuler une représentation récursive.

10.5.5 Le pgcd de deux polynômes

C'est l'algorithme le plus difficile implanté dans les bibliothèques *BLAD*. Sur les quarante mille lignes des bibliothèques, j'estime qu'il y en a vingt cinq mille rien que pour le pgcd. J'ai essayé d'implanter un algorithme proche de celui de MAPLE dont les grandes lignes sont décrites dans le livre [44] de Keith O. Geddes, Stephen Czapor et George Labahn mais la mise en œuvre en MAPLE est plus subtile que ce décrit le livre. Les algorithmes décrits sont spécialisés pour le cas des polynômes à coefficients entiers.

Plusieurs méthodes sont implantées. Certaines sont très rapides mais peuvent échouer, comme le pgcd heuristique décrit dans [19] et dans [44, page 320]. L'algorithme appelé lorsque tous les autres ont échoué est le « *extended Zassenhaus gcd algorithm* » [74] dû à Joel Moses et David Yun (voir [44, page 314]) qui évalue toutes les variables des polynômes dont on cherche le pgcd sauf une (privilégiée), qui calcule le pgcd des polynômes évalués et qui remonte le résultat en utilisant un schéma inspiré du « lemme de Hensel » [48]. Le lemme de Kurt Wilhelm Sebastian Hensel est au départ un lemme théorique concernant les nombres p–adiques [108, chapter VIII, paragraph 7] et il semble que Hans Julius Zassenhaus ait été le premier à l'adapter dans [109] pour les calculs de pgcd et de factorisation de polynômes. Joel Moses et David Yun ont donc baptisé leur algorithme en son honneur.

Toute la difficulté pratique réside dans la remontée de Hensel. Le problème consiste à bien gérer les coefficients initiaux des polynômes vis–à–vis de la variable privilégiée, de façon à éviter de calculer leurs inverses algébriques modulo différents idéaux. La solution adoptée en MAPLE et en *BLAD* est due à Paul Shyh–Horng Wang [106]. C'est au départ une idée qui s'applique à la factorisation complète d'un polynôme f en plusieurs variables : connaissant une factorisation complète de l'initial de f vis–à–vis de la variable privilégiée et une factorisation du polynôme f' obtenu en évaluant toutes les autres variables de f, il est (souvent) possible d'attribuer chacun des facteurs de l'initial aux facteurs de f'. Cette connaissance permet d'éviter la croissance des données lors de la remontée de Hensel mais la factorisa-

tion complète de polynômes en plusieurs variables qu'elle nécessite est un procédé coûteux. L'astuce consiste à bien choisir la variable privilégiée de façon à factoriser complètement des initiaux aussi petits que possible.

La factorisation complète de polynômes en plusieurs variables applique en fait des mécanismes très proches de ceux mis en œuvre pour le pgcd ; évaluer toutes les variables sauf une, factoriser complètement le polynôme en une variable obtenu et remonter le résultat en utilisant un schéma inspiré du lemme de Hensel. Pour la factorisation des polynômes en une variable, je n'ai implanté que la méthode élémentaire d'Elwyn Ralph Berlekamp [4] (voir aussi [44, page 347] ou encore le livre en Français [25, chapitre 4] de James Davenport, Yvon Siret et Évelyne Tournier).

La réalisation de ces algorithmes nécessite des polynômes non seulement à coefficients entiers mais aussi à coefficients entiers modulo p où p est soit un entier machine soit un grand entier. Des polynômes à coefficients rationnels sont aussi un peu utiles, ne serait–ce que pour les entrées-sorties. Pour éviter de coder quatre fois les mêmes choses, j'ai « bricolé » un mécanisme me fournissant un peu de généricité sur les coefficients des polynômes. Le code générique ainsi obtenu est instancié sur les quatre types de coefficients numériques par des filtres écrits grâce à l'utilitaire UNIX *sed*.

Conclusion

Même si je continue à travailler la théorie des algorithmes dédiés à l'élimination différentielle, je consacre de plus en plus de mon temps à leur chercher des applications, notamment dans le domaine de la modélisation en biologie. Plus précisément, je cherche des applications qui nécessitent la mise au point de chaînes logicielles qui puissent profiter ponctuellement d'un outil simple d'emploi dédié à l'élimination différentielle. J'ai réalisé les bibliothèques *BLAD* dans ce but et je les considère comme le principal résultat de mon activité de recherche de ces cinq dernières années. Mon expérience passée avec *diffalg*, l'expérience des autres avec *GB+RS*, *GMP*, *Triade* etc ... montrent l'importance que peuvent prendre des logiciels lorsqu'ils sont bien réalisés. Le développement de chacun de ces logiciels a toujours pris énormément de temps et d'énergie à leurs auteurs : il faut donc bien les réaliser *avant* de chercher des applications.

Dans le domaine théorique, l'étude et l'adaptation d'intégrateurs numériques d'équations différentielles qui puissent être appliqués aux chaînes différentielles régulières produites par les algorithmes d'élimination différentielle constitue un objectif essentiel. En effet, les applications à la modélisation en biologie approfondies dans l'équipe calcul formel (projet *LÉPISME*, réduction de modèles) montrent l'importance de prolonger les calculs symboliques par des calculs numériques et notamment (dans le cadre de la réduction de modèles) par de l'intégration d'équations différentielles. À quoi servirait–il en effet d'automatiser la phase d'élimination différentielle si le traitement qui la suit doit être mené manuellement ?

Dans le domaine des applications, je pense que la présentation, l'interface — la conception — des composants logiciels que nous produisons est beaucoup plus importante que leur capacité de calcul brute. C'est elle qui déterminera si l'élimination différentielle doit se diffuser auprès du grand public scientifique ou pas. Dans le monde de la recherche en calcul formel, c'est un aspect malheureusement considérablement négligé : le monde du calcul formel est majoritairement peuplé de mathématiciens qui confondent informatique et algorithmique. L'algorithmique est une discipline très proche des mathématiques. La conception de logiciels, c'est autre chose.

Les retours d'information reçus sur notre proposition de continuation du projet *LÉPISME* me font penser qu'il vaut mieux développer une bibliothèque dédiée à l'estimation de paramètres qu'un logiciel fermé : la méthode mise en œuvre dans *LÉPISME* enchaîne plusieurs étapes qui peuvent chacune être effectuées de plusieurs façons. Je pense qu'il est préférable de laisser plus de souplesse à l'utilisateur. Si mon analyse est juste, on voit que l'erreur à corriger est une erreur de conception de logiciel et pas une erreur théorique.

Les moyens modernes ont permis aux biologistes d'amasser de gigantesques quantités de données. La difficulté consiste à les gérer. C'est cette difficulté qui a été mise en avant par Bernard Vandenbunder lors de la création de l'Institut de Recherche Interdisciplinaire. C'est elle aussi qui a motivé la mise au point du langage SBML : le volumineux modèle SBML que nous étudions en ce moment a été construit manuellement, via une interface graphique, à partir d'une liste de 242 articles. On se doute bien que de tels modèles comportent des erreurs. Il ne sert donc à rien d'essayer de leur appliquer des calculs et des raisonnements algébriques sophistiqués. Le premier travail des auteurs de tels modèles va probablement consister à essayer de les valider et de les corriger peu à peu. Je pense que les chaînes logicielles qui serviront ce but pourront avoir l'usage de bibliothèques d'élimination différentielle, ne serait–ce que pour mener certains calculs simples de préparation d'équations. Encore faut–il que ces bibliothèques soient simples d'emploi.

Pour que ce vœu se réalise et pour prendre en compte une remarque qui nous a été adressée au sujet de *LÉPISME*, je suis entré en contact avec les auteurs de ces modèles pour interagir très tôt avec eux. C'est plus facile à faire dans le cadre ci–dessus que ce ne l'était dans le cadre du projet *LÉPISME*.

Index

Bibliographie

[1] Philippe Aubry. *Ensembles triangulaires de polynômes et résolution de systèmes algébriques. Implantation en Axiom.* PhD thesis, Univ. Paris VI, 1999.

[2] Philippe Aubry, Daniel Lazard, and Marc Moreno Maza. On the theories of triangular sets. *Journal of Symbolic Computation*, 28 :105–124, 1999.

[3] Stefania Audoly, Giuseppina Bellu, Leontina D'Angio, Maria Pia Saccomani, and Claudio Cobelli. Global Identifiability of Nonlinear Models of Biological Systems. *IEEE Transactions on Biomedical Engineering*, 48(1) :55–65, 2001.

[4] Elwyn Ralph Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46 :1853–1859, 1967.

[5] François Boulier. A new criterion to avoid useless critical pairs in Buchberger's algorithm. Technical report, Université Lille I, 59655, Villeneuve d'Ascq, France, October 2001. (ref. LIFL 2001–07).

[6] François Boulier, Lilianne Denis-Vidal, Thibaut Henin, and François Lemaire. LÉPISME. In *presented at the ICPSS conference*, 2004. submitted to the Journal of Symbolic Computation.

[7] François Boulier. *Étude et implantation de quelques algorithmes en algèbre différentielle*. PhD thesis, Université Lille I, 59655, Villeneuve d'Ascq, France, 1994.

[8] François Boulier. Efficient computation of regular differential systems by change of rankings using Kähler differentials. Technical report, Université Lille I, 59655, Villeneuve d'Ascq, France, November 1999. (ref. LIFL 1999–14, presented at the MEGA2000 conference).

[9] François Boulier, Daniel Lazard, François Ollivier, and Michel Petitot. Representation for the radical of a finitely generated differential ideal. In *Proceedings of ISSAC'95*, pages 158–166, Montréal, Canada, 1995.

[10] François Boulier, Daniel Lazard, François Ollivier, and Michel Petitot. Computing representations for radicals of finitely generated differential ideals. Technical report, Université Lille I, LIFL, 59655, Villeneuve d'Ascq, France, 1997. (ref. IT306, december 1998 version published in the habilitation thesis of Michel Petitot).

[11] François Boulier, François Lemaire, and Marc Moreno Maza. PARDI ! In *proceedings of ISSAC'01*, pages 38–47, London, Ontario, Canada, 2001.

[12] François Boulier and François Lemaire. Computing canonical representatives of regular differential ideals. In *proceedings of ISSAC 2000*, pages 37–46, St Andrews, Scotland, 2000.

[13] François Boulier, François Lemaire, and Marc Moreno Maza. Well known theorems on triangular systems and the D^5 principle. In *Proceedings of Transgressive Computing 2006*, pages 79–91, Granada, Spain, 2006.

[14] François Boulier, Marc Moreno Maza, and Cosmin Oancea. A new henselian construction and its application to polynomial gcds over direct products of fields. In *proceedings of EACA'04*, Universidad de Santander, Spain, 2004.

[15] François Boulier and Sylvain Neut. Cartan's characters and stairs of characteristic sets. In *Proceedings of AAECC 11*, pages 363–372, Sydney, Australia, 2001.

[16] Driss Bouziane, Abdelillah Kandri Rody, and Hamid Maârouf. Unmixed–Dimensional Decomposition of a Finitely Generated Perfect Differential Ideal. *Journal of Symbolic Computation*, 31 :631–649, 2001.

[17] Bruno Buchberger. *A criterion for detecting unnecessary reductions in the construction of Gröbner bases*, volume 72 of *Lecture Notes in Computer Science*, pages 3–21. Springer Verlag, 1979.

[18] David Castro, Klemens Hägele, J. E. Morais, and Luis Miguel Pardo. Kronecker's and Newton's approaches to solving : a first comparison. *Journal of complexity*, 17(1) :212–303, 2001.

[19] B. W. Char, Keith O. Geddes, and Gaston H. Gonnet. GCDHEU : Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation. *Journal of Symbolic Computation*, 9 :31–48, 1989.

[20] Yves Cherruault. *Modèles et méthodes mathématiques pour les sciences du vivant.* PUF, Paris, 1998.

[21] Shang-Ching Chou and Xiao-Shan Gao. On the dimension of an arbitrary ascending chain. *Chinese Bulletin of Science*, 38 :799–904, 1993.

[22] George Edwin Collins and Alkiviadis G. Akritas. Polynomial real root isolation using Descartes'rule of signs. In *proceedings of ISSAC'76*, pages 272–275, Yorktown Heights NY, 1976.

[23] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction à l'algorithmique.* Dunod, Paris, 2 edition, 2002.

[24] Xavier Dahan, Marc Moreno Maza, Éric Schost, Wenyuan Wu, and Yuzhen Xie. Lifting Techniques for Triangular Decompositions. In *proceedings of ISSAC 2005*, Beijing, China, 2005.

[25] James H. Davenport, Yvon Siret, and Évelyne Tournier. *Calcul formel, Systèmes et algorithmes de manipulations algébriques.* études et recherches en informatique. Masson, Paris, 1987.

[26] Michel Demazure. *Cours d'algèbre. Primalité. Divisibilité. Codes.* Cassini, Paris, 1997.

[27] S. Demignot and D. Domurado. Effect of prosthetic sugar groups on the pharmacokinetics of glucose-oxidase. *Drug Design Deliv.*, 1 :333–348, 1987.

[28] Jan Denef and Leonard Lipshitz. Power Series Solutions of Algebraic Differential Equations. *Mathematische Annalen*, 267 :213–238, 1984.

[29] Lilianne Denis-Vidal. *Identifiabilité de modèles non linéaires paramétriques de dynamiques classiques et à retard. Planification d'expériences et estimation de paramètres.* Mémoire d'Habilitation à Diriger des Recherches, Université de Technologie de Compiègne, may 2004.

[30] Lilianne Denis-Vidal, Ghislaine Joly-Blanchard, and Céline Noiret. System identifiability (symbolic computation) and parameter estimation (numerical computation). In *Numerical Algorithms*, volume 34, pages 282–292, 2003.

[31] Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Math.*, 35 :413–422, 1913.

[32] Sette Diop. Elimination in Control Theory. *Mathematics of Control, Signal and Systems*, 4 :17–42, 1991.

[33] Sette Diop. Differential algebraic decision methods and some application to system theory. *Theoretical Computer Science*, 98 :137–161, 1992.

[34] Sette Diop and Michel Fliess. Nonlinear observability, identifiability, and persistent trajectories. In *Proc. 30th CDC*, pages 714–719, Brighton, 1991.

[35] Jean Della Dora, Claire Dicrescenzo, and Dominique Duval. About a new method for computing in algebraic number fields. In *Proceedings of EUROCAL85, vol. 2*, volume 204 of *Lecture Notes in Computer Science*, pages 289–290. Springer Verlag, 1985.

[36] Lionel Ducos. Optimizations of the subresultant algorithm. *Journal of Pure and Applied Algebra*, 145 :149–163, 2000.

[37] David Eisenbud. *Commutative Algebra with a View Toward Algebraic Geometry*, volume 150 of *Graduate Texts in Mathematics*. Springer Verlag, 1995.

[38] Jean-Charles Faugère. A fast, easy to use algorithm for dynamic memory management. Technical report, LIP6, université Paris VI, january 1998. (unpublished).

[39] Jean-Charles Faugère. A new efficient algorithm to compute Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139 :61–88, 1999.

[40] Jean-Charles Faugère. A new efficient algorithm for computing Göbner bases without reduction to zero (F_5). In *proceedings of ISSAC 2002*, pages 75–83, Villeneuve d'Ascq, France, 2002.

[41] Jean-Charles Faugère, Patricia Gianni, Daniel Lazard, and Teo Mora. Efficient computation of Gröbner bases by change of orderings. *Journal of Symbolic Computation*, 16 :329–344, 1993.

[42] Michel Fliess. Automatique et corps différentiels. *Forum Math.*, 1 :227–238, 1989.

[43] R. Gebauer and H. M. Möller. On an Installation of Buchberger's Algorithm. *Journal of Symbolic Computation*, 6(2&3) :275–286, October/December 1988.

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

[45] Ernst Hairer, Syvert Paul Norsett, and Gerhard Wanner. *Solving ordinary differential equations I. Nonstiff problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer–Verlag, New York, 2 edition, 1993.

[46] Ernst Hairer and Gerhard Wanner. *Solving ordinary differential equations II. Stiff and Differential–Algebraic Problems*, volume 14 of *Springer Series in Computational Mathematics*. Springer–Verlag, New York, 2 edition, 1996.

[47] Eldon Hansen. *Global Optimization Using Interval Analysis*. Marcel Dekker Inc., New York, 1992.

[48] Kurt Wilhelm Sebastian Hensel. *Theorie der Algebraischen Zahlen*. Teubner, Leipzig, 1908.

[49] Évelyne Hubert. *Étude Algébrique et Algorithmique des Singularités des Équations Différentielles Implicites*. PhD thesis, Institut National Polytechnique de Grenoble, France, 1997.

[50] Évelyne Hubert. Factorization free decomposition algorithms in differential algebra. *Journal of Symbolic Computation*, 29(4,5) :641–662, 2000.

[51] Évelyne Hubert and Nicolas Le Roux. Computing Power Series Solutions of a Nonlinear PDE System. In *Proceedings of ISSAC03*, pages 148–155, Philadelphia, USA, 2003.

[52] Joseph Johnson. Kähler differentials and differential algebra. *Annals of Mathematics*, 89 :92–98, 1969.

[53] Erwin Kähler. *Einfürhung in die Theorie der Systeme von Differentialgleichungen*. Teubner, Leipzig, Germany, 1934.

[54] Mickael Kalkbrener. A Generalized Euclidean Algorithm for Computing Triangular Representations of Algebraic Varieties. *Journal of Symbolic Computation*, 15 :143–167, 1993.

[55] Anatolii Alekseevich Karatsuba and Ofman Yu. Multiplication of multiple numbers by mean of automata. *Dokadly Akad. Nauk SSSR*, 145(2) :293–294, 1962.

[56] Ellis Robert Kolchin. *Differential Algebra and Algebraic Groups*. Academic Press, New York, 1973.

[57] Klaus Kühnle and Ernst Wilhelm Mayr. Exponential Space Computation of Gröbner Bases. In *proceedings of ISSAC'96*, pages 63–71, Zürich, Switzerland, 1996.

[58] Daniel Lazard. A new method for solving algebraic systems of positive dimension. *Discrete Applied Mathematics*, 33 :147–160, 1991.

[59] Daniel Lazard. Solving Zero–dimensional Algebraic Systems. *Journal of Symbolic Computation*, 13 :117–131, 1992.

[60] François Lemaire. An orderly linear PDE system with analytic initial conditions with a non analytic solution. *Special Issue on Computer Algebra and Computer Analysis, Journal of Symbolic Computation*, 35(5) :487–498, 2003.

[61] François Lemaire, Marc Moreno Maza, and Yuzhen Xie. The RegularChains library in MAPLE 10. In Ilias S. Kotsireas, editor, *The MAPLE conference*, pages 355–368, 2005.

[62] François Lemaire. *Contribution à l'algorithmique en algèbre différentielle*. PhD thesis, Université Lille I, 59655, Villeneuve d'Ascq, France, january 2002.

[63] Ziming Li and Dongming Wang. Coherent, regular and simple systems in zero decompositions of partial differential systems. *Systems Science and Mathematical Sciences*, 12 :43–60, 1999.

[64] L. Ljung and S. T. Glad. On global identifiability for arbitrary model parametrisations. *Automatica*, 30 :265–276, 1994.

[65] Henri Lombardi, Marie-Françoise Roy, and Mohab Safey El Din. New structure theorem for subresultants. *Journal of Symbolic Computation*, 29(4,5) :663–690, 2000.

[66] Hamid Maârouf. *Étude de Quelques Problèmes Effectifs en Algèbre Différentielle*. PhD thesis, Université Cadi Ayyad, Morocco, 1996.

[67] Elizabeth L. Mansfield. *Differential Gröbner Bases*. PhD thesis, University of Sydney, Australia, 1991.

[68] Yu Matijasevic. Enumerable sets are diophantine. *Sov. Math. Dokl.*, 11 :354–357, 1970.

[69] Marc Moreno Maza. *Calculs de Pgcd au–dessus des Tours d'Extensions Simples et Résolution des Systèmes d'Équations Algébriques*. PhD thesis, Université Paris VI, France, 1997.

[70] Marc Moreno Maza. On Triangular Decompositions of Algebraic Varieties. Technical report, NAG, 2000. (presented at the MEGA2000 conference, submitted to the JSC).

[71] Marc Moreno Maza and Renaud Rioboo. Polynomial gcd computations over towers of algebraic extensions. In *Proceedings of AAECC11*, pages 365–382. Springer Verlag, 1995.

[72] Sally Morrison. Yet another proof of Lazard's lemma. private communication, december 1995.

[73] Sally Morrison. The Differential Ideal $[P] : M^{\infty}$. *Journal of Symbolic Computation*, 28 :631–656, 1999.

[74] Joel Moses and David Yuan Yee Yun. The EZGCD Algorithm. In *Proceedings of the annual ACM conference*, pages 159–166, 1973.

[75] Céline Noiret. *Utilisation du calcul formel pour l'identifiabilité de modèles paramétriques et nouveaux algorithmes en estimation de paramètres*. PhD thesis, Université de Technologie de Compiègne, 2000.

[76] Oussous Nour-Eddine. *Contribution à l'identification de modèles par l'algèbre non commutative*. Mémoire d'Habilitation à Diriger des Recherches, Université Lille I, décembre 2001.

[77] Kanae Oda, Yukiko Matsuoka, Akira Funahashi, and Hiroaki Kitano. A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, 2005.

[78] François Ollivier. *Le problème de l'identifiabilité structurelle globale : approche théorique, méthodes effectives et bornes de complexité*. PhD thesis, École Polytechnique, Palaiseau, France, 1990.

[79] François Ollivier. A proof of Lazard's lemma. private communication, october 1998.

[80] Ariane Péladan-Germa. *Tests effectifs de Nullité dans des extensions d'anneaux différentiels*. PhD thesis, École Polytechnique, Palaiseau, France, 1997.

[81] Hervé Perdry. *Aspects constructifs de la théorie des corps valués*. PhD thesis, Université de Franche–Comté, 2001.

[82] Jean-François Pommaret. New Perspectives in Control Theory for Partial Differential Equations. *IMA Journal of Mathematics Control and Information*, 9 :305–330, 1992.

[83] Gregory J. Reid, Ping Lin, and Allan D. Wittkopf. Differential Elimination–Completion Algorithms for DAE and PDAE. *Studies in Applied Mathematics*, 106(1) :1–45, 2001.

[84] Gregory J. Reid, Allan D. Wittkopf, and Alan Boulton. Reduction of systems of nonlinear partial differential equations to simplified involutive forms. *European Journal of Applied Math.*, pages 604–635, 1996.

[85] Charles Riquier. *Les systèmes d'équations aux dérivées partielles*. Gauthier–Villars, Paris, 1910.

[86] Joseph Fels Ritt. *Differential Algebra*. Dover Publications Inc., New York, 1950. Available at `http://www.ams.org/online_bks/coll33`.

[87] Azriel Rosenfeld. Specializations in differential algebra. *Trans. Amer. Math. Soc.*, 90 :394–407, 1959.

[88] Fabrice Rouillier and Paul Zimmermann. Efficient isolation of a polynomial real roots. Technical report, INRIA, February 2001. (number 4113).

[89] Colin J. Rust, Gregory J. Reid, and Allan D. Wittkopf. Existence and Uniqueness Theorems for Formal Power Series Solutions of Analytic Differential Systems. In *proceedings of ISSAC'99*, Vancouver, Canada, 1999.

[90] Brahim Sadik. Une note sur les algorithmes de décomposition en algèbre différentielle. *Comptes Rendus de l'Académie des Sciences*, 330 :641–646, 2000.

[91] Josef Schicho and Ziming Li. A construction of radical ideals in polynomial algebra. Technical report, RISC, Johannes Kepler University, Linz, Austria, august 1995.

[92] Arnold Schönage and Volker Strassen. Schnelle Multiplikation Grosser Zahlen. *Computing*, 7 :281–292, 1971.

[93] Alexandre Sedoglavic. A Probabilistic Algorithm to Test Local Algebraic Observability in Polynomial Time. *Journal of Symb. Comp.*, 33(5) :735–755, 2002.

[94] Abraham Seidenberg. Some basic theorems in differential algebra (characteristic p arbitrary). *Trans. Amer. Math. Soc.*, 73 :174–190, 1952.

[95] Abraham Seidenberg. An elimination theory for differential algebra. *Univ. California Publ. Math. (New Series)*, 3 :31–65, 1956.

[96] Abraham Seidenberg. Abstract differential algebra and the analytic case. *Proc. Amer. Math. Soc.*, 9 :159–164, 1958.

[97] Abraham Seidenberg. Abstract differential algebra and the analytic case II. *Proc. Amer. Math. Soc.*, 23 :689–691, 1969.

[98] Natacha Skrzypczak. Modélisation des réseaux de gènes : formalisme SBML2. Réduction des modèles biologiques. Technical report, Mémoire de Master 2. Université Lille I, France, 2005.

[99] Barry M. Trager. Algebraic Factoring and Rational Function Integration. In *proceedings of the ACM Symposium on Symbolic and Algebraic Computation*, pages 219–226, 1976.

[100] Wu Wen Tsün. On the foundation of algebraic differential geometry. *Mechanization of Mathematics, research preprints*, 3 :2–27, 1989.

[101] Bruno Louis van der Waerden. *Algebra*. Springer Verlag, Berlin, seventh edition, 1966.

[102] Nathalie Verdière. *Identifiabilité de systèmes d'équations aux dérivées partielles semi–discrétisées et applications à l'identifiabilité paramétrique de modèles en pharmacocinétique ou en pollution*. PhD thesis, Université de Technologie de Compiègne, France, 2005.

[103] José M. G. Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Science of the USA*, 99(9) :5988–5992, 2002.

[104] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, United Kingdom, 1999.

[105] Éric Walter. *Identifiability of State Space Models*, volume 46 of *Lecture Notes in Biomathematics*. Springer Verlag, 1982.

[106] Paul Shyh-Horng Wang. An Improved Multivariate Polynomial Factoring Algorithm. *Math. Comp.*, 32(144) :1215–1231, 1978.

[107] Thomas Yan. The Geobucket Data Structure for Polynomials. *Journal of Symbolic Computation*, 25 :285–293, 1998.

[108] Oscar Zariski and Pierre Samuel. *Commutative Algebra*. Van Nostrand, New York, 1958.

[109] Hans Julius Zassenhaus. On Hensel Factorization I. *Journal of Number Theory*, 1 :291–311, 1969.