THESE

présentée à

L'UNIVERSITE DE LIMOGES

pour obtenir le

DOCTORAT DE L'UNIVERSITE DE LIMOGES
en MATHEMATIQUES
Option : Calcul formel

(arrêté du 30 Mars 1992)

par

Teresa GOMEZ-DIAZ

Quelques applications de l'évaluation dynamique

Soutenue le 21 Janvier 1994 devant le jury composé de

Président	:	**Guy ROBIN**
Rapporteurs	:	**Philippe FLAJOLET**
		Tomás RECIO
Examinateurs	:	**M. Abdelfattah BARKATOU**
		Dominique DUVAL
		Christian LAIR
		Daniel LAZARD
		Juan LLOVET

A mi madre

. . . Y
a "mis" niños
Julián,
Mario
y Andrés.

merci à Dominique Duval.

Dominique a su me proposer un travail qui m'a accrochée à la recherche, a su supporter tous les inconvénients de travailler avec moi avec beaucoup de patience, et m'a toujours encouragée à faire mieux (même si cela lui a demandé du travail supplémentaire ...). Je lui en suis particulièrement reconnaissante.

merci à Tomás Recio et Philippe Flajolet.

Ils ont rapporté sur ce travail, et je suis sûre que ce n'était pas une tâche facile, surtout compte tenu des délais ...

merci à Guy Robin et aux membres du jury.

Ils me font un grand honneur de présider et composer mon jury de thèse.

gracias à Tomás Recio.

Tomás siempre ha sabido ofrecer una nueva oportunidad a este espiritu inquieto. Su apoyo y su interés constante a sido siempre importante para mi.

gracias a los equipos de Algebra Computational de Santander y Alcalá de Henares.

merci à Odile Simon,

pour sa collaboration au début de la clôture constructible.

merci à l'équipe de Calcul Formel et au Département de Mathématiques de Limoges,

qui m'ont toujours accueillie fort gentiment.

merci à mes collègues de bureau : Abdel, Anne, Carlos, Marc.

Je suis très heureuse d'avoir des collègues comme vous.

merci à Joël

qui m'a toujours sauvée des désastres informatiques avec gentillesse et qui, avec Anne, Cathy et Marc, a contribué à la version finale de cette thèse.

gracias a mi madre, familia y amigos.

siempre cerca de mi y a veces un poco lejos.

Le commencement de cette thèse a été partiellement financé par El Consejo Social de la Universidad de Alcalá de Henares avec deux aides à la recherche :

- Septembre–Octobre 1991 où j'ai commencé le programme de calcul de la forme canonique de Jordan.
- Juillet–Août 1992 où je suis arrivée à une première version du programme de la clôture constructible dynamique.

La suite de ces deux travaux a été partiellement financée par :

- un projet de recherche espagnol : CICyT PB 89/0379/C02/01 (avant 92) – CICyT PB 92/0498/C02/01 (après 92).
- un projet de recherche européen : POSSO Esprit/BRA 6846.

Cette thèse a été élaborée au sein du LACO : "Laboratoire d'Arithmétique, Calcul Formel et Optimisation", URA CNRS 1586, Université de Limoges.

Table des matières

Introduction

Ce mémoire est composé de deux travaux qui sont indépendants, et dont la caractéristique commune est qu'ils utilisent l'évaluation dynamique :

I. Calcul de la forme canonique de Jordan.

II. Clôture constructible dynamique.

Commençons pour une brève description de l'évaluation dynamique.

L'*évaluation dynamique* est une façon de faire des calculs plus générale que les techniques habituelles. Elle peut être décrite comme une automatisation – et une généralisation – du calcul avec des *paramètres* tel qu'il est fait “à la main”.

Par exemple considérons l'équation

$$ax + b = 0$$

avec x variable et a et b paramètres. Si nous nous intéressons aux solutions de cette équation et si nous effectuons les calculs à la main, nous trouvons une réponse du type :

$$\begin{cases} \text{Si } a = 0 \text{ et } b = 0 \text{ alors } x \text{ } \textit{quelconque} \\ \\ \text{Si } a \neq 0 \text{ et } b \text{ } \textit{quelconque} \text{ alors } x = \frac{-b}{a} \end{cases}$$

tandis que la solution obtenue dans certains systèmes de calcul formel est :

$$x = \frac{-b}{a}$$

L'évaluation dynamique nous permet d'obtenir de façon automatique la première réponse.

Pourquoi une telle différence entre ces deux réponses ?

Il est clair que le problème se pose au moment d'effectuer le test d'égalité :

- quand les systèmes de calcul formel se posent la question “$a = 0$?”, ils répondent *non*
- quand nous nous sommes posés la même question, nous avons répondu *quelquefois oui et quelquefois non.*

L'évaluation dynamique est le processus de calcul qui tient compte de ce problème :

> *"Une question peut avoir plusieurs réponses, et il faut continuer le calcul pour chacune des réponses."*

Elle est définie de façon rigoureuse en utilisant la théorie des esquisses [DR].

La première réalisation de l'évaluation dynamique a été implantée par C. Dicrescenzo et D. Duval [DDD, DD] (dans un premier temps en Reduce, actuellement en Axiom). Elle a permis l'automatisation des calculs avec des nombres algébriques, c'est le programme de la *Clôture algébrique dynamique*. Dans ce programme les nombres algébriques sont considérés comme des paramètres d'un type spécial, et ils sont représentés par un symbole soumis à des conditions *algébriques* :

$$a \text{ tel que } P(a) = 0$$

où P est un polynôme (en une variable).

Passons à la description des travaux présentés ici. L'ordre de présentation de ces travaux correspond à l'ordre chronologique, puisque notre premier travail en relation avec l'évaluation dynamique a porté sur la forme de Jordan. Mais il répond aussi à une autre raison importante : dans la première partie nous sommes de simples *utilisateurs* de l'évaluation dynamique, ce qui nous a permis de nous concentrer sur la partie mathématique et algorithmique du problème. Dans la deuxième partie nous proposons un outil pour faire des calculs, difficile à construire sans une connaissance profonde de l'évaluation dynamique.

I. Calcul de la forme canonique de Jordan

Dans la première partie de ce mémoire, nous étudions le calcul de la forme canonique de Jordan d'une matrice. Nous avons programmé le calcul de la forme canonique de Jordan, d'une matrice de passage et de son inverse. Par exemple, ces trois matrices peuvent être utilisées pour calculer de l'exponentielle d'une matrice lors de la résolution d'un système d'équations différentielles [Laz].

Dans notre programme nous utilisons la clôture algébrique dynamique pour les calculs avec les valeurs propres de la matrice donnée. Le programme de la clôture algébrique dynamique résout de façon élégante les problèmes que posent habituellement les calculs avec des nombres algébriques. De plus, l'utilisateur n'est pas censé connaître l'intérieur des programmes de la clôture algébrique ou de l'évaluation dynamique.

Pour réaliser efficacement ces calculs, il est préférable qu'une seule valeur propre intervienne à tout moment. Cette condition est vérifiée pour le calcul de la matrice de Jordan et de la matrice de passage. Ozello montre cette condition pour l'inverse d'une telle matrice de passage [Oz].

Nous avons obtenu deux nouvelles méthodes pour calculer la matrice inverse d'une matrice de passage, de façon que les calculs sont effectués en utilisant une seule valeur propre à chaque fois. Ces deux méthodes sont valables pour des matrices de passage plus générales que celles du cas de Jordan. Pour l'obtention de ces méthodes, nous avons étudié la structure de l'espace total, décomposé en somme directe de sous-espaces caractéristiques correspondant aux valeurs propres de la matrice donnée.

Ces deux méthodes sont améliorées dans le cas particulier de Jordan. Ces améliorations sont obtenues à partir de la structure de chaque sous-espace caractéristique pour chaque valeur propre.

En général, les programmes et méthodes existants pour le calcul de la forme canonique de Jordan utilisent des formes canoniques intermédiaires pour son obtention (forme de Frobenius, etc..., [Oz, MO, Gil, ML] par exemple). Cela est dû principalement aux problèmes de calcul avec des nombres algébriques.

L'utilisation de la clôture algébrique dynamique nous permet de nous affranchir de ces problèmes. Nous calculons directement la forme de Jordan d'une matrice, sans passer par d'autres formes canoniques. De plus, il semble que nos méthodes sont facilement parallélisables, ce qui paraît difficile pour les autres méthodes [RV].

II. Clôture constructible dynamique

Dans la deuxième partie, nous décrivons le programme de la *Clôture constructible dynamique*. Ce programme permet de calculer avec des paramètres en un sens très général (qui inclut les paramètres soumis à des conditions algébriques).

Nous considérons un *paramètre* comme une constante de valeur non spécifiée. Lorsqu'un paramètre intervient dans un calcul, toutes ses valeurs possibles doivent être considérées. C'est l'évaluation dynamique qui se charge de détecter des comportements différents pour différentes valeurs lors du calcul, et de gérer l'obtention du résultat dans chaque cas possible.

C'est la première fois qu'une telle méthode de calcul est implantée de manière tout-à-fait systématique.

Nous montrons aussi comment ce programme peut être utilisé pour la résolution des systèmes polynomiaux et pour la démonstration de théorèmes en géométrie élémentaire.

Résolution des systèmes polynomiaux C'est à travers la clôture constructible dynamique que l'évaluation dynamique propose une méthode pour résoudre les systèmes polynomiaux. La description de la solution consiste en une famille de systèmes triangulaires, sans solution commune, et sans introduire de solution *parasite*.

Il existe plusieurs méthodes pour résoudre (en un sens voisin du nôtre) des systèmes polynomiaux [Bu, Kal, La, Se]. La méthode proposée ici est proche de celle de Seidenberg [Se].

Un avantage de notre méthode par rapport aux autres est la possibilité de considérer de façon simple une classe plus large de systèmes, les systèmes *constructibles* où les équations contiennent des égalités ($=$) et des non-égalités ($\neq$).

Démonstration de théorèmes en géométrie élémentaire Un théorème en géométrie peut être traduit dans la clôture constructible dynamique. Une fois choisi un système de coordonnées, les points de la figure sont représentés par des paramètres. Les hypothèses sont traduites comme l'imposition de contraintes (interdire ou imposer certaines valeurs aux paramètres), la conclusion comme l'étude d'un test d'égalité sur les paramètres.

D'autres méthodes existent pour étudier la démonstration automatique [BCK, CY], mais leur utilisation révèle quelques problèmes [CY]. Nous montrons sur un exemple que l'utilisation de la clôture constructible dynamique permet d'éviter certains de ces problèmes. Par exemple, nous pouvons éviter de façon simple certains types de figures (triangles dégénérés, etc ...) en imposant des contraintes.

Forme canonique de Jordan d'une matrice avec des paramètres Avec le programme de la clôture constructible dynamique nous pouvons considérer des calculs faisant intervenir des matrices dont les coefficients sont des paramètres. En particulier nous pouvons utiliser les méthodes exposées dans la première partie de ce mémoire pour calculer la forme canonique de Jordan d'une telle matrice. Cela correspond à l'utilisation simultanée des clôtures algébrique et constructible. Aucun système de calcul ne permet en ce moment, l'obtention d'un tel résultat.

Autres applications Bien d'autres applications de la clôture constructible dynamique sont possibles, par exemple dans le domaine de la robotique elle a été déjà utilisée “à la main” [GR].

Pour finir cette introduction, nous remarquons que les méthodes proposées ici sont valables sur un corps quelconque. Elles sont programmées dans le système de calcul formel Axiom [JS]. Le choix de ce langage est très important. En particulier, grâce à sa *généricité*, les programmes implantés sont eux aussi valables sur un corps quelconque. Cela s'est traduit par exemple, par la possibilité d'utiliser simultanément les clôtures algébrique et constructible.

Références

[Bu] B. Buchberger. – *Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory.* Recent Trends in Multidimensional System Theory, Bose (ed.), (Reidel, 1985).

[BCK] B. Buchberger, G.E. Collins, B. Kutzler. – *Algebraic Methods for Geometry Reasoning.* Ann. Rev. Comput. Sci. 3, (1988).

[CY] S. C. Chou, J. G. Yang. – *On the Algebraic Formulation of Certain Geometry Statements and Mechanical Geometry Theorem Proving.* Algorithmica 4 p. 237-262 (1989).

[DDD] J. Della Dora, C. Dicrescenzo, D. Duval. – *About a New Method for Computing in Algebraic Number Fields.* Eurocal'85, vol.2, Springer Lecture Notes in Computer Science 204, ed. G. Goos, J. Hartmanis, p. 289–290 (1985).

[DD] C. Dicrescenzo, D. Duval. – *Algebraic Extensions and Algebraic Closure in Scratchpad.* Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 358, ed. P. Gianni, p. 440–446 (1989).

[DR] D. Duval, J.-C. Reynaud. – *Sketches and Computation (Part I): Basic Definitions and Static Evaluation* et *(Part II): Dynamic Evaluation and Applications.* Mathematical Structures in Computer Science, à paraître.

[Gil] I. Gil – *Contribution à l'Algèbre Linéaire Formelle. Formes normales de matrices et applications.* Thèse de l'Institut National Polytechnique de Grenoble (1993).

[GR] M.J. González-López, T. Recio. – *The ROMIN Inverse Geometric Model and the Dynamical Evaluation Method.* The Scafi papers draft, p. 117-141 (1991).

[Kal] M. Kalkbrener. – *A Generalized Euclidean Algorithm for Computing Triangular Representations of Algebraic Varieties.* Journal of Symbolic Computation 15, p.143-167 (1993).

[JS] R. D. Jenks, R. S. Sutor. – *Axiom, The Scientific Computation System.* Springer-Verlag (1992).

[La85] D. Lazard – *Maniement des Nombres Algébriques : Le calcul de l'exponentielle d'une matrice est–il exponentiel ?.* Séminaire d'informatique théorique, L.I.T.P. Univ. Paris VI (1985).

[La91] D. Lazard. – *A New Method for Solving Algebraic Systems of Positive Dimension.* Discrete Applied Mathematics 33, p. 147-160 (1991).

[MO] K. Martin, J. M. Olazabal – *An algorithm to compute the change basis for the rational form of K-endomorphisms.* Extracta Mathematicae, 6, n. 2-3, p. 142-144 (1992).

[ML] T. M. L. Mulders, A. H. M. Levelt – *A Maple package for the Computation of Normal Forms of Matrices.* Maple Share Library (1993).

[Oz] P. Ozello – *Calcul exact des Formes de Jordan et de Frobenius d'une matrice.* Thèse de l'Université Scientifique et Médicale de Grenoble (1987).

[RV] J. L. Roch, G. Villard – *Fast Parallel Computation of the Jordan Normal Form of Matrices.* Parallel Processing Letters, to appear (1994).

[Se] A. Seidenberg. – *An Elimination Theory for Differential Algebra.* Univ. of California Publications in Math 3 p. 31-65 (1954).

Partie I
Calcul de la forme canonique de Jordan

Introduction

L'origine de cette étude est la programmation du calcul de la forme canonique de Jordan en utilisant l'évaluation dynamique, ou plus précisément, le programme de la clôture algébrique dynamique – en fait, la clôture algébrique dynamique est le programme qui applique la technique d'évaluation dynamique aux calculs avec des nombres algébriques.

Pour utiliser efficacement la clôture algébrique dynamique il faut montrer que, à chaque moment du calcul, il y a une *seule* racine du polynôme (non nécessairement irréductible) qui intervient ; le programme gère automatiquement la distinction des racines avec des comportements différents.

Le calcul de la forme canonique de Jordan d'une matrice à coefficients dans un corps (commutatif) vérifie cette condition : en général toute explication de ce calcul commence par *Soit* λ *une valeur propre ...*, c'est-à-dire *une racine quelconque* du polynôme ... Ce calcul comprend le calcul de la forme canonique et d'une matrice de passage de la matrice donnée à la forme canonique.

Mais ce qui n'est pas très connu en général, c'est que le calcul de la matrice inverse de cette matrice de passage vérifie aussi cette condition. Ce résultat est montré pour première fois (semble-t-il) dans la thèse d'Ozello (p.25) [Oz, Laz] :

> *"Bien que les éléments de* P^{-1} *appartiennent à l'extension* $\mathbf{K}(\lambda_1, \ldots, \lambda_p)$*, nous avons montré que dans chaque colonne* (ligne dans notre cas) *de* P^{-1}*, chaque élément s'exprime comme une expression polynomiale en un seul des nombres* λ_k*."*

Nous donnons deux autres preuves de ce résultat. Nous montrons aussi, de deux façons différentes, un résultat un peu plus général en considérant la décomposition de l'espace total comme somme directe de sous-espaces caractéristiques. D'où on peut obtenir le résultat d'Ozello comme le cas particulier dans lequel on considère le calcul de la forme canonique de Jordan.

Chacune de ces deux preuves consiste essentiellement en une méthode de calcul de la matrice inverse de la matrice de passage. La première méthode utilise la résolution des systèmes d'équations linéaires. La deuxième méthode utilise le calcul des bases duales.

Nous remarquons que nous nous intéressons au calcul direct de la forme canonique de Jordan d'une matrice donnée. Pour d'autres méthodes pour obtenir cette forme canonique voir [Oz, ML, Gil, MO].

Finalement, les algorithmes obtenus sont valables pour n'importe quelle description des nombres algébriques, mais ils ont été programmés seulement dans le cadre de la clôture algébrique dynamique.

Le plan de ce travail est le suivant :

- Dans le premier chapitre, nous rappelons les concepts nécessaires, nous fixons des notations et nous introduisons des théorèmes fondamentaux qui seront démontrés aux sections suivantes.
- Dans le deuxième chapitre, nous exposons une méthode classique pour le calcul de la forme canonique et d'une matrice de passage. Pour cela nous utilisons les concepts de *famille de sous-espaces générateurs* et de *tours*, plus ou moins disséminés dans la littérature.
- Dans le troisième chapitre, nous étudions le calcul de la matrice inverse d'une matrice de passage dans un cadre un peu plus général que celui de Jordan, en considérant la décomposition de l'espace total comme somme directe de sous-espaces caractéristiques.
- Dans le quatrième chapitre, nous étudions le calcul de la matrice inverse d'une matrice de passage dans le cas de Jordan en particulier. Cela nous conduit a étudier des relations d'orthogonalité entre certains sous-espaces.
- Dans le cinquième chapitre, nous décrivons les algorithmes obtenus (et programmés) ainsi que la liaison avec la clôture algébrique dynamique. Nous présentons aussi des exemples.

1 Matrice de Jordan, une matrice de passage et son inverse : préliminaires

Dans cette section nous rappelons les concepts nécessaires et nous fixons des notations : ils peuvent être trouvés dans [FF, La, LFA, CHO, Ne], par exemple.

Nous introduisons aussi les théorèmes fondamentaux qui seront démontrés dans les prochaines sections.

Commençons par rappeler un résultat d'algèbre linéaire.

Théorème 1.1 *Soit $\mathbf{K}$ un corps quelconque (commutatif). Soit $\mathbf{E}$ un espace vectoriel sur $\mathbf{K}$ de dimension finie, et u un endomorphisme de $\mathbf{E}$.*

1. *Soit $\mathbf{E} = \mathbf{E}_1 \oplus \ldots \oplus \mathbf{E}_m$ une décomposition de $\mathbf{E}$ stable par u, c'est-à-dire, telle que $u(\mathbf{E}_i) \subseteq \mathbf{E}_i$ pour tout i. Alors dans toute base $B = B_1 \cup \ldots \cup B_m$ de $\mathbf{E}$*

adaptée à cette décomposition, la matrice U de u est de la forme :

$$U = \begin{pmatrix} U_1 & | & & 0 \\ --- & & & \\ & & \ddots & \\ & & & --- \\ 0 & & | & U_m \end{pmatrix}$$

où U_i est la matrice qui représente l'endomorphisme $u|_{\mathbf{E}_i}$ de $\mathbf{E}_i$ dans la base B_i.

2. *Réciproquement, soit u est un endomorphisme de $\mathbf{E}$ qui se représente par une telle matrice U dans une base $B = B_1 \cup \ldots \cup B_m$ de $\mathbf{E}$ (où le cardinal de B_i est égal à la dimension de U_i). Notons $\mathbf{E}_i$ le sous-espace vectoriel de $\mathbf{E}$ engendré par B_i. Alors la décomposition $\mathbf{E} = \mathbf{E}_1 \oplus \ldots \oplus \mathbf{E}_m$ est stable par u.*

1.1 Matrice de Jordan

Soit $\mathbf{K}$ un corps quelconque (commutatif) et $\overline{\mathbf{K}}$ sa clôture algébrique.

Notons $M_n = M_{n\times n}(\mathbf{K})$ l'ensemble des matrices carrées de dimension $n \in \mathbf{N}$ sur le corps $\mathbf{K}$ et de la même façon $\overline{M}_n = M_{n\times n}(\overline{\mathbf{K}})$ sur $\overline{\mathbf{K}}$. Nous notons par I_j la matrice identité de dimension j et par I la matrice identité de dimension n.

Nous nous plaçons dans l'espace vectoriel $\overline{\mathbf{K}}^n$. Posons $(e_1, \ldots, e_n)$ la base canonique de $\overline{\mathbf{K}}^n$.

Parfois nous considérons les éléments de l'espace comme des éléments de $M_{n\times 1}(\overline{\mathbf{K}})$, c'est-à-dire comme des vecteurs colonnes de dimension n. De même, parfois nous considérons une suite de vecteurs $(v_1, \ldots, v_l)$ comme une matrice dans $M_{n\times l}(\overline{\mathbf{K}})$.

Soit $A \in M_n$ et soit $p_c(x) = det(A - xI) \in \mathbf{K}[x]$ son *polynôme caractéristique.* Notons $\lambda_1, \ldots, \lambda_p$ les *valeurs propres* de la matrice A, c'est-à-dire les p racines **distinctes** de $p_c(x)$ dans $\overline{\mathbf{K}}$. On a $p_c(x) = (x-\lambda_1)^{r_1} \ldots (x-\lambda_p)^{r_p}$ dans $\overline{\mathbf{K}}[x]$ et nous appelons r_k la *multiplicité caractéristique* de λ_k, $k = 1, \ldots, p$.

Soit $p_m(x)$ le *polynôme minimal* de A. Par le théorème de Hamilton–Cayley on a $p_m(x) = (x-\lambda_1)^{s_1} \ldots (x-\lambda_p)^{s_p} \in \overline{\mathbf{K}}[x]$, et s_k s'appelle la *multiplicité minimale* de λ_k, $k = 1, \ldots, p$; elle vérifie $1 \leq s_k \leq r_k$.

Notations Soit $\lambda \in \{\lambda_1, \ldots, \lambda_p\}$; on pose :

- r la multiplicité caractéristique de λ,
- s la multiplicité minimale de λ,
- $N_i = Ker(A - \lambda I)^i$ le *sous-espace caractéristique* associé à λ et à i, $i = 0, 1, \ldots, s$;
  - $N_0 = 0$.

– N_1 est le *sous-espace propre* associé à λ. Si $v \in N_1$ alors v est appelé un *vecteur propre* associé à λ.
– Parfois nous noterons N_s par $N(\lambda)$. La dimension de N_s est r.

A représente (dans la base canonique de $\overline{\mathbf{K}}^n$) un endomorphisme

$$a : \overline{\mathbf{K}}^n \to \overline{\mathbf{K}}^n \text{ défini par } a(v) = Av$$

Cet endomorphisme a (ou la matrice A) fournit une décomposition stable par a de l'espace total :

$$\overline{\mathbf{K}}^n = N(\lambda_1) \oplus \ \dots \ \oplus N(\lambda_p).$$

Donc la matrice A est semblable dans $\overline{\mathbf{K}}^n$ à une matrice $J \in M_n(\overline{\mathbf{K}})$ avec

$$J = \begin{pmatrix} J(\lambda_1) & | & & 0 \\ --- & & & \\ & & \ddots & \\ & & & --- \\ 0 & & | & J(\lambda_p) \end{pmatrix}$$

Pour chaque $\lambda \in \{\lambda_1, \ \dots, \ \lambda_p\}$, le sous-espace $N(\lambda)$ a une décomposition stable par a (comme expliqué dans 2.2.2) :

$$N(\lambda) = T_1 \oplus \ \dots \ \oplus T_{m(\lambda)}$$

donc $J(\lambda)$ peut s'écrire :

$$J(\lambda) = \begin{pmatrix} j_1(\lambda) & | & & 0 \\ --- & & & \\ & & \ddots & \\ & & & --- \\ 0 & & | & j_{m(\lambda)}(\lambda) \end{pmatrix} \in M_r(\mathbf{K}(\lambda))$$

Notons $d_l = dim(T_l)$, alors $d_1 + \ \dots \ + d_{m(\lambda)} = r$.

De plus (voir 2.2.2) T_l a une base $t_l = (v^1, \ v^2, \ \dots v^{d_l})$ telle que $a(v^i) = v^{i-1} + \lambda v^i$, $i = 1, \ \dots, \ l$, avec $v^0 = 0$. Donc dans cette base, $j_l(\lambda) = j(\lambda, d_l)$ où :

$$j(\lambda, d) = \begin{pmatrix} \lambda & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{pmatrix} \in M_d(\mathbf{K}(\lambda))$$

On appelle J la *forme de Jordan* pour la matrice A. Nous appelons $j(\lambda, d)$ *matrice élémentaire de Jordan* de dimension d, et $J(\lambda)$ *matrice générale de Jordan* pour λ.

Le nombre $m(\lambda)$ de $j(\lambda, d)$ dans $J(\lambda)$ coïncide avec la dimension du sous-espace propre N_1 (appelée la *multiplicité géométrique* de λ) et il y a au moins un bloc $j(\lambda, s)$ de dimension s ; tous les autres blocs sont de dimension au plus s.

La matrice J est unique sauf par la disposition des blocs $j(\lambda_k, d_l)$ dans sa diagonale principale. Nous fixons par ordre de dimension décroissante la disposition des blocs $j(\lambda, d)$ dans $J(\lambda)$, ainsi la forme canonique de Jordan dépend uniquement de la disposition des blocs $J(\lambda_k)$, c'est-à-dire *de l'ordre de description des racines* de $p_c(x)$.

1.2 Matrice de passage

Dire que A est semblable à J signifie qu'il existe une matrice $P \in \overline{M}_n$ avec $det(P) \neq 0$ telle que

$$P^{-1}AP = J.$$

La matrice P s'appelle *matrice de passage* entre A et J. Les colonnes d'une telle matrice P forment une base B_J de l'espace $\overline{\mathbf{K}}^n$ dans laquelle l'endomorphisme a est représenté par J.

Cette base B_J est composée de bases B_{λ_k} des espaces $N(\lambda_k)$, $k = 1, \ldots, p$, telles que chaque endomorphisme

$$a|_{N(\lambda_k)} : N(\lambda_k) \to N(\lambda_k)$$

est représenté par $J(\lambda_k)$ dans la base B_{λ_k}, $k = 1, \ldots, p$.

Remarque Il y a en général plusieurs matrices P vérifiant $P^{-1}AP = J$.

Définition Soit $M \in M_n$, nous dirons que M *est conforme à* A si M est composée par p blocs–colonne

$$M = \left(\begin{array}{c|c|c} M(\lambda_1) & \ldots & M(\lambda_p) \end{array} \right)$$

avec $M(\lambda_k) \in M_{n \times r_k}(\mathbf{K}(\lambda_k))$, $k = 1, \ldots, p$.

Remarque En particulier, une matrice M est conforme à A si et seulement si elle est conforme à la transposée de A, A^t.

Théorème 1.2 *Il existe une matrice de passage entre A et J conforme à A.*

Ce résultat est classique. Nous donnons une démonstration dans la section 2.

1.3 Inverse de la matrice de passage

Soit J la forme canonique de Jordan de la matrice A. On construit ici une matrice R_J de la manière suivante :

- R_J a p blocs carrés dans sa diagonale principale

$$R_J = \begin{pmatrix} R_{J(\lambda_1)} & | & & & 0 \\ --- & & & & \\ & & \ddots & & \\ & & & & --- \\ 0 & & & | & R_{J(\lambda_p)} \end{pmatrix}$$

- pour chaque $\lambda \in \{\lambda_1, \ldots, \lambda_p\}$, $R_{J(\lambda)}$ a $m(\lambda)$ blocs carrés dans sa diagonale principale :

$$R_{J(\lambda)} = \begin{pmatrix} R_{d_1} & | & & & 0 \\ --- & & & & \\ & & \ddots & & \\ & & & & --- \\ 0 & & & | & R_{d_{m(\lambda)}} \end{pmatrix}$$

- chaque bloc R_d est une matrice carrée de dimension d avec des 1 dans sa diagonale secondaire et des 0 ailleurs :

$$\begin{pmatrix} 0 & & 1 \\ & \cdot^{\cdot^{\cdot}} & \\ 1 & & 0 \end{pmatrix}$$

Lemme 1.3 *Les matrices R_J et R_d vérifient :*

- $R_d{}^{-1} = R_d{}^t = R_d$ *et* $R_J{}^{-1} = R_J{}^t = R_J$
- $R_J J^t R_J = J$
- *Soit* $B = (b_{i,j})_{i,j=1,\ldots,d} \in M_d$*, alors*
  - $R_d B = (b_{d+1-i,j})_{i,j=1,\ldots,d}$ *(change l'ordre des lignes)*
  - $B R_d = (b_{i,d+1-j})_{i,j=1,\ldots,d}$ *(change l'ordre des colonnes)*
- *Si J est une matrice diagonale, alors R_J est la matrice identité.*

Lemme 1.4 *Soit P une matrice de passage entre A et J alors la matrice $(P^{-1})^t R_J$ est une matrice de passage entre A^t et J.*

Démonstration
On a $P^{-1}AP = J$. Si l'on considère la transposée, on en déduit $P^t A^t (P^{-1})^t = J^t$, et alors

$$(R_J P^t) A^t ((P^{-1})^t R_J) = R_J J^t R_J = J$$

Remarque La différence entre la matrice $(P^{-1})^t$ et la matrice de passage entre A^t et J, $(P^{-1})^t R_J$, est seulement l'ordre de disposition des colonnes.

Le résultat suivant est montré dans la thèse d'Ozello. Nous en donnons deux autres preuves dans la section 4. D'autre part, il est également possible d'obtenir ce résultat comme conséquence du théorème 3.1.

Théorème 1.5 *Soit P une matrice de passage entre A et J conforme à A, alors la matrice $(P^{-1})^t$ est conforme à A.*

2 Calcul de la forme canonique de Jordan et d'une matrice de passage conforme

Nous exposons ici une méthode classique pour le calcul de la matrice de Jordan et d'une matrice de passage entre A et J. La matrice de passage ainsi obtenue vérifie la propriété d'être conforme à A.

Ce type de méthode peut être trouvé dans la littérature classique, mais nous proposons ici la variante que nous avons implantée. Nous l'exposons de façon très détaillée pour diverses raisons :

- d'une part, cela nous permet de fixer les notations et concepts qui seront utilisés dans les sections suivantes
- d'autre part, les algorithmes que l'on utilise pour le calcul de la matrice inverse de la matrice de passage sont similaires à celui-ci.

2.1 Calcul indépendant pour chaque valeur propre

Soit $A \in M_n$, $p_c(x) \in \mathbf{K}[x]$ son polynôme caractéristique et $\lambda_1, \ldots, \lambda_p$ les valeurs propres de la matrice A, c'est-à-dire les p racines distinctes de $p_c(x)$ dans $\overline{\mathbf{K}}$. On a

$$J = \left(\begin{array}{ccc} J(\lambda_1) \;| & 0 & 0 \\ --- & & \\ 0 & \ddots & 0 \\ & & --- \\ 0 & 0 & |\; J(\lambda_p) \end{array}\right) \quad \text{et } P = \left(\begin{array}{c|c|c} & & \\ P(\lambda_1) & \ldots & P(\lambda_p) \\ & & \end{array}\right)$$

où $J(\lambda_k)$ est la matrice générale de Jordan correspondante à λ_k et les colonnes de $P(\lambda_k)$ forment une base de $N(\lambda_k)$, $k = 1, \ldots, p$.

Pour calculer J et P, il suffit de calculer $J(\lambda_k)$ et $P(\lambda_k)$ pour chaque k, $k = 1, \ldots, p$.

Donc, à partir de ce moment nous considérons dans cette section λ une valeur propre de la matrice A, c'est-à-dire une racine **quelconque** du polynôme caractéristique.

On sait déjà que $J(\lambda) \in M_r(\mathbf{K}(\lambda))$ et que $P(\lambda) \in \overline{M}_{n\times r}$. Nous allons voir comment l'on peut calculer $J(\lambda)$ et $P(\lambda)$ de sorte que les coefficients de $P(\lambda)$ soient dans $\mathbf{K}(\lambda)$.

2.2 Structure du sous-espace caractéristique $N(\lambda)$. Calcul de $J(\lambda)$ et $P(\lambda)$

Définition Nous appelons *base de passage de* $N(\lambda)$ toute base de $N(\lambda)$ telle que l'endomorphisme $a|_{N(\lambda)} : N(\lambda) \rightarrow N(\lambda)$ est représenté dans cette base par une matrice générale de Jordan.

Notations

- $a_\lambda : N(\lambda) \rightarrow N(\lambda)$ est l'endomorphisme de $N(\lambda)$ défini par $a_\lambda(v) = (A - \lambda I)v$
- Rappelons que $N_i = Ker(A - \lambda I)^i$. Posons $n_i = dim(N_i)$, $i = 0, 1, \ldots, s$.

Lemme 2.1 *Les noyaux N_i vérifient les relations suivantes (où le symbole $\subset$ note l'inclusion stricte) :*

$$N_0 = 0 \subset N_1 \subset \ \ldots \ \subset N_{s-1} \subset N_s = N_{s+1}$$

avec $dim(N_s) = r$ et $N_i = N_s$ pour tout $i \geq s$.

Remarque On en déduit :

$$n_0 = 0 < n_1 < \ \ldots \ < n_{s-1} < n_s.$$

Définition Soit $v \in N_i$ tel que $v \notin N_{i-1}$, nous disons que v est un vecteur caractéristique *de niveau* i.

Remarque Soit v un vecteur caractéristique de niveau i, avec $i \geq 1$ alors

- $v \neq 0$
- $a_\lambda(v)$ est un vecteur caractéristique de niveau $i - 1$.

2.2.1 Sous-espaces générateurs

Lemme 2.2 *Soit i un entier tel que $s > i \geq 1$, et soit H un sous-espace vectoriel de N_{i+1} tel que $H \cap N_i = 0$. Alors*

- $a_\lambda |_H : H \rightarrow N_s$ *est injective*
- $a_\lambda(H) \subseteq N_i$

- $a_\lambda(H) \cap N_{i-1} = 0$

Le théorème suivant nous montre la structure du sous-espace $N(\lambda)$. Nous voyons qu'il suffit de connaître une famille $(G_s, \ \ldots, \ G_1)$ de sous-espaces de $N(\lambda)$ pour connaître $N(\lambda)$.

Théorème 2.3 *Il existe une famille ordonnée $(G_s, \ \ldots, \ G_1)$ de sous-espaces de N_s telle que, en notant $g_i = dim(G_i)$:*

- $g_s \geq 1$
- *pour tout $i = s, \ \ldots, \ 1$*

$$N_i = N_{i-1} \oplus (\oplus_{j=i}^{s} a_\lambda^{j-i}(G_j))$$
$$n_i = n_{i-1} + (\textstyle\sum_{j=i}^{s} g_j)$$

Remarque Nous écrivons ici les formules de cet énoncé de façon étendue pour améliorer la compréhension de la démonstration.

$$N_s = N_{s-1} \oplus G_s$$

$$N_{s-1} = N_{s-2} \oplus a_\lambda(G_s) \oplus G_{s-1}$$

$$\vdots$$

$$N_2 = N_1 \oplus a_\lambda^{s-2}(G_s) \oplus \ \ldots \ \oplus a_\lambda(G_3) \oplus G_2$$

$$N_1 = \qquad a_\lambda^{s-1}(G_s) \oplus \ \ldots \ \oplus a_\lambda^2(G_3) \oplus a_\lambda(G_2) \oplus G_1$$

Démonstration
Si $s = 1$ alors $G_1 = N_1$. Supposons $s > 1$.
Construction de G_s
Soit G_s un supplémentaire de N_{s-1} dans N_s. Un tel sous-espace G_s vérifie :

- $N_s = N_{s-1} \oplus G_s$
- $g_s = n_s - n_{s-1} \geq 1$

Hypothèse d'induction pour $i+1$, avec $s - 1 > i \geq 1$
Supposons la famille $(G_s, \ \ldots, \ G_{i+1})$ construite. On a :

- $N_{i+1} = N_i \oplus (\oplus_{j=i+1}^{s} a_\lambda^{j-(i+1)}(G_j))$
- $n_{i+1} = n_i + (\sum_{j=i+1}^{s} g_j)$

Construction de G_i
Posons $H = (\oplus_{j=i+1}^{s} a_{\lambda}^{j-(i+1)}(G_j))$. On a $N_{i+1} = N_i \oplus H$, d'où $N_i \cap H = 0$. En utilisant le lemme précédent, on a $a_{\lambda}\,|_H \colon H \to N_s$ injective, $a_{\lambda}(H) \subseteq N_i$ et $N_{i-1} \cap a_{\lambda}(H) = 0$.

Soit G_i un supplémentaire de $N_{i-1} \oplus a_{\lambda}(H)$ dans N_i, (dans le cas où $N_{i-1} \oplus a_{\lambda}(H) = N_i$ on a $G_i = 0$). Un tel sous-espace G_i vérifie :

- $N_i = N_{i-1} \oplus (\oplus_{j=i}^{s} a_{\lambda}^{j-i}(G_j))$
- $n_i = n_{i-1} \oplus (\sum_{j=i}^{s} g_j)$

Remarque Pour la construction du sous-espace G_i, on considère un supplémentaire *quelconque* de $N_{i-1} \oplus a_{\lambda}^{s-i}(G_s) \oplus \ldots \oplus a_{\lambda}^{2}(G_{i+2}) \oplus a_{\lambda}(G_{i+1})$ dans N_i.

Corollaire 2.4 *Les dimensions des sous-espaces G_i, $i = s, \ldots, 1$ vérifient :*

- $g_s + g_{s-1} + \ldots + g_1 = n_1$
- $sg_s + (s-1)g_{s-1} + \ldots + g_1 = n_s = r$

Définition Soit $(G_s, \ldots, G_1)$ une famille ordonnée de sous-espaces de N_s vérifiant les conditions du théorème précédent. Nous appelons cette famille *une famille de sous-espaces générateurs de N_s*. Nous appelons G_i un *sous-espace générateur de niveau i*. Si $v \in G_i$, $v \neq 0$, nous dirons que v est un *vecteur générateur de niveau i*.

Notation Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs dans N_s, posons :

$$M_s = G_s$$

$$M_{s-1} = a_{\lambda}(G_s) \oplus G_{s-1}$$

$$\vdots$$

$$M_1 = a_{\lambda}^{s-1}(G_s) \oplus \ldots \oplus a_{\lambda}^{2}(G_3) \oplus a_{\lambda}(G_2) \oplus G_1$$

et $m_i = dim(M_i)$, $i = s, \ldots, 1$,

Corollaire 2.5 *Les sous-espaces M_i, $i = s, \ldots, 1$ vérifient :*

- $N_i = M_i \oplus N_{i-1}$, $i = s, \ldots, 1$
- $N_s = M_s \oplus \ldots \oplus M_2 \oplus M_1$

et pour leurs dimensions on a :

- $m_i = n_i - n_{i-1} \geq 0$
- $g_s = m_s \geq 1$
- $g_i = m_i - m_{i+1} \geq 0$, $i = s-1, \ldots, 1$

Remarque Soit $v \in M_i$, $v \neq 0$ alors v est un vecteur caractéristique de niveau i. En particulier un vecteur générateur de niveau i est un vecteur caractéristique de niveau i.

2.2.2 Tours

Définition Soit v^i un vecteur caractéristique de niveau i. On note $v^{i-j} = a_\lambda^j(v^i)$, $j = 1, \ldots, i-1$. Alors le niveau de v^{i-j} est $i-j$. Nous appelons la *tour de hauteur i engendrée par v^i* l'ensemble ordonné de vecteurs :

$$t^i = (v^1, \ldots, v^{i-1}, v^i).$$

Lemme 2.6 *Les éléments d'une tour sont linéairement indépendants.*

Démonstration
Soit $t^i = (v^1, \ldots, v^{i-1}, v^i)$ une tour de hauteur i engendrée par le vecteur caractéristique v^i de niveau i. Soit :

$$w = \alpha_1 v^1 + \ldots + \alpha_{i-1} v^{i-1} + \alpha_i v^i$$

et supposons $w = 0$, alors :

$0 = a_\lambda^{i-1}(w) = \alpha_i a_\lambda^{i-1}(v^i)$ et donc $\alpha_i = 0$

Supposons $\alpha_i = \ldots = \alpha_{j+1} = 0$, alors :

$0 = a_\lambda^{j-1}(w) = \alpha_j a_\lambda^{j-1}(v^j)$ et donc $\alpha_j = 0$

Lemme 2.7 *Soit v^i et w^j deux vecteurs générateurs de niveaux respectifs i et j avec $i > j$. Alors v^k et w^k sont linéairement indépendants pour tout k tel que $1 \leq k \leq j$.*

Démonstration
Soit $\alpha v^k + \beta w^k = 0$. Alors $\alpha v^k + \beta w^k \in a_\lambda^{i-k}(G_i) \oplus a_\lambda^{j-k}(G_j)$ (théorème 2.3) donc $\alpha = \beta = 0$.

Notation et remarque Soit T^i un sous-espace de N_s engendré par les éléments d'une tour de hauteur i. On a :

- $dim(T^i) = i$
- $a(T^i) \subseteq T^i$

Lemme 2.8 *Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs dans N_s. Soit T_1^i et T_2^j deux sous-espaces de N_s engendrés respectivement par $t_1^i = (v^1, \ldots, v^{i-1}, v^i)$ et $t_2^j = (w^1, \ldots, w^{j-1}, w^j)$ avec $v^i \in G_i$ et $w^j \in G_j$ deux vecteurs générateurs de niveaux i et j. Supposons que si $i = j$, v^i et w^j sont linéairement indépendants. Alors $T_1^i \cap T_2^j = 0$.*

Démonstration
Soit $w \in T_1^i \cap T_2^j$. Alors nous pouvons écrire w comme

$$\begin{aligned} w &= \alpha_1 v^1 + \ldots + \alpha_{i-1} v^{i-1} + \alpha_i v^i \\ &= \beta_1 w^1 + \ldots + \beta_{j-1} w^{j-1} + \beta_j w^j \end{aligned}$$

d'où, si nous supposons $i \geq j$

$$0 = (\alpha_1 v^1 - \beta_1 w^1) + \ldots + (\alpha_j v^j - \beta_j w^j) + \alpha_{j+1} v^{j+1} + \ldots + \alpha_i v^i \tag{1}$$

Si $i > j$ alors

$$\begin{aligned} 0 &= a_\lambda^{i-1}[(\alpha_1 v^1 - \beta_1 w^1) + \ldots + (\alpha_j v^j - \beta_j w^j) + \alpha_{j+1} v^{j+1} + \ldots + \alpha_i v^i] \\ &= \alpha_i a_\lambda^{i-1}(v^i) \end{aligned}$$

d'où $\alpha_i = 0$.

De même, on obtient $\alpha_{j+1} = \ldots = \alpha_{i-1} = 0$.

L'égalité (1) s'écrit maintenant, pour tout $i \geq j$

$$0 = (\alpha_1 v^1 - \beta_1 w^1) + \ldots + (\alpha_j v^j - \beta_j w^j)$$

et donc

$$\begin{aligned} 0 &= a_\lambda^{j-1}[(\alpha_1 v^1 - \beta_1 w^1) + \ldots + (\alpha_j v^j - \beta_j w^j)] \\ &= a_\lambda^{j-1}[(\alpha_j v^j - \beta_j w^j)] \end{aligned}$$

Si $i = j$ alors $(\alpha_j v^j - \beta_j w^j)$ appartient à $N_{j-1} \cap G_j$ qui est nul par le théorème 2.3 et donc $(\alpha_j v^j - \beta_j w^j) = 0$. Par hypothèse v^j et w^j son linéairement indépendants d'où $\alpha_j = \beta_j = 0$.

Si $i > j$ alors $(\alpha_j v^j - \beta_j w^j)$ appartient à $N_{j-1} \cap [a_\lambda^{i-j}(G_i) \oplus G_j]$ qui est nul par le théorème 2.3 et donc $(\alpha_j v^j - \beta_j w^j) = 0$. Par le lemme 2.7 on a que les vecteurs v^j et w^j sont linéairement indépendants d'où $\alpha_j = \beta_j = 0$.

De même, on obtient $\alpha_{j-1} = \beta_{j-1} = 0, \ \ldots, \ \alpha_1 = \beta_1 = 0$.

Remarque Soit G_i un sous-espace générateur de niveau i et v^i un élément non nul de G_i. Soit $t^i = (v^1, \ \ldots, \ v^{i-1}, \ v^i)$ une tour de hauteur i engendrée par v^i. Soit T^i le sous-espace de N_s engendré par les vecteurs de t^i. L'endomorphisme

$$a|_{T^i} : \ T^i \ \rightarrow \ T^i$$

se représente par la matrice élémentaire de Jordan $j(\lambda, i)$ dans la base $(v^1, \ \ldots, \ v^{i-1}, \ v^i)$.

Notation Soit G_i un sous-espace générateur de niveau i. Nous noterons une base de G_i par

$$(v^i_{m_{i+1}+1}, \ldots, v^i_{m_i})$$

avec $i = s, \ldots, 1$ et $m_{s+1} = 0$.

Théorème 2.9

1. *Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs de N_s. Pour chaque $i = s, \ldots, 1$ soit $(v^i_{m_{i+1}+1}, \ldots, v^i_{m_i})$ une base de G_i (avec $m_{s+1} = 0$) et pour chaque j, $j = m_{i+1}+1, \ldots, m_i$, soit t^i_j la tour de hauteur i engendrée par v^i_j. Alors l'ensemble*

$$t^s_1 \cup \ldots \cup t^s_{m_s} \cup t^{s-1}_{m_s+1} \cup \ldots \cup t^{s-1}_{m_{s-1}} \cup \ldots \cup t^1_{m_2+1} \cup \ldots \cup t^1_{m_1}$$

   (sans changer l'ordre des vecteurs !) est une base de passage de N_s.

2. *Toute base de passage de N_s peut être construite de cette façon à partir d'une famille de sous-espaces générateurs $(G_s, \ldots, G_1)$ de N_s.*

Démonstration

1. Posons T^i_j le sous-espace de N_s engendré par les vecteurs de la tour t^i_j, $i = s, \ldots, 1$, $j = m_{i+1}+1, \ldots, m_i$. Alors

$$N_s = T^s_1 \oplus \ldots \oplus T^s_{m_s} \oplus T^{s-1}_{m_s+1} \oplus \ldots \oplus T^{s-1}_{m_{s-1}} \oplus \ldots \oplus T^1_{m_1}$$

   Nous en déduisons que dans la base indiquée, l'endomorphisme

$$a|_{N_s} : N_s \to N_s$$

   se représente par une matrice générale de Jordan $J(\lambda)$. Pour chaque $i = s, \ldots, 1$, la matrice $J(\lambda)$ a g_i matrices élémentaires de Jordan $j(\lambda, i)$.

2. Soit B une base de passage de N_s. D'après la forme de $J(\lambda)$, nous savons que B est la réunion de :

   g_s familles libres de dimension s que l'on note $(t^s_1, \ldots, t^s_{m_s})$

   $\vdots$

   g_1 familles libres de dimension 1 que l'on note $(t^1_{m_2+1}, \ldots, t^1_{m_1})$

   avec $g_s \geq 1$, et t^i_j est une tour de hauteur i. Notons v^i_j le vecteur caractéristique de niveau i qui engendre t^i_j, $i = s, \ldots, 1$, $j = m_{i+1}+1, \ldots, m_i$ (avec $m_{s+1} = 0$). Pour chaque $i = s, \ldots, 1$, soit G_i le sous-espace de N_s engendré par $(v^i_{m_{i+1}+1}, \ldots, v^i_{m_i})$. Alors on vérifie facilement que $(G_s, \ldots, G_1)$ est une famille de sous-espaces générateurs de N_s.

Remarque Le théorème précédent nous montre la bijection existant entre les bases de passage et les familles de sous-espaces générateurs de N_s.

Remarque *Calcul de la matrice générale de Jordan pour* λ
La matrice $J(\lambda)$ a g_i matrices élémentaires de Jordan de dimension i, pour chaque $i = s, \ldots, 1$. Cela nous montre qu'il est possible de connaître la matrice générale de Jordan pour λ à partir des dimensions des sous-espaces N_i, $i = s, \ldots, 1$.

Remarque *Calcul de* $P(\lambda) \in M_{n\times r}(\mathbf{K}(\lambda))$
Pour chaque i, $i = s, \ldots, 1$, nous pouvons calculer une base de N_i telle que les coefficients des vecteurs de cette base soient dans $\mathbf{K}(\lambda)$: en effet, par définition $N_i = Ker(A - \lambda I)^i$ et $(A - \lambda I)^i$ a ses coefficients dans $\mathbf{K}(\lambda)$.

D'après le procédé de construction utilisé dans la démonstration du théorème 2.3, nous pouvons ensuite déterminer une famille $(G_s, \ldots, G_1)$ de sous-espaces générateurs de N_s ayant la même propriété.

En conclusion, la matrice $P(\lambda)$ construite ainsi a ses coefficients dans $\mathbf{K}(\lambda)$, d'où le théorème 1.2. De plus tous les calculs correspondants peuvent se faire dans $\mathbf{K}(\lambda)$.

3 Calcul de la matrice inverse d'une matrice de passage : cas général

Considérons la matrice $A \in M_n$, l'endomorphisme a associé et la décomposition de l'espace total fournie par A :

$$\overline{\mathbf{K}}^n = N(\lambda_1) \oplus \ \ldots \ \oplus N(\lambda_p).$$

Nous avons vu que dans toute base $B = B_1 \cup \ldots \cup B_p$ de $\mathbf{E}$ adaptée à cette décomposition, l'endomorphisme a est représenté par une matrice

$$\overline{A} = \begin{pmatrix} \overline{A}_1 & | & & 0 \\ - - - & & & \\ & & \ddots & \\ & & & - - - \\ 0 & & | & \overline{A}_p \end{pmatrix}$$

avec $\overline{A}_k \in \overline{M}_{r_k}$.

Soit $\overline{P}$ une matrice de passage entre A et $\overline{A}$. Nous pouvons choisir $\overline{P}$ conforme à A, c'est-à-dire,

$$\overline{P} = \begin{pmatrix} & | & & | & \\ \overline{P}(\lambda_1) & | & \ldots & | & \overline{P}(\lambda_p) \\ & | & & | & \end{pmatrix}$$

avec $\overline{P}(\lambda_k) \in M_{r_k}(\mathbf{K}(\lambda_k))$, $k = 1, \ldots, p$.

Théorème 3.1 *La matrice* $(\overline{P}^{-1})^t$ *est conforme à* A.

Dans ce chapitre nous donnons deux démonstrations différentes de ce théorème. Chaque démonstration fournit une méthode de calcul de $\overline{P}^{-1}$.

La première méthode utilise la résolution des systèmes d'équations linéaires et elle a des relations avec le calcul d'inverses généralisées pour des matrices rectangulaires. La deuxième méthode utilise le calcul des bases duales, avec un algorithme du même style que le processus d'orthogonalisation de Gram-Schmidt.

Remarque Le théorème 1.5 est un cas particulier de ce théorème.

Notation Rappelons que nous avons noté $N_i = Ker(A - \lambda I)^i$ et $N(\lambda) = N_s$. Posons $\widetilde{N}_i = Ker(A^t - \lambda I)^s$ et $\widetilde{N}(\lambda) = \widetilde{N}_s$.

3.1 Calcul indépendant pour chaque valeur propre

Le théorème suivant nous montre pourquoi, pendant le calcul de la matrice inverse de la matrice de passage, il suffit de considérer une seule valeur propre à chaque fois.

Théorème 3.2 *Soit* λ, μ *deux valeurs propres de* A *avec* $\lambda \neq \mu$ *et soit* $v \in N(\lambda)$ *et* $w \in \widetilde{N}(\mu)$. *Alors* $w^t v = 0$.

Démonstration
Soit $v \in N(\lambda)$ et $w \in \widetilde{N}(\mu)$. Posons

$$x = (A - \lambda I)v$$

$$y = (A^t - \mu I)w$$

alors

$$w^t x = w^t[(A - \lambda I)v] = w^t A v - \lambda w^t v$$

$$y^t v = [(A^t - \mu I)w]^t v = w^t[(A - \mu I)v] = w^t A v - \mu w^t v$$

Si $w^t x = y^t v$ alors $w^t v = 0$ puisque $\lambda \neq \mu$.

Notons i (respectivement j) le plus petit indice tel que $v \in N_i$ (respectivement tel que $w \in \widetilde{N}_j$). Nous montrons le théorème par récurrence sur $i + j$:

- Si $i + j \leq 1$ alors $i = 0$ ou $j = 0$ et donc $v = 0$ ou $w = 0$, d'où $w^t v = 0$.
- Soit $i + j \geq 2$ et supposons la propriété vraie pour tous $(i_0,\ j_0)$ tels que $i_0 + j_0 < i + j$, alors $w^t x = 0$ (puisque $(i-1) + j < i + j$) et $y^t v = 0$ (puisque $i + (j-1) < i + j$), d'où $w^t v = 0$.

Corollaire 3.3 *Soit*

$$P = \left(\begin{array}{c|c|c} & & \\ P_1 & \ldots & P_p \\ & & \end{array} \right) \textit{ et } Q = \left(\begin{array}{c|c|c} & & \\ Q_1 & \ldots & Q_p \\ & & \end{array} \right)$$

avec $P_k \in \overline{M}_{n\times r_k}$ *contenant par colonnes une base (quelconque) de l'espace* $N(\lambda_k)$*, et* $Q_k \in \overline{M}_{n\times r_k}$ *contenant par colonnes une base (quelconque) de l'espace* $\widetilde{N}(\lambda_k)$ $k = 1, \ldots, p$. *Alors*

$$Q^t P = \begin{pmatrix} Q^t{}_1 P_1 & | & 0 & & 0 \\ - - - & & & & \\ 0 & & \ddots & & 0 \\ & & & & - - - \\ 0 & & 0 & | & Q^t{}_p P_p \end{pmatrix}$$

Maintenant nous pouvons **poser notre problème** comme suit :

> *Soit* $\overline{P}$ *une matrice de passage entre* A *et* $\overline{A}$ *conforme à* A. *Pour chaque* k, $k = 1, \ldots, p$ *existe–t–il une matrice* $\overline{Q}(\lambda_k) \in M_{n\times r_k}(\mathbf{K}(\lambda_k))$ *dont les colonnes forment une base de* $\widetilde{N}(\lambda_k)$ *et telle que*
>
> $$\overline{Q}(\lambda_k)^t \overline{P}(\lambda_k) = I_{r_k} \text{ ?}$$

Remarque En ce cas, si l'on pose

$$\overline{Q} = \left(\begin{array}{c|c|c} & & \\ \overline{Q}(\lambda_1) & \ldots & \overline{Q}(\lambda_k) \\ & & \end{array} \right)$$

on a $\overline{Q}^t = \overline{P}^{-1}$ avec $\overline{Q}$ conforme à A, ce qui montre le théorème 3.1.

Soit λ une valeur propre de la matrice A et $\overline{P}$ une matrice de passage entre A et $\overline{A}$ conforme à A. Notons $\overline{P}(\lambda)$ le bloc–colonne de $\overline{P}$ correspondant à λ. Nous allons montrer l'existence d'une base $B(\lambda)$ de $\widetilde{N}(\lambda)$ telle que la matrice $\overline{Q}(\lambda)$ construite à partir de $B(\lambda)$ vérifie $\overline{Q}(\lambda)^t\overline{P}(\lambda) = I_r$ et les coefficients de $\overline{Q}(\lambda)$ soient dans $\mathbf{K}(\lambda)$. Nous verrons aussi comment l'on peut calculer une telle base.

Remarque Dans le cas d'existence de ces bases, elles doivent être *uniques*, étant donné que l'on cherche la matrice inverse de la matrice $\overline{P}$.

Remarque Pour calculer la matrice $\overline{Q}(\lambda)$ nous cherchons une base de $\widetilde{N}(\lambda)$. Nous voyons dans ce chapitre que ce noyau est le seul sous-espace de $\overline{\mathbf{K}}^n$ qui vérifie les conditions nécessaires pour trouver une telle base.

3.2 Méthode de calcul de $B(\lambda)$ par résolution de systèmes linéaires

Nous voulons calculer une base $B(\lambda) = (w_1, \ldots, w_r)$ de $\widetilde{N}(\lambda)$ telle que la matrice $\overline{Q}(\lambda)$ construite à partir de $B(\lambda)$ vérifie $\overline{Q}(\lambda)^t \overline{P}(\lambda) = I_r$.

Notons $v_1, \ldots, v_r$ les colonnes de $\overline{P}(\lambda)$. Alors nous cherchons une base $(w_1, \ldots, w_r)$ de $\widetilde{N}(\lambda)$ telle que

$$\begin{array}{ll} w_i^t v_i = 1 & i = 1, \ldots, r \\ w_i^t v_j = 0 & i \neq j \end{array}$$

Lemme 3.4 *Soit* $\mathbf{K}$ *un corps quelconque (commutatif). Soit* $\mathbf{E}$ *un espace vectoriel sur* $\mathbf{K}$ *de dimension finie, et* u *un endomorphisme de* $\mathbf{E}$ *tel que* $Ker(u^2) = Ker(u)$. *Alors* $Im(u) \cap Ker(u) = 0$.

Démonstration
Soit $x \in Im(u) \cap Ker(u)$, alors :

$x \in Im(u)$ et donc $x = u(y),\ y \in \mathbf{E}$

$x \in Ker(u)$ et donc $u(x) = 0$

d'où $0 = u(x) = u^2(y)$.

En conséquence $y \in Ker(u^2) = Ker(u)$, d'où $x = 0$.

Proposition 3.5 *Soit*

$$S = \begin{pmatrix} (A^t - \lambda I)^s \\ ----- \\ \overline{P}(\lambda)^t \end{pmatrix}.$$

Alors $rang(S) = n$.

Démonstration
On a $rang(S) = rang(S^t)$, or :

$$S^t = \left(\begin{array}{c|c} (A - \lambda I)^s & \overline{P}(\lambda) \end{array} \right).$$

On a $rang((A - \lambda I)^s) = n - r$ et $rang(\overline{P}(\lambda)) = r$. De plus, les colonnes de $(A - \lambda I)^s$ engendrent $Im(a_\lambda^s)$ et les colonnes de $\overline{P}(\lambda)$ engendrent $Ker(a_\lambda^s)$, donc les colonnes de S^t engendrent $Im(a_\lambda^s) + Ker(a_\lambda^s)$.

D'autre part $Ker((a_\lambda^s)^2) = Ker(a_\lambda^s)$, d'où, en utilisant le lemme précédent, $Im(a_\lambda^s) \cap Ker(a_\lambda^s) = 0$. D'ici $Im(a_\lambda^s) + Ker(a_\lambda^s) = Im(a_\lambda^s) \oplus Ker(a_\lambda^s)$ et pour leurs dimensions, on a $(n - r) + r = n$.

Considérons la famille de systèmes d'équations linéaires :

$$SX = e_{n+j},\ j = 1,\ \ldots,\ r.$$

D'après la proposition 3.5 chaque système de cette famille possède une seule solution que l'on note w_j, $j = 1, \ldots, r$. On a :

- $w_j \in M_{n\times 1}(\mathbf{K}(\lambda))$, $j = 1, \ldots, r$
- $(w_1, \ldots, w_r)$ est une base de $\widetilde{N}(\lambda)$
- et

$$\begin{array}{ll} w_j^t v_j = 1 & j = 1,\ \ldots,\ r \\ w_j^t v_k = 0 & j \neq k \end{array}$$

Nous avons construit ainsi une base $B(\lambda) = (w_1, \ldots, w_r)$ de $\widetilde{N}(\lambda)$ telle que la matrice $\overline{Q}(\lambda)$ construite à partir de $B(\lambda)$ vérifie $\overline{Q}(\lambda)^t\overline{P}(\lambda) = I_r$. De plus, ses coefficients sont dans $\mathbf{K}(\lambda)$ et tous les calculs peuvent se faire dans $\mathbf{K}(\lambda)$.

Relation avec le calcul d'inverses à gauche pour des matrices rectangulaires

Nous réalisons ici une introduction aux matrices inverses à gauche, elle peut être trouvé dans [GLR], p. 348.

Définition Soit $R \in M_{m\times n}(\mathbf{K})$, R est *inversible à gauche* s'il existe une matrice $R^I \in M_{n\times m}(\mathbf{K})$ tel que $R^I R = I_n$. La matrice R^I s'appelle une *inverse à gauche* de R.

Théorème 3.6 *Soit $R \in M_{m\times n}(\mathbf{K})$, ces propositions sont équivalentes :*

1. *R est inversible à gauche*
2. *les colonnes de R sont indépendantes dans $\mathbf{K}^m$ (en particulier $m \geq n$)*
3. *$Ker(R) = 0$ si l'on considère R comme une application linéaire de $\mathbf{K}^n$ vers $\mathbf{K}^m$.*

Etant donné une matrice $R \in M_{m\times n}(\mathbf{K})$ inversible à gauche, une possibilité pour calculer une matrice R^I inverse à gauche pour R est la suivante :

Comme $rang(R) = n$, on peut prendre n lignes linéairement indépendantes dans R. Ces n lignes forment une matrice $R_0 \in M_{n\times n}(\mathbf{K})$ qui est inversible. Pour simplifier nous pouvons supposer que R_0 est placé dans les n premières lignes de R. Alors la matrice $R^I = (R_0^{-1}\ \ 0)$ est une inverse à gauche de R.

En général, si on a une matrice R inversible à gauche,

$$R = \begin{pmatrix} R_0 \\ -- \\ R_1 \end{pmatrix}$$

avec R_0 inversible, la matrice

$$R^I = \left(\begin{array}{c|c} R_0^{-1} - S R_1 R_0^{-1} & S \end{array} \right)$$

est une inverse à gauche pour n'importe quelle matrice $S \in M_{n\times(m-n)}(\mathbf{K})$.

Dans notre cas, pour trouver la matrice inverse de la matrice de passage $\overline{P}$, nous cherchons une matrice inverse à gauche pour chaque $\overline{P}(\lambda_k)$, $k = 1, \ldots, p$, que nous posons $\overline{Q}(\lambda_k)^t$. Chaque matrice $\overline{Q}(\lambda_k)$ est déterminée de façon unique parce qu'on impose aux lignes de $\overline{Q}(\lambda_k)^t$ de vérifier la condition d'être des vecteurs du noyau $\widetilde{N}(\lambda_k)$.

3.3 Méthode de calcul $B(\lambda)$ en utilisant les bases duales

Dans cette section nous utilisons des résultats sur les formes bilinéaires et sur l'orthogonalité qui peuvent être trouvés dans [La, LFA], par exemple. Pour une introduction, aux bases duales nous avons utilisé [FF].

Dans la section précédente, on a proposé une méthode directe pour le calcul de la base du noyau $\widetilde{N}(\lambda)$ qui nous intéresse. Nous proposons ici une méthode qui correspond à une idée un peu différente. Nous voulons calculer la base $B(\lambda)$ à partir d'une base quelconque du noyau $\widetilde{N}(\lambda)$ en faisant des transformations similaires au processus d'orthogonalisation de Gram-Schmidt.

Dans cette section, nous considérons l'espace vectoriel $\mathbf{E} = \overline{\mathbf{K}}^n$ muni du produit scalaire (i.e., de la forme bilinéaire, symétrique et non-dégénérée) qui s'exprime dans la base canonique :

$$< x,\ y >= \sum_{i=1}^{n} x_i y_i$$

Notation Si $\mathbf{F}$ est un sous-espace de $\mathbf{E}$, on note

$$\mathbf{F}^\perp = \{x \in \mathbf{E} \mid < x,\ y >= 0\ \forall y \in \mathbf{F}\}$$

Lemme 3.7 *Soit* $\mathbf{F}$ *un sous-espace de* $\mathbf{E}$*, alors* $dim(\mathbf{F}) + dim(\mathbf{F}^\perp) = n$.

Définition Soit $\mathbf{F}$ et $\mathbf{F}'$ deux sous-espaces de $\mathbf{E}$ avec $dim(\mathbf{F}) = dim(\mathbf{F}') = m \leq n$. Soit $U = (u_1, \ldots, u_m)$ une base de $\mathbf{F}$ et $V = (v_1, \ldots, v_m)$ une base de $\mathbf{F}'$. Nous dirons que V est une base de $\mathbf{F}'$ *duale de* U si

- $< u_k, v_k >= 1, \; k = 1, \ldots, m$
- $< u_k, v_j >= 0, \; j \neq k$

Théorème 3.8 *Soit* $\mathbf{F}$ *et* $\mathbf{F}'$ *deux sous-espaces de* $\mathbf{E}$ *avec* $dim(\mathbf{F}) = dim(\mathbf{F}') = m \leq n$*. Soit* U *une base de* $\mathbf{F}$*. Alors il existe* V *base de* $\mathbf{F}'$ *duale de* U *si et seulement si* $\mathbf{F} \cap (\mathbf{F}')^{\perp} = 0$.

Démonstration
Supposons V base de $\mathbf{F}'$ duale de U. Soit $u \in \mathbf{F} \cap (\mathbf{F}')^{\perp}$:

$$u \in \mathbf{F} \Rightarrow u = \alpha_1 u_1 + \ldots + \alpha_m u_m$$

$$u \in (\mathbf{F}')^{\perp} \Rightarrow u \text{ est orthogonal à } v_1, \ldots, v_m$$

Si l'on fait les produits de u avec les vecteurs $v_1, \ldots, v_m$ on obtient $\alpha_1 = \ldots = \alpha_m = 0$

Nous montrons la réciproque de façon constructive.

Supposons que $\mathbf{F} \cap (\mathbf{F}')^{\perp} = 0$. Soit $U = (u_1, \ldots, u_m)$ une base de $\mathbf{F}$ et $W = (w_1, \ldots, w_m)$ une base de $\mathbf{F}'$. Nous allons construire par récurrence une base de $\mathbf{F}'$ duale de U à partir de la base donnée W.

Soit $0 \leq i \leq m-1$, et supposons qu'on a $W = (w_1, \ldots, w_m)$ telle que pour tout $1 \leq j \leq i$, u_j vérifie :

- $< u_j, w_j >= 1$
- $< u_j, w_l >= 0, 1 \leq l \leq m, l \neq j$

Notons (H) cette hypothèse.

Nous allons voir qu'il existe $j \in \{i+1, \ldots, m\}$ tel que $< u_{i+1}, w_j > \neq 0$.

Pour cela , nous considérons le vecteur u suivant :

$$u =< u_{i+1}, w_1 > u_1 + \ldots + < u_{i+1}, w_i > u_i - u_{i+1}$$

Ce vecteur vérifie :

- $u \in \mathbf{F}$
- $u \neq 0$ (puisque sa composante sur u_{i+1} est non nulle)

Donc $u \notin (\mathbf{F}')^{\perp}$.

Par ailleurs, calculons $< u, w_l >$ pour tout $l \in \{1, \ldots, m\}$:

$$< u, w_l >=< u_{i+1}, w_1 >< u_1, w_l > + \ldots + < u_{i+1}, w_i >< u_i, w_l > - < u_{i+1}, w_l >$$

L'hypothèse (H) implique :

- Si $l \in \{1, \ldots, i\}$ alors :

$$< u, w_l >=< u_{i+1}, w_l > - < u_{i+1}, w_l >= 0$$

- Si $l \in \{i+1, \ldots, m\}$ alors :

$$< u, w_l >= - < u_{i+1}, w_l >$$

Puisque $u \notin (\mathbf{F}')^{\perp}$, nous en déduisons qu'il existe un $l \in \{i+1, \ldots, m\}$ tel que $< u_{i+1}, w_l > \neq 0$, et nous pouvons supposer que $l = i+1$ (à permutation près).

Nous construisons maintenant une base $\overline{W}$ de $\mathbf{F}'$ à partir de W comme suit :

$\overline{w}_{i+1} = \frac{1}{< u_{i+1}, w_{i+1} >} w_{i+1}$

$\overline{w}_k = \frac{- < u_{i+1}, w_k >}{< u_{i+1}, w_{i+1} >} w_{i+1} + w_k,\ k = 1, \ldots, m,\ k \neq i+1$

Alors $\overline{W}$ est telle que pour tout $1 \leq j \leq i+1$, u_j vérifie :

- $< u_j, \overline{w}_j >= 1$
- $< u_j, \overline{w}_l >= 0,\ 1 \leq l \leq m,\ l \neq j$

Raisonnons par récurrence sur i, $0 \leq i \leq m$. Si $i = 0$ l'hypothèse de récurrence est triviale, et si $0 \leq i < m$ nous venons de voir comment l'on peut passer d'une base $W = (w_1, \ldots, w_m)$ de $\mathbf{F}'$ qui vérifie l'hypothèse de récurrence au rang i à une base $\overline{W} = (\overline{w}_1, \ldots, \overline{w}_m)$ de $\mathbf{F}'$ qui vérifie l'hypothèse de récurrence au rang $i+1$. Finalement, dans le cas $i = m$ nous obtenons $V = \overline{W}$ base de $\mathbf{F}'$ duale de U.

Remarque Etant donné $\mathbf{F}$ sous-espace de $\mathbf{E}$ de dimension m, il existe plusieurs sous-espaces $\mathbf{F}'$ de $\mathbf{E}$ de dimension m vérifiant $\mathbf{F} \cap (\mathbf{F}')^{\perp} = 0$. Pour chacun de ces sous-espaces $\mathbf{F}'$, étant donné une base U de $\mathbf{F}$, il existe une unique base V de $\mathbf{F}'$ duale de U.

Remarque [FF] (p. 79).

> *Computing a basis $\{v_1, \ldots, v_n\}$ dual to a basis $\{u_1, \ldots, u_n\}$ (of $\mathbf{E}$) is essentially equivalent to inverting a matrix P consisting of the coordinates of the vectors $u_1, \ldots, u_n$ with respect to some "orthogonal basis"*

Nous utilisons le lemme suivant dans la section 4.3

Lemme 3.9 *Soit $\mathbf{F}$ et $\mathbf{F}'$ deux sous-espaces de $\mathbf{E}$ avec $dim(\mathbf{F}) = dim(\mathbf{F}') = m \leq n$ et tels que $\mathbf{F} \cap (\mathbf{F}')^{\perp} = 0$.*

Supposons $\mathbf{F} = \mathbf{F}_1 \oplus \mathbf{F}_2$, $\mathbf{F}' = \mathbf{F}'_1 \oplus \mathbf{F}'_2$, $dim(\mathbf{F}_1) = dim(\mathbf{F}'_1) = m_1$, $dim(\mathbf{F}_2) = dim(\mathbf{F}'_2) = m_2$, et tels que $\mathbf{F}_1 \subseteq (\mathbf{F}'_2)^{\perp}$. Alors :

- $\mathbf{F}_1 \cap (\mathbf{F'}_1)^\perp = 0$
- $\mathbf{F}_2 \cap (\mathbf{F'}_2)^\perp = 0$

Démonstration

- Voyons $\mathbf{F}_1 \cap (\mathbf{F'}_1)^\perp = 0$.

  On a $\mathbf{F}_1 \subseteq \mathbf{F}$ et $\mathbf{F} \cap (\mathbf{F'})^\perp = 0$, d'où $\mathbf{F}_1 \cap (\mathbf{F'})^\perp = 0$.

  Soit $x \in \mathbf{F}_1 \cap (\mathbf{F'}_1)^\perp$. Comme $\mathbf{F}_1 \subseteq (\mathbf{F'}_2)^\perp$ alors $x \in (\mathbf{F'}_2)^\perp \cap (\mathbf{F'}_1)^\perp$. Par ailleurs on a
  $$(\mathbf{F'}_2)^\perp \cap (\mathbf{F'}_1)^\perp = [\mathbf{F'}_2 \oplus \mathbf{F'}_1]^\perp = (\mathbf{F'})^\perp$$
  Donc nous avons $x \in \mathbf{F}_1 \cap (\mathbf{F'})^\perp = 0$.

- Voyons $\mathbf{F}_2 \cap (\mathbf{F'}_2)^\perp = 0$.

  Puisque $\mathbf{F} \cap (\mathbf{F'})^\perp = 0$ et $dim(\mathbf{F}) = m$, $dim((\mathbf{F'})^\perp) = n - m$ on a $\mathbf{E} = \mathbf{F} \oplus (\mathbf{F'})^\perp$. Ce que l'on peut aussi écrire $\mathbf{E} = \mathbf{F}_2 \oplus [\mathbf{F}_1 \oplus (\mathbf{F'})^\perp]$.

  D'autre part $(\mathbf{F'})^\perp \subseteq (\mathbf{F'}_2)^\perp$ (puisque $\mathbf{F'}_2 \subseteq \mathbf{F'}$) et $\mathbf{F}_1 \subseteq (\mathbf{F'}_2)^\perp$, et donc $\mathbf{F}_1 \oplus (\mathbf{F'})^\perp \subseteq (\mathbf{F'}_2)^\perp$.

  Comme $dim(\mathbf{F}_1) + dim((\mathbf{F'})^\perp) = m_1 + n - m = m_1 + n - m_1 - m_2 = n - m_2 = dim((\mathbf{F'}_2)^\perp)$ on a $\mathbf{F}_1 \oplus (\mathbf{F'})^\perp = (\mathbf{F'}_2)^\perp$.

  En conséquence $\mathbf{E} = \mathbf{F}_2 \oplus [\mathbf{F}_1 \oplus (\mathbf{F'})^\perp] = \mathbf{F}_2 \oplus (\mathbf{F'}_2)^\perp$ et donc $\mathbf{F}_2 \cap (\mathbf{F'}_2)^\perp = 0$.

Remarque Nous pouvons obtenir la démonstration du lemme précédent à partir de la démonstration constructive du théorème 3.8. Pour cela nous regardons de près le calcul des bases duales.

Soit U_1 et U_2 bases respectives de $\mathbf{F}_1$ et de $\mathbf{F}_2$, V_1 et V_2 bases de $\mathbf{F'}_1$ et de $\mathbf{F'}_2$. Donc (U_1, U_2) et (V_1, V_2) sont bases de $\mathbf{F}$ et de $\mathbf{F'}$.

Commençons le calcul de la base duale. Soit $u \in U_1$ et nous cherchons $v \in (V_1, V_2)$ tel que $< u,\ v > \neq 0$. Puisque $\mathbf{F}_1 \subseteq (\mathbf{F'}_2)^\perp$ la seule possibilité est $v \in V_1$. On en déduit que nous pouvons obtenir une base duale de U_1 à partir de V_1 et donc $\mathbf{F}_1 \cap (\mathbf{F'}_1)^\perp = 0$.

Supposons que nous avons trouvé ainsi une base duale de U_1 à partir de V_1. Maintenant la démonstration du théorème 3.8 nous dit que nous pouvons continuer le calcul de base duale avec U_2 et V_2 en oubliant la partie correspondante à $\mathbf{F}_1$ et obtenir ainsi une base duale de U_2 à partir de V_2. Donc $\mathbf{F}_2 \cap (\mathbf{F'}_2)^\perp = 0$.

Remarque Reciproquement, supposons que nous avons des relations d'orthogonalité entre $\mathbf{F}_1$ et $\mathbf{F'}_2$ ($\mathbf{F}_1 \subseteq (\mathbf{F'}_2)^\perp$), et des relations de "dualité" entre $\mathbf{F}_1$ et $\mathbf{F'}_1$ ($\mathbf{F}_1 \cap (\mathbf{F'}_1)^\perp = 0$) et entre $\mathbf{F}_2$ et $\mathbf{F'}_2$ ($\mathbf{F}_2 \cap (\mathbf{F'}_2)^\perp = 0$). Cela ne suffit pas pour pouvoir affirmer que $\mathbf{F} \cap (\mathbf{F'})^\perp = 0$ car il manque des relations d'orthogonalité entre $\mathbf{F'}_1$ et $\mathbf{F}_2$.

Nous étudions maintenant le cas particulier qui nous intéresse, avec $\mathbf{F} = N(\lambda)$ et $\mathbf{F}' = \widetilde{N}(\lambda)$. Le théorème 3.11 ci dessous prouve que $\mathbf{F} \cap (\mathbf{F}')^{\perp} = 0$ à partir du lemme suivant.

Lemme 3.10

$$[\widetilde{N}(\lambda_k)]^{\perp} = \bigoplus_{\substack{j=1,\ \dots,\ p \\ j \neq k}} N(\lambda_j)$$

Démonstration
On a :

$$\overline{\mathbf{K}}^n = N(\lambda_1) \oplus \ \dots\ \oplus N(\lambda_p)$$

et de la même façon :

$$\overline{\mathbf{K}}^n = \widetilde{N}(\lambda_1) \oplus \ \dots\ \oplus \widetilde{N}(\lambda_p)$$

Dans le théorème 3.1 nous avons vu que si $i \neq j$:

$$\left.\begin{array}{l} v \in N(\lambda_i) \\ w \in \widetilde{N}(\lambda_j) \end{array}\right\} \Rightarrow <v,\ w>= 0$$

ce qui est équivalent à :

$$\left.\begin{array}{l} v \in \oplus_{i \neq j} N(\lambda_i) \\ w \in \widetilde{N}(\lambda_j) \end{array}\right\} \Rightarrow <v,\ w>= 0$$

et il s'ensuit que :

$$\oplus_{i \neq j} N(\lambda_i) \subseteq [\widetilde{N}(\lambda_j)]^{\perp}$$

D'autre part $dim[\widetilde{N}(\lambda_j)]^{\perp} = n - r_j$ et donc :

$$\oplus_{i \neq j} N(\lambda_i) = [\widetilde{N}(\lambda_j)]^{\perp}$$

Théorème 3.11

$$N(\lambda) \cap [\widetilde{N}(\lambda)]^{\perp} = 0$$

Démonstration
Posons $\lambda = \lambda_k$, du lemme précédent on déduit :

$$\oplus_{j \neq k} N(\lambda_j) = [\widetilde{N}(\lambda_k)]^{\perp}$$

et comme

$$\overline{\mathbf{K}}^n = N(\lambda_k) \oplus (\oplus_{j \neq k} N(\lambda_j))$$

alors

$$N(\lambda) \cap [\widetilde{N}(\lambda)]^{\perp} = 0$$

Remarque Comme conséquence, si $(v_1, \ldots, v_r)$ est la base de $N(\lambda)$ formée par les colonnes de $\overline{P}(\lambda)$, nous pouvons calculer sa base duale dans $\widetilde{N}(\lambda)$.

De plus, si l'on considère une base W de $\widetilde{N}(\lambda)$ telle que les coefficients de ses vecteurs sont dans $\mathbf{K}(\lambda)$, le processus de construction de la base duale de $(v_1, \ldots, v_r)$ (avec les coefficients de v_i dans $\mathbf{K}(\lambda)$) à partir de W nous montre que les coefficients des vecteurs de la base obtenue sont dans $\mathbf{K}(\lambda)$ (puisque tous les calculs correspondants peuvent se faire dans $\mathbf{K}(\lambda)$).

Remarque Si $\overline{P}$ est une matrice de passage entre A et $\overline{A}$, pour trouver son inverse nous avons considéré pour chaque λ valeur propre de A le sous-espace $\widetilde{N}(\lambda)$ et nous avons fait un calcul de base duale.

Le théorème 3.11 nous assure que le calcul de base duale est possible dans $\widetilde{N}(\lambda)$. Est–il possible de faire ce calcul dans d'autres sous-espaces de $\overline{\mathbf{K}}^n$? Le lemme 3.10 nous montre les relations d'orthogonalité de $\widetilde{N}(\lambda)$ avec $N(\mu)$, $\mu \neq \lambda$. On en déduit que $\widetilde{N}(\lambda)$ est le seul sous-espace de $\overline{\mathbf{K}}^n$ qui vérifie les conditions nécessaires pour trouver la matrice inverse de $\overline{P}$.

4 Calcul de la matrice inverse d'une matrice de passage : cas de Jordan

Rappelons que dans la section précédente nous avons montré :

> *Si P est une matrice de passage entre A et $\overline{A}$ conforme à A, alors pour chaque λ valeur propre de A, il existe une matrice $Q(\lambda) \in M_{n\times r}(\mathbf{K}(\lambda))$ dont les colonnes forment une base (notée $B(\lambda)$) de $\widetilde{N}(\lambda)$ et telle que*
>
> $$Q(\lambda)^t P(\lambda) = I_r$$

et cela peut s'appliquer dans le cas où $\overline{A} = J$.

Dans cette section nous continuons l'étude dans ce cas, mais nous nous intéressons à l'obtention de $Q(\lambda)$ à partir d'une famille de générateurs de $\widetilde{N}(\lambda)$: puisque la matrice $P(\lambda)$ est déterminée à partir de $n_1 (\leq r)$ vecteurs, nous voulons déterminer $Q(\lambda)$ à partir du même nombre de vecteurs. De l'unicité de $Q(\lambda)$ on déduit l'unicité de cette famille de sous-espaces générateurs. De plus, nous pouvons obtenir cette famille avec les deux méthodes proposées dans la section 3.

Notation Rappelons que nous avons noté $\widetilde{N}_i = Ker(A^t - \lambda I)^i$ et $\widetilde{N}(\lambda) = \widetilde{N}_s$. Posons $\widetilde{a}_\lambda$ l'endomorphisme de $\widetilde{N}_s$ tel que $\widetilde{a}_\lambda(w) = (A^t - \lambda I)w$.

Remarque $dim(\widetilde{N}_i) = dim(N_i) = n_i$, $i = 1, \ldots, s$.

4.1 Relations entre les tours de $N(\lambda)$ et de $\widetilde{N}(\lambda)$

Soit $v^i \in N(\lambda)$ un vecteur générateur de niveau i et $w^j \in \widetilde{N}(\lambda)$ un vecteur générateur de niveau j. Nous étudions ici quelques relations existantes entre les tours engendrées par ces vecteurs.

Notation Soit $w^j \in \widetilde{N}(\lambda)$ un vecteur générateur de niveau j et $\overline{t}^j = (w^1, \ldots, w^j)$ la tour engendrée par w^j. Nous posons $\widetilde{t}^j = (w^j, \ldots, w^1)$ (et nous l'appelons aussi tour de hauteur j engendrée par w^j).

Remarque La notation précédente correspond à l'égalité des matrices suivante :

$$\widetilde{t}^j = \overline{t}^j R_j$$

avec

$$R_j = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix} \in M_j.$$

Notation Si $\overline{Q}(\lambda)$ est une matrice correspondant à une base de passage de $\widetilde{N}(\lambda)$, nous posons $\widetilde{Q}(\lambda) = \overline{Q}(\lambda) R_{J(\lambda)}$.

Dans toute cette section 4, nous utiliserons souvent la remarque et les propositions ci-dessous.

Remarque Posons

$$v = a_\lambda(x) = (A - \lambda I)x \text{ avec } x \in N(\lambda)$$

$$w = \widetilde{a}_\lambda(y) = (A^t - \lambda I)y \text{ avec } y \in \widetilde{N}(\lambda)$$

Alors :

$$w^t v = [(A^t - \lambda I)y]^t v = y^t (A - \lambda I)v = y^t a_\lambda(v)$$

$$w^t v = w^t [(A - \lambda I)x] = [(A^t - \lambda I)w]^t x = [\widetilde{a}_\lambda(w)]^t x$$

Proposition 4.1 *Soit* $\mathbf{K}$ *un corps quelconque (commutatif). Soit* $\mathbf{E}$ *un espace vectoriel sur* $\mathbf{K}$ *de dimension finie, et* u *un endomorphisme de* $\mathbf{E}$ *représenté par la matrice* A *dans la base canonique. Soit* $\widetilde{u}$ *l'endomorphisme de* $\mathbf{E}$ *représenté par* A^t *dans la base canonique. Alors* $[Im(\widetilde{u})]^\perp = Ker(u)$.

Démonstration
$[Im(\widetilde{u})]^\perp = \{x \in \mathbf{E} \mid\ x^t y = 0\ \forall y \in Im(\widetilde{u})\} = \{x \in \mathbf{E} \mid\ x^t A^t z = 0\ \forall z \in \mathbf{E}\} =$
$\{x \in \mathbf{E} \mid\ z^t A x = 0\ \forall z \in \mathbf{E}\} = \{x \in \mathbf{E} \mid\ Ax \in (\mathbf{E})^\perp\} = Ker(u).$

Corollaire 4.2 *Pour chaque i, $i = 1, \ldots, s$ on a :*

$$[Im(\tilde{a}^i_\lambda)]^\perp = N_i$$

$$[Im(a^i_\lambda)]^\perp = \widetilde{N}_i$$

Proposition 4.3 *Soit $v^i \in N(\lambda)$ un vecteur générateur de niveau i et $t^i = (v^1, \ldots, v^i)$ la tour engendrée par v^i. Soit $w^j \in \widetilde{N}(\lambda)$ un vecteur générateur de niveau j et $\tilde{t}^j = (w^j, \ldots, w^1)$ la tour engendrée par w^j. Alors :*

1. $(w^{l_2})^t v^{l_1} = 0$ *si* $l_1 + l_2 \leq max(i, j)$
2. $(w^{l_2})^t v^{l_1} = (w^j)^t v^k$ *si* $l_1 + l_2 = j + k$ *et* k *tel que* $1 \leq k \leq i$.

Démonstration
On a $(w^{l_2})^t v^{l_1} = (\tilde{a}^{j-l_2}_\lambda(w^j))^t a^{i-l_1}_\lambda(v^i)$

1. Supposons que $l_1 + l_2 \leq max(i, j)$, alors :
   - si $max(i, j) = i$ alors $i + j - (l_1 + l_2) \geq i + j - max(i, j) = j$, donc $(w^{l_2})^t v^{l_1} = (\tilde{a}^{i+j-(l_1+l_2)}_\lambda(w^j))^t v^i = 0$
   - si $max(i, j) = j$ alors $i + j - (l_1 + l_2) \geq i + j - max(i, j) = i$, donc $(w^{l_2})^t v^{l_1} = (w^j)^t a^{i+j-(l_1+l_2)}_\lambda(v^i) = 0$
2. Soit k tel que $1 \leq k \leq i$ et supposons $l_1 + l_2 = j + k$, alors :
   $(w^{l_2})^t v^{l_1} = (w^j)^t a^{i+j-(l_1+l_2)}_\lambda(v^i) = (w^j)^t a^{i-k}_\lambda(v^i) = (w^j)^t v^k$

Remarque et notation D'après le lemme précédent nous savons que $(w^{l_2})^t v^{l_1}$ ne dépend que de $l_1 + l_2$. Donc si $k = l_1 + l_2$, nous posons $\alpha_k = (w^{l_2})^t v^{l_1}$.

Conséquence Si nous considérons le lemme précédent de façon matricielle, nous avons :

$$(\tilde{t}^j)^t t^i = \begin{pmatrix} (w^j)^t \\ (w^{j-1})^t \\ \vdots \\ (w^2)^t \\ (w^1)^t \end{pmatrix} \begin{pmatrix} v^1 & v^2 & \ldots & v^{i-1} & v^i \end{pmatrix} =$$

$$\begin{pmatrix} (w^j)^t v^1 & (w^j)^t v^2 & \dots & (w^j)^t v^{i-1} & (w^j)^t v^i \\ (w^{j-1})^t v^1 & (w^{j-1})^t v^2 & \dots & (w^{j-1})^t v^{i-1} & (w^{j-1})^t v^i \\ \vdots & \vdots & & \vdots & \vdots \\ (w^2)^t v^1 & (w^2)^t v^2 & \dots & (w^2)^t v^{i-1} & (w^2)^t v^i \\ (w^1)^t v^1 & (w^1)^t v^2 & \dots & (w^1)^t v^{i-1} & (w^1)^t v^i \end{pmatrix} =$$

$$\begin{pmatrix} \alpha_{j+1} & \alpha_{j+2} & \dots & \alpha_{j+i-1} & \alpha_{j+i} \\ \alpha_j & \alpha_{j+1} & \dots & \alpha_{j+i-2} & \alpha_{j+i-1} \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_3 & \alpha_4 & \dots & \alpha_{i+1} & \alpha_{i+2} \\ \alpha_2 & \alpha_3 & \dots & \alpha_i & \alpha_{i+1} \end{pmatrix}$$

Si $i = j$ cette matrice est carrée et triangulaire supérieure :

$$(\tilde{t^i})^t t^i = \begin{pmatrix} \alpha_{i+1} & \alpha_{i+2} & \dots & \alpha_{2i-1} & \alpha_{2i} \\ 0 & \alpha_{i+1} & \dots & \alpha_{2i-2} & \alpha_{2i-1} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & & \ddots & \alpha_{i+1} & \alpha_{i+2} \\ 0 & \dots & \dots & 0 & \alpha_{i+1} \end{pmatrix}$$

Si $i > j$, alors :

$$(\tilde{t^j})^t t^i = \begin{pmatrix} 0 & \dots & 0 & \alpha_{i+1} & \alpha_{i+2} & \dots & \alpha_{j+i} \\ 0 & \dots & 0 & 0 & \alpha_{i+1} & \dots & \alpha_{j+i-1} \\ \vdots & & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & \alpha_{i+1} \end{pmatrix}$$

Si $i < j$, alors :

$$(\tilde{t^j})^t t^i = \begin{pmatrix} \alpha_{j+1} & \alpha_{j+2} & \dots & \alpha_{j+i} \\ 0 & \alpha_{j+1} & \dots & \alpha_{j+i-1} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \alpha_{j+1} \\ 0 & \dots & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & \dots & 0 \end{pmatrix}$$

Remarque Dans tous les cas, tous les coefficients d'une même diagonale parallèle à la diagonale principale sont égaux. En particulier, il suffit de déterminer la première ligne de la matrice $(\tilde{t^j})^t t^i$ pour connaître la matrice toute entière.

Remarque Soit $P(\lambda)$ une matrice correspondant à une base de passage de $N(\lambda)$ et $\widetilde{Q}(\lambda)$ une matrice correspondant à une base de passage de $\widetilde{N}(\lambda)$. Alors nous pouvons écrire leur produit en fonction des tours :

$$(\widetilde{Q}(\lambda))^t P(\lambda) = ((\widetilde{t^j})^t t^i)$$

Remarque Le lemme 4.3 nous montre que pour n'importe quelle matrice $P(\lambda)$ correspondant à une base de passage de $N(\lambda)$ et pour n'importe quelle matrice $\widetilde{Q}(\lambda)$ correspondant à une base de passage de $\widetilde{N}(\lambda)$, la matrice $(\widetilde{Q}(\lambda))^t P(\lambda)$ a une structure très particulière, avec des blocs triangulaires nuls dont les dimensions dépendent des niveaux des tours qui y interviennent.

Exemple Considérons le cas associé à une matrice de Jordan $J(\lambda)$ avec 1 bloc de dimension 3, 2 blocs de dimension 2 et 2 blocs de dimension 1.

$$(\widetilde{Q}(\lambda))^t P(\lambda) = \left(\begin{array}{ccc|cc|cc|c|c} + & + & + & + & + & + & + & + & + \\ 0 & * & * & 0 & * & 0 & * & 0 & 0 \\ 0 & 0 & * & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & + & + & + & + & + & + & + & + \\ 0 & 0 & * & 0 & * & 0 & * & 0 & 0 \\ \hline 0 & + & + & + & + & + & + & + & + \\ 0 & 0 & * & 0 & * & 0 & * & 0 & 0 \\ \hline 0 & 0 & + & 0 & + & 0 & + & + & + \\ \hline 0 & 0 & + & 0 & + & 0 & + & + & + \end{array}\right)$$

Rappelons que nous cherchons une matrice $\widetilde{Q}(\lambda)$ telle que, étant donné $P(\lambda)$ le produit $(\widetilde{Q}(\lambda))^t P(\lambda)$ soit la matrice identité. Mais nous venons de voir que si $P(\lambda)$ et $\widetilde{Q}(\lambda)$ sont construites à partir des familles de sous-espaces générateurs, il y a déjà un grand nombre d'éléments du produit nuls, et tous les autres peuvent être déterminés à partir de quelques éléments (notés par $+$).

Nous allons voir comment, en calculant une certaine famille de sous-espaces générateurs dans $\widetilde{N}(\lambda)$ nous obtenons $\widetilde{Q}(\lambda)$ tel que $(\widetilde{Q}(\lambda))^t P(\lambda)$ est l'identité.

4.2 Calcul des sous-espaces générateurs et résolution des systèmes linéaires

Rappelons que pour la méthode proposée pour le calcul de $B(\lambda)$ avec la résolution des systèmes linéaires dans 3.2, nous avons considéré la matrice

$$S = \begin{pmatrix} (A^t - \lambda I)^s \\ ----- \\ P(\lambda)^t \end{pmatrix}$$

de $n + r$ lignes et rang n, et après nous avons résolu r systèmes

$$SX = e_{n+j}, j = 1, \ldots, r$$

pour trouver la base de $\widetilde{N}(\lambda)$ cherchée.

Maintenant nous allons résoudre seulement n_1 de ces systèmes (rappelons que $n_1 = dim(N_1) \leq r$), pour obtenir une famille libre de vecteurs de $\widetilde{N}(\lambda)$. Nous montrons que ces vecteurs forment des bases de la famille de sous-espaces générateurs de $\widetilde{N}(\lambda)$ qui engendre $\widetilde{Q}(\lambda)$ telle que $\widetilde{Q}(\lambda)^t P(\lambda) = I_r$. Pour calculer cette famille de vecteurs nous considérons les systèmes correspondant à placer 1 pour les *pieds des tours* de $N(\lambda)$, c'est-à-dire, pour les vecteurs de niveau 1 de $P(\lambda)$.

Rappelons d'abord l'obtention de la matrice $P(\lambda)$ à partir d'une famille de sous-espaces générateurs : Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs de N_s. Pour chaque $i = s, \ldots, 1$ soit $(v^i_{m_{i+1}+1}, \ldots, v^i_{m_i})$ une base de G_i (avec $m_{s+1} = 0$) et pour chaque j, $j = m_{i+1}+1, \ldots, m_i$, soit t^i_j la tour de hauteur i engendrée par v^i_j. Alors

$$P(\lambda)^t = \begin{pmatrix} (t^s_1)^t \\ --- \\ \vdots \\ --- \\ (t^s_{m_s})^t \\ --- \\ \vdots \\ --- \\ (t^1_{m_2+1})^t \\ --- \\ \vdots \\ --- \\ (t^1_{m_1})^t \end{pmatrix} \quad \text{avec} \quad (t^i_j)^t = \begin{pmatrix} (v^1_j)^t \\ --- \\ \vdots \\ --- \\ (v^i_j)^t \end{pmatrix}$$

Ainsi, pour chaque tour t^i_j on considère le système :

$$SX = \begin{pmatrix} (A^t - \lambda I)^s \\ ----- \\ \vdots \\ --- \\ (v^1_j)^t \\ --- \\ \vdots \\ --- \\ (v^i_j)^t \\ --- \\ \vdots \end{pmatrix} X = \begin{pmatrix} 0 \\ --- \\ \vdots \\ -- \\ 1 \\ -- \\ \vdots \\ -- \\ 0 \\ -- \\ \vdots \end{pmatrix}$$

Notons w^i_j les solutions. Alors $(w^i_{m_{i+1}+1}, \ldots, w^i_{m_i})$ sont $g_i = m_i - m_{i+1}$ vecteurs linéairement indépendants de $\widetilde{N}_s$ et tels que pour chaque j avec $m_{i+1}+1 \leq j \leq m_i$

on a :
$$\begin{aligned} &(v_j^1)^t w_j^i = 1 \\ &v^t w_j^i = 0 \text{ pour tout vecteur } v \text{ de } P(\lambda),\ v \neq v_j^1 \end{aligned} \tag{2}$$

Théorème 4.4 *Pour chaque* i*,* $i = s, \ \ldots, \ 1$ *soit* $\widetilde{G}_i$ *le sous-espace de* $\widetilde{N}(\lambda)$ *engendré par* $(w^i_{m_{i+1}+1}, \ \ldots, \ w^i_{m_i})$*. Alors la famille* $(\widetilde{G}_s, \ \ldots, \ \widetilde{G}_1)$ *est une famille de sous-espaces générateurs de* $\widetilde{N}(\lambda)$*.*

Démonstration
Pour montrer que $(\widetilde{G}_s, \ \ldots, \ \widetilde{G}_1)$ est une famille de sous-espaces générateurs de $\widetilde{N}_s$ nous devons montrer que cette famille vérifie les conditions du théorème 2.3, à savoir :

- $g_s \geq 1$
- pour tout $i = s, \ \ldots, \ 1$
  $$\widetilde{N}_i = \widetilde{N}_{i-1} \oplus (\oplus_{j=i}^s \tilde{a}_\lambda^{j-i}(\widetilde{G}_j))$$
  $$n_i = n_{i-1} + (\textstyle\sum_{j=i}^s g_j)$$

Puisque les nombres g_i, $i = s, \ \ldots, \ 1$ vérifient déjà les conditions exigées, il nous faut montrer que pour chaque i, $i = s, \ \ldots, \ 1$ on a :

$$\widetilde{N}_i = \widetilde{N}_{i-1} \oplus \tilde{a}_\lambda^{s-i}(\widetilde{G}_s) \oplus \ \ldots \ \oplus \tilde{a}_\lambda(\widetilde{G}_{i+1}) \oplus \widetilde{G}_i$$

Si $i = s$ alors :

- $\widetilde{G}_s \subseteq \widetilde{N}_s$
- Voyons que $\widetilde{G}_s \cap \widetilde{N}_{s-1} = 0$.
  Soit $w \in \widetilde{G}_s \cap \widetilde{N}_{s-1}$. Alors nous pouvons écrire :
  $$\begin{aligned} w &= \alpha_1 w_1^s + \ldots + \alpha_{m_s} w_{m_s}^s \\ &= x_{s-1} \end{aligned}$$
  avec $x_{s-1} \in \widetilde{N}_{s-1}$. Pour chaque j avec $1 \leq j \leq m_s$ on considère le produit
  $$\begin{aligned} (v_j^1)^t w &= \alpha_j \qquad \text{(par (2))} \\ &= (v_j^1)^t x_{s-1} \end{aligned}$$
  Or, en utilisant le colloraire 4.2, $(v_j^1)^t x_{s-1} = 0$ puisque $v_j^1 = a_\lambda^{s-1}(v_j^s)$ et $x_{s-1} \in \widetilde{N}_{s-1}$. Donc $w = 0$.

Puisque $n_s = n_{s-1} + g_s$, on en déduit l'égalité cherchée pour $i = s$:

$$\widetilde{N}_s = \widetilde{N}_{s-1} \oplus \widetilde{G}_s$$

Par ailleurs on en déduit aussi que :

- $\widetilde{N}_{s-2} \oplus \tilde{a}_\lambda(\tilde{G}_s) \subseteq \widetilde{N}_{s-1}$

  Puisqu'on a $\widetilde{N}_{s-1} \cap \tilde{G}_s = 0$ et donc, en utilisant le lemme 2.2 on obtient $\tilde{a}_\lambda |_{\tilde{G}_s} : \tilde{G}_s \to \widetilde{N}_s$ injective, $\tilde{a}_\lambda(\tilde{G}_s) \subseteq \widetilde{N}_{s-1}$ et $\widetilde{N}_{s-2} \cap \tilde{a}_\lambda(\tilde{G}_s) = 0$.

- $\tilde{G}_l \subseteq \widetilde{N}_{s-1}$, $l = s-1, \ldots, 1$

  Puisque si l'on considère l tel que $s-1 \geq l \geq 1$ et $m_{l+1}+1 \leq \gamma \leq m_l$ alors on peut écrire :
  $$w^l_\gamma = x_{s-1} + \alpha_1 w^s_1 + \ldots + \alpha_{m_s} w^s_{m_s}$$
  avec $x_{s-1} \in \widetilde{N}_{s-1}$. En considérant les produits $(v^1_j)^t w^l_\gamma$ pour $1 \leq j \leq m_s$ on obtient $\alpha_1 = \ldots = \alpha_{m_s} = 0$ (par (2)) donc $w^l_\gamma \in \widetilde{N}_{s-1}$.

Hypothèse de récurrence pour i, $s \geq i \geq 1$

- $\widetilde{N}_i = \widetilde{N}_{i-1} \oplus \tilde{a}^{s-i}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_{i+1}) \oplus \tilde{G}_i$

- $\widetilde{N}_{i-2} \oplus \tilde{a}^{s-i+1}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_i) \subseteq \widetilde{N}_{i-1}$

- $\tilde{G}_l \subseteq \widetilde{N}_{i-1}$, $l = i-1, \ldots, 1$

Nous venons de voir que cette hypothèse est vérifiée pour $i = s$. Soit $s-1 \geq i \geq 1$, et supposons l'hypothèse de récurrence vérifiée pour $i+1$, c'est-à-dire :

- $\widetilde{N}_{i+1} = \widetilde{N}_i \oplus \tilde{a}^{s-i-1}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_{i+2}) \oplus \tilde{G}_{i+1}$

- $\widetilde{N}_{i-1} \oplus \tilde{a}^{s-i}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_{i+1}) \subseteq \widetilde{N}_i$

- $\tilde{G}_l \subseteq \widetilde{N}_i$, $l = i, \ldots, 1$

Alors :

- $\tilde{G}_i \subseteq \widetilde{N}_i$

- Voyons que $\tilde{G}_i \cap (\widetilde{N}_{i-1} \oplus \tilde{a}^{s-i}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_{i+1})) = 0$

  Soit $w \in \tilde{G}_i \cap (\widetilde{N}_{i-1} \oplus \tilde{a}^{s-i}_\lambda(\tilde{G}_s) \oplus \ldots \oplus \tilde{a}_\lambda(\tilde{G}_{i+1}))$. Alors nous pouvons écrire :
  $$\begin{aligned} w &= \alpha_{m_{i+1}+1} w^i_{m_{i+1}+1} + \ldots + \alpha_{m_i} w^i_{m_i} \\ &= x_{i-1} + y_s + \ldots + y_{i+1} \end{aligned}$$
  avec $x_{i-1} \in \widetilde{N}_{i-1}$ et $y_k \in \tilde{a}^{k-i}_\lambda(\tilde{G}_k)$, $s \geq k \geq i+1$. Pour chaque j avec $m_{i+1}+1 \leq j \leq m_i$ on considère le produit
  $$\begin{aligned} (v^1_j)^t w &= \alpha_j & \text{(par (2))} \\ &= (v^1_j)^t x_{i-1} + (v^1_j)^t y_s + \ldots + (v^1_j)^t y_{i+1} \end{aligned}$$

  On a :

- $(v_j^1)^t x_{i-1} = 0$ par le corollaire 4.2, puisque $v_j^1 = a_\lambda^{i-1}(v_j^i)$ et $x_{i-1} \in \widetilde{N}_{i-1}$
- $(v_j^1)^t y_k = 0$ par le corollaire 4.2, puisque $y_k = \tilde{a}_\lambda^{k-i}(z_k) \in Im(\tilde{a}_\lambda)$ (car $k - i \geq 1$) et $v_j^1 \in N_1$.

Donc $w = 0$.

Puisque $n_i = n_{i-1} + g_s + \ldots + g_{i+1} + g_i$ on déduit :

$$\widetilde{N}_i = \widetilde{N}_{i-1} \oplus \tilde{a}_\lambda^{s-i}(\widetilde{G}_s) \oplus \ \ldots \ \oplus \tilde{a}_\lambda(\widetilde{G}_{i+1}) \oplus \widetilde{G}_i$$

Et aussi :

- $\widetilde{N}_{i-2} \oplus \tilde{a}_\lambda^{s-i+1}(\widetilde{G}_s) \oplus \ \ldots \ \oplus \tilde{a}_\lambda(\widetilde{G}_i) \subseteq \widetilde{N}_{i-1}$

  Puisque si l'on pose $H = \tilde{a}_\lambda^{s-i}(\widetilde{G}_s) \oplus \ \ldots \ \oplus \tilde{a}_\lambda(\widetilde{G}_{i+1}) \oplus \widetilde{G}_i$ alors on a $\widetilde{N}_{i-1} \cap H = 0$ et donc, en utilisant le lemme 2.2 on obtient $\tilde{a}_\lambda \mid_H : H \to \widetilde{N}_s$ injective, $\tilde{a}_\lambda(H) \subseteq \widetilde{N}_{i-1}$ et $\widetilde{N}_{i-2} \cap \tilde{a}_\lambda(H) = 0$.

- $\widetilde{G}_l \subseteq \widetilde{N}_{i-1}$, $l = i-1, \ \ldots, \ 1$

  Puisque si l'on considère l tel que $i - 1 \geq l \geq 1$ et γ tel que $m_{l+1} + 1 \leq \gamma \leq m_l$ alors on peut écrire (parce que $\widetilde{G}_l \subseteq \widetilde{N}_i$) :

  $$w_\gamma^l = x_{i-1} + y_s + \ldots + y_{i+1} + \alpha_{m_{i+1}+1} w_{m_{i+1}+1}^i + \ldots + \alpha_{m_i} w_{m_i}^i$$

  avec $x_{i-1} \in \widetilde{N}_{i-1}$ et $y_k \in \tilde{a}_\lambda^{k-i}(\widetilde{G}_k)$, $s \geq k \geq i+1$.

  En considérant les produits $(v_j^1)^t w_\gamma^l$ pour $m_{i+1} + 1 \leq j \leq m_i$ on obtient $\alpha_{m_{i+1}+1} = \ldots = \alpha_{m_i} = 0$ par (2).

  Par ailleurs, soit k tel que $s \geq k \geq i+1$, et posons :

  $$y_k = \tilde{a}_\lambda^{k-i}(\alpha_{m_{k+1}+1}^k w_{m_{k+1}+1}^k + \ldots + \alpha_{m_k}^k w_{m_k}^k)$$

  Soit j tel que $m_{k+1} + 1 \leq j \leq m_k$ et considérons les produits $(v_j^{k-i+1})^t w_\gamma^l = (a_\lambda^{i-1}(v_j^k))^t w_\gamma^l$, on obtient $\alpha_{m_{k+1}+1} = \ldots = \alpha_{m_k} = 0$ en utilisant (2).

  Donc $w_\gamma^l \in \widetilde{N}_{i-1}$.

Rappelons que nous avons fixé une famille de sous-espaces générateurs $(G_s, \ \ldots, \ G_1)$ dans $N(\lambda)$, avec $(v_{m_{i+1}+1}^i, \ \ldots, \ v_{m_i}^i)$ base de G_i. Cela nous a fourni la matrice $P(\lambda)$. Ensuite nous avons calculé la famille de sous-espaces générateurs $(\widetilde{G}_s, \ \ldots, \ \widetilde{G}_1)$ de $\widetilde{N}(\lambda)$, de bases $(w_{m_{i+1}+1}^i, \ \ldots, \ w_{m_i}^i)$; et nous avons obtenu ces vecteurs comme les solutions (uniques) des systèmes considérés.

Lemme 4.5 *Soit v_j^i un vecteur dans la base de G_i et t_j^i la tour qu'il engendre, soit w_l^k un vecteur dans la base de $\widetilde{G}_k$ et $\tilde{t}_l^k$ la tour qu'il engendre, alors :*

- $(\widetilde{t}_l^k)^t t_j^i = 0$ *si* $j \neq l$
- *si* $j = l$ *on a* $k = i$ *et* $(\widetilde{t}_l^i)^t t_j^i = I_i$

Démonstration
En utilisant le lemme 4.3 il nous faut calculer $(w_l^k)^t v_j^\gamma$ pour $\gamma = 1, \ldots, i$. On a $(w_l^k)^t v_j^\gamma = 1$ si et seulement si $j = l$ et $\gamma + k = i + 1$ d'où on déduit ce lemme de façon immédiate.

Théorème 4.6 *Soit* $\widetilde{Q}(\lambda)$ *la matrice correspondant à la famille de sous-espaces générateurs* $(\widetilde{G}_s, \ldots, \widetilde{G}_1)$*. Alors*

$$(\widetilde{Q}(\lambda))^t P(\lambda) = I_r$$

Nous avons réduit ainsi, dans le cas général le nombre de systèmes linéaires à résoudre de $r = n_s = dim(N_s)$ à $n_1 = dim(N_1)$. C'est d'autant plus intéressant que la matrice est "moins" diagonale.

4.3 Calcul des sous-espaces générateurs et bases duales

Il s'agit maintenant de trouver la famille de sous-espaces générateurs qui engendre la matrice $\widetilde{Q}(\lambda)$ telle que $(\widetilde{Q}(\lambda))^t P(\lambda) = I_r$ en utilisant le calcul des bases duales.

Remarque Rappelons que :

- Si $(G_s, \ldots, G_1)$ est une famille de sous-espaces générateurs de $N(\lambda)$ alors

$$N_1 = a_\lambda^{s-1}(G_s) \oplus \ldots \oplus a_\lambda^2(G_3) \oplus a_\lambda(G_2) \oplus G_1$$

- D'autre part on a montré dans le théorème 3.11

$$N(\lambda) \cap [\widetilde{N}(\lambda)]^\perp = 0$$

Remarque Notons qu'en utilisant le théorème 2.3 nous pouvons écrire :

$$\widetilde{N}_s = \widetilde{a}_\lambda(H) \oplus [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]$$

pour un certain sous-espace H de $\widetilde{N}_s$. De plus :

$$N_1 \subseteq [\widetilde{a}_\lambda(H)]^\perp$$

Théorème 4.7 *Soit* $(G_s, \ldots, G_1)$ *une famille de sous-espaces générateurs de* $N(\lambda)$ *et* $(\widetilde{G}_s, \ldots, \widetilde{G}_1)$ *une famille de sous-espaces générateurs de* $\widetilde{N}(\lambda)$*. Alors :*

$$N_1 \cap [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$$

Démonstration
Puisque $N_s \cap [\widetilde{N}_s]^\perp = 0$ et $N_1 \subseteq N_s$ alors $N_1 \cap [\widetilde{N}_s]^\perp = 0$.

Soit $x \in N_1 \cap [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]^\perp$. Alors :

- $x \in N_1 \subset [\tilde{a}_\lambda(H)]^\perp$
- $x \in [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]^\perp$
- $[\tilde{a}_\lambda(H)]^\perp \cap [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]^\perp = [\tilde{a}_\lambda(H) \oplus (\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1)]^\perp = [\widetilde{N}_s]^\perp$

Donc $x = 0$.

Lemme 4.8 *Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs de $N(\lambda)$ et $(\widetilde{G}_s, \ldots, \widetilde{G}_1)$ une famille de sous-espaces générateurs de $\widetilde{N}(\lambda)$.*

Pour tout i, $s \geq i \geq 1$ on a :

$$a_\lambda^{i-1}(G_i) \subseteq [\widetilde{N}_{i-1}]^\perp \subseteq [\widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1]^\perp$$

Démonstration
Soit i, $s \geq i \geq 1$, alors :

- $a_\lambda^{i-1}(G_i) \subseteq [\widetilde{N}_{i-1}]^\perp$
  puisque si $x = a_\lambda^{i-1}(z) \in a_\lambda^{i-1}(G_i)$ et $y \in \widetilde{N}_{i-1}$ alors $< x, \ y >=< z, \ \tilde{a}_\lambda^{i-1}(y) >= 0$
- $[\widetilde{N}_{i-1}]^\perp \subseteq [\widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1]^\perp$
  puisque $\widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1 \subseteq \widetilde{N}_{i-1}$

Nous utilisons le lemme 3.7 dans la démonstration du théorème suivant.

Théorème 4.9 *Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs de $N(\lambda)$ et $(\widetilde{G}_s, \ldots, \widetilde{G}_1)$ une famille de sous-espaces générateurs de $\widetilde{N}(\lambda)$.*

Pour tout i, $s \geq i \geq 1$ on a :

$$a_\lambda^{i-1}(G_i) \cap [\widetilde{G}_i]^\perp = 0$$

Démonstration
On a :

$$N_1 = a_\lambda^{s-1}(G_s) \oplus \ \ldots \ \oplus a_\lambda(G_2) \oplus G_1$$

Le théorème 4.7 nous montre que :

$$N_1 \cap [\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$$

Soit $i = s$, posons :

$$\begin{array}{ll}
\mathbf{F}_1 = a_\lambda^{s-1}(G_s) & \mathbf{F'}_1 = \widetilde{G}_s \\
\mathbf{F}_2 = a_\lambda^{s-2}(G_{s-1}) \oplus \ \ldots \ \oplus G_1 & \mathbf{F'}_2 = \widetilde{G}_{s-1} \oplus \ldots \oplus \widetilde{G}_1 \\
\mathbf{F} = N_1 = \mathbf{F}_1 \oplus \mathbf{F}_2 & \mathbf{F'} = \widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1 = \mathbf{F'}_1 \oplus \mathbf{F'}_2
\end{array}$$

Donc en utilisant le lemme 4.8 nous sommes dans les conditions du lemme 3.7 et nous pouvons en déduire que :

- $a_\lambda^{s-1}(G_s) \cap [\widetilde{G}_s]^\perp = 0$
- $[a_\lambda^{s-2}(G_{s-1}) \oplus \ \ldots \ \oplus G_1] \cap [\widetilde{G}_{s-1} \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$

Hypothèse de récurrence pour i, $s \geq i \geq 1$

- $a_\lambda^{i-1}(G_i) \cap [\widetilde{G}_i]^\perp = 0$
- $[a_\lambda^{i-2}(G_{i-1}) \oplus \ \ldots \ \oplus G_1] \cap [\widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$

Nous avons vu que cette hypothèse est vérifiée pour $i = s$. Soit i tel que $s-1 \geq i \geq 1$ et supposons l'hypothèse de récurrence pour $i+1$, c'est-à-dire :

- $a_\lambda^{i}(G_{i+1}) \cap [\widetilde{G}_{i+1}]^\perp = 0$
- $[a_\lambda^{i-1}(G_i) \oplus \ \ldots \ \oplus G_1] \cap [\widetilde{G}_i \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$

Alors nous pouvons poser :

$$\begin{array}{ll}
\mathbf{F}_1 = a_\lambda^{i-1}(G_i) & \mathbf{F'}_1 = \widetilde{G}_i \\
\mathbf{F}_2 = a_\lambda^{i-2}(G_{i-1}) \oplus \ \ldots \ \oplus G_1 & \mathbf{F'}_2 = \widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1 \\
\mathbf{F} = \mathbf{F}_1 \oplus \mathbf{F}_2 & \mathbf{F'} = \mathbf{F'}_1 \oplus \mathbf{F'}_2
\end{array}$$

En utilisant les lemmes 4.8, 3.7 nous en déduisons :

- $a_\lambda^{i-1}(G_i) \cap [\widetilde{G}_i]^\perp = 0$
- $[a_\lambda^{i-2}(G_{i-1}) \oplus \ \ldots \ \oplus G_1] \cap [\widetilde{G}_{i-1} \oplus \ldots \oplus \widetilde{G}_1]^\perp = 0$

Remarque et exemple Le théorème précédent nous montre que étant donné des familles de sous-espaces générateurs $(G_s, \ \ldots, \ G_1)$ de N_s et $(\widetilde{G}_s, \ \ldots, \ \widetilde{G}_1)$ de $\widetilde{N}_s$, si V_i est une base fixée de $a_\lambda^{i-1}(G_i)$ nous pouvons trouver W_i base duale de V_i à partir d'une base de $\widetilde{G}_i$.

Reprenons l'exemple associé à une matrice de Jordan $J(\lambda)$ avec 1 bloc de dimension 3, 2 blocs de dimension 2 et 2 blocs de dimension 1.

Supposons $P(\lambda)$ et $\widetilde{Q}(\lambda)$ correspondant à des bases de passages dans une telle situation, c'est-à-dire, construites à partir des bases des sous-espaces générateurs telles que la base de $\widetilde{G}_i$ est duale de la base de $a_\lambda^{i-1}(G_i)$. Alors :

$$(\widetilde{Q}(\lambda))^t P(\lambda) = \left(\begin{array}{ccc|cc|cc|c|c}
1 & + & + & + & + & + & + & + & + \\
0 & 1 & * & 0 & * & 0 & * & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & + & + & 1 & + & 0 & + & + & + \\
0 & 0 & * & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & + & + & 0 & + & 1 & + & + & + \\
0 & 0 & * & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & + & 0 & + & 0 & + & 1 & 0 \\
\hline
0 & 0 & + & 0 & + & 0 & + & 0 & 1
\end{array}\right)$$

C'est-à-dire, nous avons ajouté l'information sur les diagonales des sous-blocs carrés de cette matrice.

Remarque Rappelons qu'un sous-espace générateur $\widetilde{G}_i$ est un supplémentaire *quelconque* d'un certain sous-espace F (qui dépend de $\widetilde{G}_s, \ldots, \widetilde{G}_{i+1}$) dans $\widetilde{N}_i$.

Considérons V une base de N_1 et $\widetilde{V}$ une base de $\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1$. Le théorème 4.7 nous dit que l'on peut construire une base W duale de V à partir de $\widetilde{V}$. Alors W est une base du sous-espace $\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1$, mais le processus de construction de la base duale *peut changer* la famille de sous-espaces générateurs qu'on a considéré au départ (sans changer, bien sûr, le sous-espace $\widetilde{G}_s \oplus \ldots \oplus \widetilde{G}_1$).

Pour illustrer cette affirmation nous pouvons supposer $s = 2$. Considérons V_2 base de $a_\lambda(G_2)$, V_1 base de G_1, $\widetilde{V}_2$ base de $\widetilde{G}_2$ duale de V_2 et $\widetilde{V}_1$ base de $\widetilde{G}_1$ duale de V_1. Donc $V = (V_2,\ V_1)$ est une base de N_1 et $\widetilde{V} = (\widetilde{V}_2,\ \widetilde{V}_1)$ est une base de $\widetilde{G}_2 \oplus \widetilde{G}_1$. Regardons de près le calcul de la base duale de V à partir de $\widetilde{V}$ (voir remarques au lemme 3.7). Il ne manque plus qu'obtenir des relations d'orthogonalité entre $\widetilde{V}_2$ et V_1. Pour cela il faut faire des transformations du type :

$$\overline{v}_2 = \alpha \widetilde{v}_1 + \widetilde{v}_2$$

avec $\widetilde{v}_1 \in \widetilde{V}_1$ et $\widetilde{v}_2 \in \widetilde{V}_2$. Mais ces transformations changent le supplémentaire de N_1 dans N_2, et donc nous avons fait le passage de la famille $(\widetilde{G}_2,\ \widetilde{G}_1)$ à une famille de sous-espaces générateurs $(\widetilde{G}_2^0,\ \widetilde{G}_1)$ telle que

$$\widetilde{G}_2^0 \oplus \widetilde{G}_1 = \widetilde{G}_2 \oplus \widetilde{G}_1 \quad \text{et} \quad \widetilde{G}_2^0 \subseteq [G_1]^\perp$$

Plus généralement, étant donné $(G_s,\ \ldots,\ G_1)$ famille de sous-espaces de N_s, il existe des familles de sous-espaces générateurs $(\widetilde{G}_s^0,\ \ldots,\ \widetilde{G}_1^0)$ dans $\widetilde{N}_s$ telles que pour chaque i, $s \geq i \geq 2$ on a :

$$\widetilde{G}_i^0 \subseteq [a_\lambda^{i-2}(G_{i-1}) \oplus \ldots \oplus G_1]^\perp$$

Lemme 4.10 *Soit $(G_s,\ \ldots,\ G_1)$ une famille de sous-espaces générateurs de $N(\lambda)$ et soit $(\widetilde{G}_s^0,\ \ldots,\ \widetilde{G}_1^0)$ une famille de sous-espaces générateurs de $\widetilde{N}(\lambda)$ telle que pour tout i, $s \geq i \geq 2$ on a :*

$$\widetilde{G}_i^0 \subseteq [a_\lambda^{i-2}(G_{i-1}) \oplus \ldots \oplus G_1]^\perp$$

Soit V_i une base de $a_\lambda^{i-1}(G_i)$ et W_k une base de $\widetilde{G}_k^0$ duale de V_k avec $s \geq i, k \geq 1$. Soit $v_j^i \in V_i$ et t_j^i la tour qu'il engendre, soit $w_l^k \in W_k$ $\widetilde{t}_l^k$ la tour qu'il engendre, alors :

- $(\widetilde{t}_l^k)^t t_j^i = 0$ *si* $j \neq l$
- *si* $j = l$ *on a* $k = i$ *et* $(\widetilde{t}_l^i)^t t_j^i = I_i$

Démonstration et remarque La démonstration du lemme précédent est la même que la démonstration du lemme 4.5 puisque nous sommes arrivés ainsi à une famille de sous-espaces générateurs de $\widetilde{N}(\lambda)$ vérifiant les mêmes conditions.

De même, de ce lemme on déduit le théorème 4.6, c'est-à-dire que nous avons trouvé les vecteurs nécessaires pour construire la matrice $\widetilde{Q}(\lambda)$ "inverse" de $P(\lambda)$:

$$(\widetilde{Q}(\lambda))^t P(\lambda) = I_r$$

Exemple Reprenons l'exemple associé à une matrice de Jordan $J(\lambda)$ avec 1 bloc de dimension 3, 2 blocs de dimension 2 et 2 blocs de dimension 1.

Supposons $P(\lambda)$ et $\widetilde{Q}(\lambda)$ correspondant à des bases de passages obtenues des familles de sous-espaces générateurs dans les conditions du lemme 4.10. Alors :

$$(\widetilde{Q}(\lambda))^t P(\lambda) = \left(\begin{array}{ccc|cc|cc|c|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)$$

Théorème 4.11 *Soit $(G_s, \ldots, G_1)$ une famille de sous-espaces générateurs de $N(\lambda)$. Il existe une seule famille $(\widetilde{G}_s^0, \ldots, \widetilde{G}_1^0)$ de sous-espaces générateurs de $\widetilde{N}(\lambda)$ telle que pour tout i, $s \geq i \geq 2$ on a :*

$$\widetilde{G}_i^0 \subseteq [a_\lambda^{i-2}(G_{i-1}) \oplus \ldots \oplus G_1]^\perp$$

Démonstration

Soit $(\widetilde{G}_s^0, \ldots, \widetilde{G}_1^0)$ et $(\widetilde{G}_s^1, \ldots, \widetilde{G}_1^1)$ deux familles de sous-espaces générateurs dans les conditions de cet énoncé.

Pour chaque i, $s \geq i \geq 1$ soit V_i une base de G_i, $\widetilde{V}_i^0$ la base de $\widetilde{G}_i^0$ duale de la base de $a_\lambda^{i-1}(G_i)$ et $\widetilde{V}_i^1$ la base de $\widetilde{G}_i^1$ duale de la base de $a_\lambda^{i-1}(G_i)$. Soit V la base de N_s engendrée par $(V_s, \ldots, V_1)$ et $\widetilde{V}^0$, $\widetilde{V}^1$ les bases de $\widetilde{N}_s$ engendrées respectivement par $(\widetilde{V}_s^0, \ldots, \widetilde{V}_1^0)$ et par $(\widetilde{V}_s^1, \ldots, \widetilde{V}_1^1)$.

Par le théorème 4.6 nous avons que $\widetilde{V}^0$ et $\widetilde{V}^1$ sont deux bases de $\widetilde{N}_s$ duales a V, et donc elles coincident, d'où $\widetilde{V}_i^0 = \widetilde{V}_i^1$ pour $i = s, \ldots, 1$. On conclut que $\widetilde{G}_i^0 = \widetilde{G}_i^1$ pour $i = s, \ldots, 1$.

Remarque A partir des contenus de cette section nous pouvons déduire beaucoup de relations d'orthogonalité entre des sous-espaces de $N(\lambda)$ et sous-espaces de $\widetilde{N}(\lambda)$. Ces relations peuvent être très intéressantes pour les calculs que nous effectuons, en particulier pour les faire beaucoup plus effectifs. Mais ces relations demandent des énoncés précis dont nous ne disposons pas encore.

5 Algorithmes et Exemples

Nous montrons ici les algorithmes du programme que nous avons implanté en Axiom [JS]. Ces algorithmes correspondent aux méthodes expliquées dans la section 2 pour le calcul de la forme canonique de Jordan et d'une matrice de passage, et aux méthodes expliquées dans la section 3 pour le calcul de la matrice inverse de la matrice de passage. Nous avons implanté aussi les algorithmes correspondant aux méthodes de la section 4 : nous ne les montrons pas ici car ils sont un peu compliqués. Par contre les exemples donnés à la fin de cette section utilisent ces algorithmes. Ce programme utilise la clôture algébrique dynamique [DD].

5.1 Algorithmes

Voici les principaux algorithmes pour le calcul de la forme canonique de Jordan, d'une matrice de passage et de son inverse.

Pour l'instant, nous supposons qu'étant donné A une matrice carrée de dimension n avec coefficients dans un corps $\mathbf{K}$, nous avons déjà calculé λ une valeur propre de A et r sa multiplicité caractéristique.

En fait, ces informations sont obtenues avec la clôture algébrique dynamique, et nous étudions ce point dans le paragraphe suivant. Mais nous pouvons supposer aussi que nous avons obtenu ces informations par d'autres méthodes.

Voici l'algorithme principal :

Algorithme 1 *Jordan : jordan*(A, n, λ, r)
Entrée : $A \in M_n$, λ *une valeur propre de* A *de multiplicité caractéristique* r
Sortie : $[P(\lambda), J(\lambda), Q(\lambda)]$ *tel que* $Q(\lambda)P(\lambda) = I_r$ *et* $Q(\lambda)AP(\lambda) = J(\lambda)$
début

mcl:= $(A - \lambda I)$
$kmcl$:= $noyau(mcl)$
`if` $\#kmcl = r$ `then`
  $casDiagonal(\lambda, r, mcl, kmcl)$
`else`
  $casGeneral(\lambda, r, mcl, kmcl)$

fin

La tâche de cet algorithme est de détecter si nous sommes dans un cas diagonalisable ou non. Evidemment le cas diagonalisable est beaucoup plus simple que le cas général.

Cet algorithme utilise le sous-algorithme *noyau* pour le calcul d'une base du noyau d'une matrice. Il apparaît dans la liste suivante des algorithmes.

Algorithmes de base

Nous appelons ces algorithmes “de base” puisqu’ils correspondent en général, à des fonctions simples que nous n’avons pas voulu reproduire ici, ou bien a des fonctions que nous n’avons pas programmé, et pour lesquelles nous utilisons les fonctions d’Axiom (comme par exemple *noyau*).

Algorithme 2 *Noyau : noyau*(m)
Entrée : m *une matrice à coefficients dans un corps*
Sortie : $[v_1, \ldots, v_l]$ *une base du noyau de* m

Algorithme 3 *Construction d’une matrice : matrix*(lv)
Entrée : $lv = [v_1, \ldots, v_l]$ *une liste de vecteurs*
Sortie : mv *la matrice dont les colonnes sont les vecteurs de* lv

Algorithme 4 *Construction d’une matrice diagonale : diagonal*(λ, r)
Entrée : λ *élément d’un corps,* r *un entier tel que* $r \geq 1$
Sortie : md *la matrice de dimension* r *dont les éléments diagonaux sont égaux à* λ *et les autres à 0.*

Algorithme 5 *Dimensions des sous-espaces générateurs : dimenGene*$(ldim)$
Entrée : $ldim = [n_s, \ldots, n_1]$ *la liste des dimensions des noyaux des puissances de* $(A - \lambda I)$
Sortie : $[g_s, \ldots, g_1]$ *la liste des dimensions des sous-espaces générateurs (voir 2.2.1)*

Algorithme 6 *Matrice générale de Jordan : geneJblock*(λ, dg)
Entrée : λ *élément d’un corps,* $dg = [g_s, \ldots, g_1]$ *la liste des dimensions des sous-espaces générateurs*
Sortie : $J(\lambda)$ *la matrice générale de Jordan pour* λ

Algorithme 7 *Compléter une base : completer*(be, bf)
Entrée : be *une base d’un espace* E *de dimension* k_1*,* bf *une base d’un sous-espace* F *de* E *de dimension* k_2 *avec* $k_2 \leq k_1$
Sortie : *une liste de* $k_1 - k_2$ *vecteurs telle que si on l’ajoute à la base de* F*, on obtient une base de* E*.*

Algorithme 8 *Matrice du système : matrixSys*$(m1, m2)$
Entrée : $m1$ *matrice* $n \times n$ *et* $m2$ *matrice* $r \times n$
Sortie : m *matrice* $(n + r) \times n$ *avec* $m1$ *dans les* n *premières lignes et* $m2$ *dans les* r *dernières lignes*

Algorithme 9 *Résolution des systèmes linéaires : solveSys*(m, lv)
Entrée : m *matrice des coefficients,* $lv = [v_1, \ldots, v_r]$ *la liste des vecteurs pour la seconde partie de l’équation*
Sortie : $ls = [vs_1, \ldots, vs_r]$ *la liste des vecteurs solutions tels que* $m \cdot vs_i = v_i$

Algorithme 10 *Transposer une matrice : transpose(m)*
Entrée : *m une matrice de coefficients dans un corps*
Sortie : *la matrice m^t transposée de m*

Finalement l'algorithme suivant n'entre pas dans le cadre décrit pour les algorithmes précédents. C'est une fonction que nous avons implanté pour le calcul des bases de passage, mais nous considérons cette fonction trop technique pour la reproduire ici.

Algorithme 11 *Reordonner pour la base de passage : reordonner(llv)*
Entrée : *llv est une liste de listes de vecteurs*
Sortie : *lv est la liste qui contient une base de passage dans un ordre précis*

Algorithmes pour Jordan

Voici le cas diagonal :

Algorithme 12 *Cas Diagonal : casDiagonal$(\lambda, r, mcl, kmcl)$*
Entrée : *λ une valeur propre de A de multiplicité caractéristique r, la matrice caractéristique mcl et une base kmcl du noyau de mcl*
Sortie : $[P(\lambda), J(\lambda), Q(\lambda)]$ *tel que* $Q(\lambda)P(\lambda) = I_r$ *et* $Q(\lambda)AP(\lambda) = J(\lambda)$
début

$P(\lambda)$:= $matrix(kmcl)$
$J(\lambda)$:= $diagonal(\lambda, r)$
bs:= $baseSp(mcl, kmcl)$
$Q(\lambda)$:= $matrix(bs)$
`return` $[P(\lambda),\ J(\lambda),\ Q(\lambda)]$

fin

Et le cas général :

Algorithme 13 *Cas Général : casGeneral$(\lambda, r, mcl, kmcl)$*
Entrée : *λ une valeur propre de A de multiplicité caractéristique r, la matrice caractéristique mcl et kmcl une base du noyau de mcl*
Sortie : $[P(\lambda), J(\lambda), Q(\lambda)]$ *tel que* $Q(\lambda)P(\lambda) = I_r$ *et* $Q(\lambda)AP(\lambda) = J(\lambda)$
début

[s,lpmcl,lkmcl,ldim]:=*minimalM(λ,r,mcl,kmcl)*
pas:= $passage(lkmcl)$
$P(\lambda)$:= $matrix(pas)$
dg:= $dimenGene(ldim)$
$J(\lambda)$:= $geneJblock(\lambda, dg)$
bs:= $baseSp(mcl, kmcl)$
$Q(\lambda)$:= $matrix(bs)$
`return` $[P(\lambda),\ J(\lambda),\ Q(\lambda)]$

fin

Dans les deux algorithmes précédents on fait appel à la fonction $baseSp$ pour trouver la matrice $Q(\lambda)$ (qui est "essentiellement" l'inverse de $P(\lambda)$). Cette fonction est implantée de deux façon, qui correspondent aux deux méthodes exposés dans la section 3 pour le calcul de la matrice inverse de la matrice de passage.

Mais avant de considérer ce point, voyons comment l'on calcule la multiplicité minimale de la valeur propre dans le cas général, ainsi que la matrice de passage.

Pour calculer la multiplicité de la valeur propre nous calculons les puissances sucessives de la matrice $(A-\lambda I)$ jusqu'à obtenir la puissance s dont le noyau $Ker(A-\lambda I)^s$ est de dimension r.

Algorithme 14 *Multiplicité minimale : $minimalM(\lambda, r, mcl, kmcl)$*
Entrée : *λ une valeur propre de A de multiplicité caractéristique r, la matrice caractéristique mcl et $kmcl$ une base du noyau de mcl*
Sortie : *$[s, lpmcl, lkmcl, ldim]$ où :*

- *s est la multiplicité minimale de λ,*
- *$lpmcl = [mcl^s, \ldots, mcl]$ est la liste des puissances de la matrice mcl*
- *$lkmcl = [kmcls, \ldots, kmcl1]$ est la liste des bases des noyaux pour les matrices de $lpmcl$*
- *$ldim = [n_s, \ldots, n_1]$ est la liste des dimensions des noyaux de $lkmcl$*

début

```
s := 1
pm := mcl
dim := #kmcl
lpmcl := [mcl]
lkmcl := [kmcl]
ldim := [dim]
for i in 1...r repeat
    s := s+1
    pm := pm × mcl
    kpm := noyau(pm)
    dim := #kpm
    lpmcl := cons(pm,lpmcl)
    lkmcl := cons(kpm,lkmcl)
    ldim := cons(dim,ldim)
    if dim = r then Stop
return [s, lpmcl, lkmcl, ldim]
```

fin

Le lecteur peut penser que nous calculons trop de choses dans l'algorithme $minimalM$, mais toutes les données calculées par cette fonction sont utilisées ailleurs, par exemple dans le calcul de la matrice de passage.

L'algorithme ci-dessous nous montre le calcul de la matrice de passage à partir des sous-espaces générateurs, comme expliqué dansla section 2.2. Dans cette fonction gi correspond au sous-espace générateur de niveau i et mi correspond au sous-espace M_i de niveau i.

Algorithme 15 *Cas Général, base de passage : passage*$(lkmcl, mcl)$
Entrée : $lkmcl = [kmcls, \ldots, kmcl1]$ *est la liste des bases des noyaux et* $mcl = (A - \lambda I)$
Sortie : *une base de passage pas*
début

```
vpas:= []
for i in 1...#lkmcl repeat
    mi:= []
    if i = 1 then
        gi:= completer(lkmcl.1,lkmcl.2)
    else
        mi:= [mcl · v for v in vpas.last]
        if #mi ≠ #lkmcl.i then
            if i ≠ #lkmcl then
                laux:= concat(lkmcl.(i+1),mi)
                gi:= completer(lkmcl.i,laux)
            else
                gi:= completer(lkmcl.i,mi)
            mi:= concat(mi,gi)
    vpas:= concat(vpas,mi)
pas:= reordonner(vpas)
return pas
```

fin

Passons maintenant à la fonction $baseSp$ pour le calcul de la matrice $Q(\lambda)$. Nous voulons calculer une base spéciale de $Ker(A^t - \lambda I)^s$ (c'est-à-dire une base telle que la matrice $Q(\lambda)$ vérifie $Q(\lambda)^t P(\lambda) = I_r$) et nous avons vu dans la section 3 que cela peut être donné par deux méthodes. Voyons la résolution des systèmes linéaires :

Algorithme 16 *Calcul d'une base spéciale de* $Ker(A^t - \lambda I)^s$ *en utilisant la résolution des systèmes : baseSp*$(mcl, kmcl)$
Entrée : *la matrice caractéristique mcl et une base kmcl de son noyau*
Sortie : *une base spéciale de* $Ker(A^t - \lambda I)^s$
début

```
tmcl:= transpose(mcl)
maux:= matrix(kmcl)
maS:= matrixSys(tmcl,maux)
r:= #kmcl
for i in 1...r repeat
    v_i := e_{n+i}
lv:= [v_1, ..., v_r]
ls:= solveSys(maS,lv)
return ls
```

fin

L'algorithme suivant utilise le calcul des bases duales :

Algorithme 17 *Calcul d'une base spéciale de $Ker(A^t - \lambda I)^s$ en utilisant les bases duales : $baseSp(mcl, kmcl)$*
Entrée : *la matrice caractéristique mcl et une base kmcl de son noyau*
Sortie : *une base spéciale de $Ker(A^t - \lambda I)^s$*
début

```
tmcl:= transpose(mcl)
ktmcl:= noyau(tmcl)
nk:= dualBase(kmcl,tkmcl)
return nk
```

fin

Voici l'algorithme de calcul des bases duales (où $p \cdot q$ réprésente le produit scalaire de p par q). Il correspond à la démonstration constructive du théorème 3.8.

Algorithme 18 *Bases duales : $dualBase(bf, bfp)$*
Entrée : *bf une base d'un sous-espace F et bfp une base d'un sous-espace F' tels que $F \cap (F')^\perp = 0$ et $dim(F) = dim(F')$*
Sortie : *une base de F' duale de bf*
début

```
db:=[]
for p in bf repeat
    bfpaux:= []
    for q in bfp repeat
        pro:= p · q
        bfp:= rest(bfp)
        if pro = 0 then
            bfpaux:= cons(q,bfpaux)
        else
```

```
                *db*:= *cons(q,db)*
                *bfp*:= *concat(bfpaux,bfp)*
                *Stop*
        bp.1:= (1/pro)bp.1
        for i in 2...#db repeat
            db.i:= -(db.i · p) bp.1 + db.i
        for i in 1...#bfp repeat
            bfp.i:= -(dfp.i · p) bp.1 + dfp.i
    return db
```

fin

5.2 Jordan et la clôture algébrique dynamique

L'utilisation de la clôture algébrique dynamique [DD] est extrêmement simple. Nous avons construit une petite fonction pour le calcul d'une valeur propre, elle utilise la fonction *characteristicPolynomial* qui calcule le polynôme charactéristique d'une matrice. Voici donc, la fonction *eigenvalue* :

Algorithme 19 *Calcul d'une valeur propre : eigenvalue*(A)
Entrée : A *matrice carrée de dimension* n
Sortie : λ *une valeur propre de* A
début

```
    pc:= characteristicPolynomial(A)
    ev:= rootOf(pc, 'ev)
    return ev
```

fin

Cette fonction correspond à l'introduction de la clôture algébrique dans notre programme, avec la fonction `rootOf` (où le deuxième argument –`'ev`– est un symbole pour l'écriture externe de la racine). La clôture algébrique dynamique dispose aussi d'une fonction `multiplicity(pc,ev)` que nous utilisons pour calculer la multiplicité caractéristique de la valeur propre.

De cette façon nous avons implanté notre programme pour Jordan. Il n'y a plus qu'à faire appel a notre fonction *jordan* avec les données et `allCases`.

`allCases` est la fonction du processus de l'évaluation dynamique qui est chargée de gérer la distinction des racines avec des comportements differentes (voir [DD] ou [Go] pour avoir plus d'information sur `allCases`).

5.3 Exemples

Nous présentons ici quelques exemples sur les nombres rationnels. Pour chaque exemple nous utilisons deux fonctions selon l'algorithme choisi pour le calcul de la matrice inverse de la matrice de passage :

- *jordanUBB* c'est-à-dire, en utilisant l'algorithme des bases duales
- *jordanSLS* c'est-à-dire, en utilisant l'algorithme de résolution des systèmes linéaires.

Les résultats sont les mêmes pour les deux méthodes (puisque seul change le calcul de la matrice inverse), ils sont présentés "en petit".

Exemple 1 Dans cet exemple nous considérons une matrice générale de Jordan.

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

```
allCases(jordanUBB$NDJORDAN(K),aa)$DCTRL(CA,M,Sol)

[value is
             +1  0  0  0  0+             +2  1  0  0  0+
             |             |             |             |
             |0  1  0  0  0|             |0  2  1  0  0|
             |             |             |             |
[transition= |0  0  1  0  0|, jordanBlock= |0  0  2  0  0|,
             |             |             |             |
             |0  0  0  1  0|             |0  0  0  2  1|
             |             |             |             |
             +0  0  0  0  1+             +0  0  0  0  2+
          +1  0  0  0  0+
          |             |
          |0  1  0  0  0|
          |             |
 inverse= |0  0  1  0  0|]
          |             |
          |0  0  0  1  0|
          |             |
          +0  0  0  0  1+
in case ev = 2]

Time: 5.27 sec

allCases(jordanSLS$NDJORDAN(K),aa)$DCTRL(CA,M,Sol)

[value is
             +1  0  0  0  0+             +2  1  0  0  0+
             |             |             |             |
             |0  1  0  0  0|             |0  2  1  0  0|
             |             |             |             |
[transition= |0  0  1  0  0|, jordanBlock= |0  0  2  0  0|,
             |             |             |             |
             |0  0  0  1  0|             |0  0  0  2  1|
             |             |             |             |
             +0  0  0  0  1+             +0  0  0  0  2+
```

```
         +1  0  0  0  0+
         |            |
         |0  1  0  0  0|
         |            |
 inverse= |0  0  1  0  0|]
         |            |
         |0  0  0  1  0|
         |            |
         +0  0  0  0  1+
in case ev = 2]

Time: 1.77 sec
```

C'est-à-dire, la forme canonique de A est A et la matrice de passage obtenue est la matrice identité.

Exemple 2 Cet exemple est dû à [Ne], p.107. Nous trouvons cet exemple très intéressant, et nous croyons qu'il est très bien choisi, malgré l'opinion de l'auteur :

> *Unfortunately, it is a little difficult to construct an interesting example of low order.*

Cet exemple nous a permis de tester nos programmes et de détecter quelques erreurs.

$$A = \begin{pmatrix} 1 & 0 & -1 & 1 & 0 \\ -4 & 1 & -3 & 2 & 1 \\ -2 & -1 & 0 & 1 & 1 \\ -3 & -1 & -3 & 4 & 1 \\ -8 & -2 & -7 & 5 & 4 \end{pmatrix}$$

```
allCases(jordanUBB$NDJORDAN(K),bb)$DCTRL(CA,M,Sol)

[value is
            +0  1  0   0   0+            +2  1  0  0  0+
            |               |            |             |
            |0  2  0  - 1  1|            |0  2  1  0  0|
            |               |            |             |
[transition= |1  1  0  - 1  0|, jordanBlock= |0  0  2  0  0|,
            |               |            |             |
            |1  2  1  - 1  0|            |0  0  0  2  1|
            |               |            |             |
            +1  5  0  - 2  0+            +0  0  0  0  2+
         + 3   0   2   0  - 1+
         |                   |
         | 1   0   0   0   0 |
         |                   |
 inverse= |- 1  0  - 1  1   0 |]
         |                   |
         | 4   0   1   0  - 1|
         |                   |
         + 2   1   1   0  - 1+
in case ev = 2]

Time: 1.46 sec

allCases(jordanSLS$NDJORDAN(K),bb)$DCTRL(CA,M,Sol)

Time: 2.55 sec
```

C'est-à-dire $P^{-1} \cdot A \cdot P = J$, avec :

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 & 1 \\ 1 & 1 & 0 & -1 & 0 \\ 1 & 2 & 1 & -1 & 0 \\ 1 & 5 & 0 & -2 & 0 \end{pmatrix} \quad J = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \quad P^{-1} = \begin{pmatrix} 3 & 0 & 2 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 \\ 4 & 0 & 1 & 0 & -1 \\ 2 & 1 & 1 & 0 & -1 \end{pmatrix}$$

Exemple 3 Voyons maintenant un exemple avec deux valeurs propres distinctes.

$$A = \begin{pmatrix} 1 & 1 & -1 & 2 & -1 \\ 2 & 0 & 1 & -4 & -1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 0 & 1 \\ 0 & 0 & -3 & 3 & -1 \end{pmatrix}$$

```
allCases(jordanUBB$NDJORDAN(K),dd)$DCTRL(CA,M,Sol)

[value is
            + 2   0  1+
            |        |
            |- 2  2  0|              +1  1  0+
            |        |               |       |
[transition= | 2   0  0|, jordanBlock= |0  1  1|,
            |        |               |       |
            | 2   0  0|              +0  0  1+
            |        |
            + 0   0  0+
         +      1         +
         |0  0  -    0    0 |
         |      2          |
         |                 |
 inverse= |   1       1    1 |]
         |0  -  0    -    - |
         |   2       2    2 |
         |                 |
         +1  0  0  - 1  - 1+
in case ev = 1,

value is
            + 3    1 +
            |        |
            |- 3  - 1|
            |        |              +- 1   1 +
[transition= | 0    0 |, jordanBlock= |        |,
            |        |              + 0   - 1+
            | 0    1 |
            |        |
            + 3    0 +
         +              1+
         |0  0   0   0  -|
 inverse= |              3|]
         |               |
         +0  0  - 1  1  0+
in case ev = - 1]

Time: 2.21 sec

allCases(jordanSLS$NDJORDAN(K),dd)$DCTRL(CA,M,Sol)
```

```
Time: 3.44 sec
```

C'est-à-dire $P^{-1} \cdot A \cdot P = J$, avec :

$$P = \left(\begin{array}{rrr|rr} 2 & 0 & 1 & 3 & 1 \\ -2 & 2 & 0 & -3 & -1 \\ 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \end{array}\right) \qquad J = \left(\begin{array}{rrr|rr} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{array}\right)$$

$$P^{-1} = \left(\begin{array}{rrrrr} 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 & -1 & -1 \\ \hline 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & -1 & 1 & 0 \end{array}\right)$$

Exemple 4 Voici un exemple d'une matrice diagonalisable :

$$A = \begin{pmatrix} 3 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

```
allCases(jordanUBB$NDJORDAN(K),ee)$DCTRL(CA,M,Sol)

[value is
             +  2          +
             |ev  - 3ev + 1|
[transition= |             |, jordanBlock= [ev],
             |  - ev + 1   |
             |             |
             +      1      +
          +1   2   2      1  1   2   5      2    1      2+
 inverse= |- ev  - - ev + -  - ev  - - ev + -  - - ev + -|]
          +6      3      3  6       6      3    6      3+
          3     2
in case ev  - 6ev  + 9ev - 2 = 0]

Time: 1.59 sec

allCases(jordanSLS$NDJORDAN(K),ee)$DCTRL(CA,M,Sol)

Time: 1.85 sec
```

C'est-à-dire $P^{-1} \cdot A \cdot P = J$, avec :

$$P = \left(\begin{array}{c|c|c} \lambda_1^2 - 3\lambda_1 + 1 & \lambda_2^2 - 3\lambda_2 + 1 & \lambda_3^2 - 3\lambda_3 + 1 \\ -\lambda_1 + 1 & -\lambda_2 + 1 & -\lambda_3 + 1 \\ 1 & 1 & 1 \end{array}\right)$$

$$J = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

$$P^{-1} = \begin{pmatrix} \frac{1}{6}\lambda_1^2 - \frac{2}{3}\lambda_1 + \frac{1}{3} & \frac{1}{6}\lambda_1^2 - \frac{5}{6}\lambda_1 + \frac{2}{3} & -\frac{1}{6}\lambda_1 + \frac{2}{3} \\ \hline \frac{1}{6}\lambda_2^2 - \frac{2}{3}\lambda_2 + \frac{1}{3} & \frac{1}{6}\lambda_2^2 - \frac{5}{6}\lambda_2 + \frac{2}{3} & -\frac{1}{6}\lambda_2 + \frac{2}{3} \\ \hline \frac{1}{6}\lambda_3^2 - \frac{2}{3}\lambda_3 + \frac{1}{3} & \frac{1}{6}\lambda_3^2 - \frac{5}{6}\lambda_3 + \frac{2}{3} & -\frac{1}{6}\lambda_3 + \frac{2}{3} \end{pmatrix}$$

où λ_1, λ_2 et λ_3 sont les trois racines distinctes de :

$$p(x) = x^3 - 6x^2 + 9x - 2.$$

Exemple 5 Et finalement une matrice non diagonalisable avec des nombres algébriques comme valeurs propres.

$$A = \begin{pmatrix} 0 & 8 & 0 & 0 & 9 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & -5 \\ 0 & -4 & 1 & 0 & -4 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

```
allCases(jordanUBB$NDJORDAN(K),cc)$DCTRL(CA,M,Sol)

[value is
             +            5        +
             |- 3ev + 3   - ev - 5 |
             |            2        |
             |                     |
             |3                    |
             |- ev - 3    - ev + 1 |
             |2                    |
             |                     |               +ev 1 +
[transition= |             3       |, jordanBlock= |     |,
             | ev - 1    - - ev + 3|               +0  ev+
             |             2       |
             |                     |
             |3                    |
             |- ev - 1       1     |
             |2                    |
             |                     |
             +- ev + 2       0     +
          +33      47   9      25   31      45  9      25   19      27+
          |-- ev + --   - ev + --   -- ev + --  - ev + --   -- ev + --|
          |8       8    2      4    8       8   2      4    4       4 |
 inverse= |                                                           |]
          |  11                 11    9      3    3      9    9       |
          |- -- ev - 2  - 2ev - --  - - ev - -  - - ev - -  - - ev - 3|
          +  8                  4     8      2    2      4    4       +
          2
in case ev  - 2 = 0,
```

```
value is
             +  9+
             |- -|
             |  7|
             |   |
             |  9|
             |- -|
             |  7|
             |   |
[transition= | 1 |, jordanBlock= [1], inverse= [- 7  - 7  - 7  - 7  - 7]]
             | - |
             | 7 |
             |   |
             | 9 |
             | - |
             | 7 |
             |   |
             + 1 +
in case ev = 1]

Time: 9.26 sec

allCases(jordanSLS$NDJORDAN(K),cc)$DCTRL(CA,M,Sol)

Time: 11.32 sec
```

C'est-à-dire $P^{-1} \cdot A \cdot P = J$, avec :

$$P = \left(\begin{array}{cc|cc|c} -3\lambda_1+3 & \frac{5}{2}\lambda_1-5 & -3\lambda_2+3 & \frac{5}{2}\lambda_2-5 & -\frac{9}{7} \\ \frac{3}{2}\lambda_1-3 & -\lambda_1+1 & \frac{3}{2}\lambda_2-3 & -\lambda_2+1 & -\frac{9}{7} \\ \lambda_1-1 & -\frac{3}{2}\lambda_1+3 & \lambda_2-1 & -\frac{3}{2}\lambda_2+3 & \frac{1}{7} \\ \frac{3}{2}\lambda_1-1 & 1 & \frac{3}{2}\lambda_2-1 & 1 & \frac{9}{7} \\ -\lambda_1+2 & 0 & -\lambda_2+2 & 0 & 1 \end{array}\right)$$

$$J = \left(\begin{array}{cc|cc|c} \lambda_1 & 1 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 \\ \hline 0 & 0 & \lambda_2 & 1 & 0 \\ 0 & 0 & 0 & \lambda_2 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

$$P^{-1} = \begin{pmatrix} \frac{33}{8}\lambda_1 + \frac{47}{8} & \frac{9}{2}\lambda_1 + \frac{25}{4} & \frac{31}{8}\lambda_1 + \frac{45}{8} & \frac{9}{2}\lambda_1 + \frac{25}{4} & \frac{19}{4}\lambda_1 + \frac{27}{4} \\ -\frac{11}{8}\lambda_1 - 2 & -2\lambda_1 - \frac{11}{4} & -\frac{9}{8}\lambda_1 - \frac{3}{2} & -\frac{3}{2}\lambda_1 - \frac{9}{4} & -\frac{9}{4}\lambda_1 - 3 \\ \hline \frac{33}{8}\lambda_2 + \frac{47}{8} & \frac{9}{2}\lambda_2 + \frac{25}{4} & \frac{31}{8}\lambda_2 + \frac{45}{8} & \frac{9}{2}\lambda_2 + \frac{25}{4} & \frac{19}{4}\lambda_2 + \frac{27}{4} \\ -\frac{11}{8}\lambda_2 - 2 & -2\lambda_2 - \frac{11}{4} & -\frac{9}{8}\lambda_2 - \frac{3}{2} & -\frac{3}{2}\lambda_2 - \frac{9}{4} & -\frac{9}{4}\lambda_2 - 3 \\ \hline -7 & -7 & -7 & -7 & -7 \end{pmatrix}$$

où λ_1 et λ_2 sont les deux racines distinctes de :

$$p(x) = x^2 - 2.$$

Conclusion

Nous avons montré :

- Soit A une matrice carrée et P une matrice de passage dont les coefficients de chaque colonne s'expriment en fonction d'une seule valeur propre de A. Alors les éléments de chaque ligne de P^{-1} s'expriment aussi en fonction d'une seule valeur propre.

  De plus nous décrivons deux nouveaux algorithmes permettant de calculer P^{-1} en manipulant une seule valeur propre à tout moment du calcul.

  Pour cela nous avons étudié la structure de l'espace vectoriel, décomposé comme somme directe des sous-espaces caractéristiques correspondant aux valeurs propres de A.

- En particulier, ce résultat s'applique aux matrices de passage entre A et sa forme canonique de Jordan.

  De plus dans ce cas nous décrivons une version plus efficace de chacun de nos deux algorithmes.

  Pour cela nous avons étudié la structure de chaque sous-espace caractéristique comme somme directe de certains sous-espaces en utilisant des familles des sous-espaces générateurs.

Finalement, nous avons implanté le calcul de la forme canonique de Jordan, d'une matrice de passage conforme et de son inverse en utilisant la clôture algébrique dynamique, et nous avons traité plusieurs exemples avec ce programme.

L'utilisation de la clôture algébrique dynamique nous a permis d'oublier les problèmes de calculs avec des nombres algébriques qui se posent en général dans ce type de questions.

Références

[CHO] L. Chambadal, J. L. Ovaert – *Cours de Mathématiques. Algèbre II.* Gauthier-Villars Editeur (1972).

[DD] C. Dicrescenzo, D. Duval. – *Algebraic Extensions and Algebraic Closure in Scratchpad.* Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 358, ed. P. Gianni, p. 440–446 (1989).

[FF] D. K. Faddeev, V. N. Faddeeva – *Computational Methods of Linear Algebra.* W. H. Freeman and Company (1963).

[Gil] I. Gil – *Contribution à l'Algèbre Linéaire Formelle. Formes normales de matrices et applications.* Thèse de l'Institut National Polytechnique de Grenoble (1993).

[GLR] I. Gohberg, P. Lancaster, L. Rodman – *Matrix Polynomials.* Academic Press (1982).

[Go] T. Gómez-Díaz – *II. Clôture constructible dynamique.* Cette thèse (1994).

[JS] Jenks R. D., Sutor R. D. – *AXIOM, The Scientific Computation System.* Springer-Verlag (1992).

[La] S. Lang – *Algebra.* Addison-Wesley Publishing Company (1965).

[Laz] D. Lazard – *Maniement des Nombres Algébriques : Le calcul de l'exponentielle d'une matrice est–il exponentiel ?.* Séminaire d'informatique théorique, L.I.T.P. Univ. Paris VI (1985).

[LFA] J. Lelong-Ferrand, J. M. Arnaudiès – *Cours de Mathématiques - Tome I, Algèbre (2e éd.).* Dunond (1974).

[MO] K. Martin, J. M. Olazabal – *An algorithm to compute the change basis for the rational form of K-endomorphisms.* Extracta Mathematicae, 6, n. 2-3, p. 142-144 (1992).

[ML] T. M. L. Mulders, A. H. M. Levelt – *A Maple package for the Computation of Normal Forms of Matrices.* Maple Share Library (1993).

[Ne] E. D. Nering – *Linear Algebra and Matrix Theory.* John Wiley and Sons, Inc. (1963).

[Oz] P. Ozello – *Calcul exact des Formes de Jordan et de Frobenius d'une matrice.* Thèse de l'Université Scientifique et Médicale de Grenoble (1987).

Partie II
Clôture constructible dynamique

Introduction

Dans ce document, nous décrivons l'implantation de la "*Clôture Constructible Dynamique*" que nous avons réalisée dans le système de calcul formel Axiom [JS]. Dans l'article [Du90] on décrit les premières intentions sur ce programme, ainsi que des exemples (calculés "à la main").

Ce programme est une application de la technique appelée "évaluation dynamique" introduite pour faire des calculs avec des nombres algébriques [DDD]. Cette technique peut être décrite de façon simple comme l'automatisation de la "discussion en fonction de valeurs de paramètres" au sens du lycée. Pour avoir une justification rigoureuse de l'évaluation dynamique, on peut s'appuyer sur la théorie des esquisses [DR]. Elle a déjà été utilisée pour écrire d'autres programmes comme *la clôture algébrique d'un corps* [DDD, DD] ou *le corps premier de caractéristique arbitraire* [Du89].

Par exemple, si on demande à notre logiciel le calcul du rang de la matrice :

$$\begin{pmatrix} 1 & 1 \\ 1 & a \end{pmatrix}$$

en fonction des valeurs du paramètre complexe a, le système répond de façon automatique :

```
[ value is 1 in case a = 1,
  value is 2 in case a /= 1 ]
```

(où `/=` signifie $\neq$) en utilisant un algorithme de calcul du rang qui a été écrit pour les matrices à coefficients sur un corps quelconque, sans aucune notion de "paramètre".

On voit déjà, même sur ce petit exemple, l'importance des tests d'égalité.

C'est la première fois qu'une telle méthode de calcul est implantée de manière tout-à-fait systématique.

Le plan de ce travail est le suivant :

- Dans la première section, nous rappelons la définition de corps et nous introduisons les concepts de paramètre et de clôture constructible.

- Dans la deuxième section, nous introduisons la clôture constructible dynamique et ses principaux opérateurs : `newElement`, `areEqual` et `areDifferent`.

- Dans la troisième section, nous étudions de façon approfondie les éléments essentiels pour réaliser des calculs dans la clôture constructible dynamique, spécialement les scindages.
- Dans la quatrième section, nous décrivons le programme réalisé et ses différentes composantes.
- La cinquième section est consacrée aux exemples et applications.

1 La clôture constructible d'un corps

1.1 Corps

Rappelons qu'on peut définir un *corps* (commutatif) comme un ensemble $\mathbf{K}$ muni de deux constantes 0 et 1, de deux lois binaires $+$ et $\times$, d'une loi unaire $-$, et d'une loi unaire inv définie sur $\mathbf{K} - \{0\}$, soumises aux formules qui traduisent les propriétés suivantes :

- la loi $+$ est associative, commutative, d'élément neutre 0, de symétrique $-$,
- la loi $\times$ est associative, commutative, d'élément neutre 1,
- la loi $\times$ est distributive par rapport à $+$,

et enfin aux deux formules :

- $1 \neq 0$,
- pour tout $x \neq 0$, $x \times inv(x) = 1$.

Notons l'apparition dans ces deux dernières formules du prédicat d'égalité à 0. Notons aussi que la soustraction (on pose bien sûr $x - y = x + (-y)$) permet de ramener tout test d'égalité dans $\mathbf{K}$ à un test d'égalité à 0 (en effet $x = y$ équivaut à $(x - y) = 0$).

Parmi les corps figurent entre autres le corps des réels, noté $\mathbf{R}$, le corps des complexes $\mathbf{C}$, le corps des rationnels $\mathbf{Q}$, ainsi que le corps fini $\mathbf{F}_p$ à p éléments pour tout nombre premier p (formé des entiers modulo p). Les corps $\mathbf{R}$ et $\mathbf{Q}$ sont ordonnés, c'est-à-dire que l'on peut y définir une relation d'ordre "compatible" avec la structure de corps, mais ce n'est pas le cas des corps $\mathbf{C}$ ni $\mathbf{F}_p$. De toute manière nous ne nous intéressons pas ici aux questions d'ordre, et plus précisément, nous nous limitons aux calculs qui ne font intervenir que les quatre opérations de corps et les tests d'égalité. Pour la gestion des questions d'ordre dans des corps ordonnés, l'implantation de la "*clôture algébrique réelle d'un corps ordonné*" [DGV] est en cours et l'implantation de la "*clôture constructible réelle d'un corps ordonné*" est envisagée.

Calculs dans un corps

Considérons un calcul sur un corps $\mathbf{K}$ consistant en un nombre fini d'opérations $+$, $-$, $\times$ et *division par un élément non nul*, à partir des constantes 0 et 1, et des tests d'égalité.

S'il n'y a que des opérations $+$, $-$, $\times$, le calcul peut se réaliser entièrement dans la structure d'anneau sous-jacent. Notons $\mathbf{A}$ l'anneau sous-jacent au corps $\mathbf{K}$. S'il y a des divisions ou des tests d'égalité, nous effectuons un nombre fini de tests d'égalité à zéro. Nous construisons ainsi, au fur et à mesure du calcul, un ensemble multiplicatif S engendré par les éléments dont on sait qu'ils sont non nuls. Le calcul est effectué, entièrement, dans l'anneau des fractions de $\mathbf{A}$ associé à S, appelé aussi *l'anneau de fractions à dénominateurs dans S* [Bo].

Paramètres dans un corps

Nous considérons comme *paramètre* une constante dans un corps de valeur non spécifiée.

Lorsqu'un paramètre a intervient dans un calcul, toutes les valeurs possibles de cette constante doivent être considérées. Nous notons $V(a)$ l'ensemble des valeurs possibles du paramètre a. Nous verrons que l'ensemble $V(a)$ peut changer au cours du calcul.

1.2 Clôture constructible

Dans tout ce qui suit, nous fixons un corps $\mathbf{K}$, que nous appelons le *corps de base*, et qui sera souvent dans les applications un "petit" corps, typiquement $\mathbf{K} = \mathbf{Q}$ ou bien $\mathbf{K} = \mathbf{F}_p$. On pourrait même prendre comme corps de base le "corps premier de caractéristique arbitraire" de [Du89].

La clôture constructible de $\mathbf{K}$ est un corps $\hat{\mathbf{K}}$ qui contient toute extension de type fini de $\mathbf{K}$ (c'est-à-dire toute extension de $\mathbf{K}$ admettant un nombre fini de générateurs en tant que $\mathbf{K}$-algèbre) et qui est minimal pour cette propriété.

C'est donc un "énorme" corps, qui contient entre autres la clôture algébrique de $\mathbf{K}$ et qui a un degré de transcendance infini sur $\mathbf{K}$. Lorsque $\mathbf{K} = \mathbf{Q}$, on peut remplacer $\hat{\mathbf{K}}$ par $\mathbf{C}$ grâce au "principe de Lefschetz".

Nous avons choisi le mot "constructible" à cause de la relation avec les *ensembles constructibles* de la géométrie algébrique :

Un *ensemble constructible* V de $\mathbf{C}^n$ peut être défini comme

$$V = \{(a_1, \ldots, a_n) \in \mathbf{C}^n \mid P_1(a_1, \ldots, a_n) \ \varsigma_1 \ 0 \text{ et } \ldots \text{ et } P_k(a_1, \ldots, a_n) \ \varsigma_k \ 0\}$$

avec $P_i \in \mathbf{C}[x_1, \ldots, x_n]$ et $\varsigma_i \in \{=, \neq\}$.

L'analogie avec la clôture constructible deviendra claire au début de la section 3.

Calculs dans la clôture constructible

De façon similaire aux calculs effectués dans des corps, un calcul dans la clôture constructible est entièrement fait dans une extension $\mathbf{L}$ de type fini sur $\mathbf{K}$. De même, cette extension dépend du calcul à effectuer et elle peut *changer* pendant le calcul puisque, en fait, elle est construite au fur et à mesure du calcul. C'est pourquoi nous nous plaçons a priori dans "le plus gros" corps possible.

2 Clôture constructible dynamique

De façon plus "constructive" et plus précise, $\hat{\mathbf{K}}$ est un corps contenant $\mathbf{K}$ (noté `CL` ci-dessous) :

```
CL:= DynamicConstructibleClosure(K)
```

et muni d'un "générateur de constantes" :

```
newElement: Symbol -> CL
```

Plus précisément, toute affectation de la forme :

```
a:= newElement(A)
```

donne accès, en le nommant, à un nouvel élément a de $\hat{\mathbf{K}}$. Cet élément doit être considéré comme un nouveau "paramètre" intervenant dans le calcul. Par la suite, pour simplifier les notations, nous ne distinguerons pas l'élément de $\hat{\mathbf{K}}$ du symbole qui le représente, ce qui correspond à une instruction du type :

```
a:= newElement('a)
```

A cette nouvelle constante a est associé un ensemble de valeurs possibles dans $\hat{\mathbf{K}}$ (noté $V(a)$). Cet ensemble est décrit par des *contraintes* sur le paramètre, qui seront vues à la prochaine section.

Pour imposer ou interdire des valeurs pour ces paramètres, nous avons doté $\hat{\mathbf{K}}$ de deux opérateurs :

```
areEqual: (CL,CL) -> Boolean
areDifferent: (CL,CL) -> Boolean
```

Ces opérateurs sont chargés d'*imposer* des contraintes sur les paramètres. Ils seront étudiés de façon plus précise à la prochaine section. La valeur booléenne de ces opérateurs indique si la nouvelle contrainte est compatible avec les anciennes.

Le rôle de ces trois fonctions est très important : c'est avec ce générateur de constantes et ces deux opérateurs qu'on va construire, au fur et au mesure d'un calcul donné, le sous-ensemble $\mathbf{L}$ de $\hat{\mathbf{K}}$ nécessaire pour obtenir la solution (ou plutôt les diverses solutions dans les différents cas) de ce calcul. En plus, la description de $\mathbf{L}$ à chaque moment du calcul va être donnée par les appels faits à `newElement` et par les contraintes que doivent vérifier les paramètres ainsi ajoutés.

La construction de $\mathbf{L}$ est récursive : si aucun appel de `newElement` n'a été fait, alors $\mathbf{L} = \mathbf{K}$. Sinon, à un point donné du calcul, soit n le nombre d'appels de `newElement` avant ce point, et soient $a_1, a_2, \ldots, a_n$ les paramètres qui ont été successivement introduits. A ce point, les éléments de $\hat{\mathbf{K}}$ que l'on sait nommer sont construits à partir des éléments de $\mathbf{K}$ et des a_i en utilisant les opérations de corps, donc $\mathbf{L} = \mathbf{K}(a_1,\ a_2,\ \ldots,\ a_n)$.

En travaillant récursivement sur n, on peut supposer qu'on sait calculer sur le sous-corps $\mathbf{K}_{n-1} = \mathbf{K}(a_1,\ a_2,\ \ldots,\ a_{n-1})$, et que l'on veut à partir de cela trouver une méthode de calcul sur $\mathbf{L} = \mathbf{K}_{n-1}(a_n)$. Le problème est ainsi ramené au cas où $n = 1$, ce que nous supposons désormais. Notons a l'unique élément de $\hat{\mathbf{K}}$ introduit par `newElement`, et $\mathbf{L}$ le corps $\mathbf{K}(a)$.

3 Calculs dans la clôture constructible dynamique. Scindages

Nous allons voir maintenant comment on peut faire des calculs dans $\mathbf{L} = \mathbf{K}(a)$, en supposant que l'on a déjà tout ce qui est nécessaire aux calculs sur $\mathbf{K}$.

3.1 Contraintes

Théorème 3.1 *Soit a un paramètre dans $\hat{\mathbf{K}}$, alors a vérifie, en tout moment du calcul, une contrainte d'un de ces types :*

- `anyElement` *: il n'y a pas de contrainte sur a.*
- `exception` *: une contrainte du type $P_1(a) \neq 0$ et … et $P_k(a) \neq 0$ avec $P_1,\ \ldots,\ P_k$ polynômes en une variable sur $\mathbf{K}$ unitaires de degré > 0 et sans facteur commun.*
- `algebraic` *: une contrainte du type $P(a) = 0$ avec P polynôme en une variable sur $\mathbf{K}$ unitaire de degré > 0.*

De plus, lorsque la caractéristique de $\mathbf{K}$ est nulle, les polynômes $P,\ P_1,\ \ldots,\ P_k$ ci-dessus peuvent être choisis sans facteur multiple.

Démonstration
Nous réalisons la preuve de ce théorème dans les prochains paragraphes.

Chacune de ces contraintes associe à a un ensemble de valeurs possibles $V(a)$ dans $\hat{\mathbf{K}}$:

- `anyElement` : $V(a) = \hat{\mathbf{K}}$.
- `exception` : a peut prendre comme valeur un élément quelconque de $\hat{\mathbf{K}}$ autre qu'une des racines du polynôme P_i pour chaque $i = 1, \ldots, k$.
- `algebraic` : a peut prendre comme valeur une racine quelconque de P.

Remarque Il est clair que les paramètres vérifiant une contrainte de type `algebraic` ont pour valeur des éléments algébriques sur le corps de base. Les valeurs des paramètres qui vérifient une contrainte de type `exception` peuvent être ou bien algébriques ou bien *transcendantes* sur $\mathbf{K}$.

3.2 Représentation et opérations de base

Nous montrons ici comment l'on peut représenter les éléments de $\mathbf{L}$ à partir d'objets connus, et que sur cette représentation on peut implanter les quatre opérations $+$, $-$, $\times$ et inv, en conservant toujours l'un des trois types de contraintes du théorème 3.1 sur a. Nous étudions le cas particulier des tests d'égalité dans la section suivante.

La représentation des éléments de $\mathbf{L} = \mathbf{K}(a)$ peut se faire par des fractions rationnelles en une variable sur $\mathbf{K}$, c'est-à-dire par une partie du corps $\mathbf{K}(t)$, où t est une "indéterminée" :

- Les éléments de $\mathbf{L}$ qui sont dans $\mathbf{K}$ sont leurs propres représentants, c'est-à-dire les fractions constantes de $\mathbf{K}(t)$.
- L'élément a est représenté par l'indéterminée t.
- Si b et c sont deux éléments de $\mathbf{L}$ représentés respectivement par les fractions B et C alors, $b + c$ est représenté par $B + C$, $-b$ par $-B$, $b \times c$ par $B \times C$, et si $b \neq 0$ alors $inv(b)$ est représenté par $1/B$.

Ceci nous permet de retrouver aisément un élément b de $\mathbf{L}$ à partir d'un de ses représentants $B \in \mathbf{K}(t)$ puisqu'alors $B = B_1/B_2$ avec $B_2(a) \neq 0$ et $b = B(a) = B_1(a)/B_2(a)$.

Il est à noter que cette représentation n'est certainement pas *canonique*. Par exemple, si la contrainte sur a est de type `algebraic` comme $a^2 + 1 = 0$, alors l'élément $b = a + 1$ de $\mathbf{L}$ peut être représenté aussi bien par $t + 1$ que par $(t^3 - 1)/t^2$. Si on a des contraintes de type `exception` comme $a^2 + 1 \neq 0$, alors l'élément $b = a + 1$ peut être représenté par $t + 1$ ou par $(t^3 + t^2 + t + 1)/(t^2 + 1)$. On peut cependant, pour chaque ensemble de contraintes, définir une représentation canonique, et munir l'ensemble de fractions utilisé :

- d'une relation d'équivalence (qu'on appelle *égalité grossière* ou `roughEqual`) pour reconnaître si deux fractions (dans notre exemple $t+1$ et $(t^3-1)/t^2$) représentent, à un moment donné du calcul, le même élément de $\mathbf{L}$. Noter que si deux éléments sont grossièrement égaux, alors ils sont *égaux*, mais la réciproque peut être fausse.

- et d'une fonction (qu'on appelle *réduction* ou `reduce`) pour passer d'une fraction au représentant canonique choisi dans sa classe d'équivalence.

Notons aussi que certaines fractions rationnelles de $\mathbf{K}(t)$ ne représentent aucun élément de $\mathbf{L}$: par exemple la fraction $1/(t^2+1)$, si la contrainte sur a est $a^2+1=0$, ou toute fraction $P(t)/Q(t)$ avec $Q(t)$ non constant, si a est du type `anyElement` (car a peut alors en particulier être une racine de $Q(t)$). Cependant ce type de fraction ne peut pas apparaître dans la représentation que nous avons choisie. La partie de $\mathbf{K}(t)$ utilisée pour la représentation des éléments connus de $\mathbf{L}$ est construite au fur et à mesure du calcul et, bien sûr, elle change pendant le calcul :

- Si a est du type `anyElement`, les seuls éléments de $\mathbf{L}$ auxquels on a accès sont des expressions polynomiales en a (de degré quelconque) et donc nous utilisons $\mathbf{K}[t]$ pour la représentation.

- Si a est du type `exception`, disons $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$, alors les polynômes $P_1, \ldots, P_k$ engendrent un ensemble multiplicatif S d'éléments non nuls et donc la représentation est fournie pour l'ensemble des fractions de $\mathbf{K}(t)$ avec dénominateur dans S.

- Si a est du type `algebraic`, disons $P(a)=0$, la représentation peut être donnée par l'ensemble des fractions de $\mathbf{K}(t)$ avec dénominateur $Q(t)$ tel que $pgcd(P,Q)=1$. Mais une telle fraction peut toujours se ramener à un polynôme de $\mathbf{K}[t]$ de degré strictement plus petit que le degré de P en utilisant l'égalité de Bezout.

3.3 Tests d'égalité et scindages

Passons maintenant au test d'égalité, et aux contraintes sur a. Il faut déterminer si la réponse à un test d'égalité `=` dans $\mathbf{L}$ est `true` pour toutes les valeurs possibles de a, ou `false` pour toutes les valeurs possibles de a, ou si les deux réponses sont possibles. Et dans ce dernier cas il faut préciser l'ensemble de valeurs possibles pour la réponse `true`, et l'ensemble de valeurs possibles pour la réponse `false`. Cela s'appelle un *scindage*.

Puisque l'ensemble des valeurs possibles associé à un paramètre est décrit par une contrainte, un scindage est en fait une opération sur les contraintes, donc répondre à un test d'égalité `=` signifie donner la nouvelle contrainte sur a pour que la réponse soit `true`, et la nouvelle contrainte pour qu'elle soit `false`. Ces modifications doivent de plus donner des contraintes de l'un des trois types annoncés.

Remarque Soit $b \in \mathbf{L} = \mathbf{K}(a)$ tel que $b \in \mathbf{K}$, alors répondre au test "$b = 0$?" ne change pas la contrainte sur a.

Lemme 3.2 *Soit $b \in \mathbf{L} = \mathbf{K}(a)$ tel que $b \notin \mathbf{K}$. Le test "$b = 0$?" peut se ramener à :*

- *ou bien un test "$b' = 0$?" avec $b' \in \mathbf{K}$*
- *ou bien un test "$B(a) = 0$?" avec $B \in \mathbf{K}[t]$ polynôme unitaire et de degré > 0.*

Démonstration
Soit $b \in \mathbf{L}$ de représentant $F \in \mathbf{K}(t)$. Posons $F = B_1/B_2$ avec $B_1,\ B_2 \in \mathbf{K}[t]$ et tel que $B_2(a) \neq 0$.

Si le degré de B_1 est nul, posons $b' = B_1 \in \mathbf{K}$. Alors la question "$b = 0$?" est équivalente à la question "$b' = 0$?".

Si le degré de B_1 est > 0, soit B le polynôme unitaire associé à B_1. Alors la question "$b = 0$?" est équivalente à la question "$B(a) = 0$?".

Définition Soit P et B deux polynômes unitaires dans $\mathbf{K}[t]$. Définissons *la partie première de P par rapport à B*, noté $pp(P, B)$, comme le facteur de plus haut degré de P qui est premier à B.

Remarque Nous pouvons écrire :

$$P = pp(P, B) \times pgcd(P, B) \times C$$

avec C tel que tout diviseur irréductible de C dans $\mathbf{K}[t]$ soit un diviseur de B. Alors, en particulier, tout diviseur irréductible de C divise $pgcd(P, B)$, et donc toute racine de P est racine de $pgcd(P, B)$ ou (exclusif) de $pp(P, B)$.

Décidons de toujours choisir $pgcd(P, B)$ et $pp(P, B)$ unitaires. Notons que si $D = pgcd(P, B)$ alors $pp(P, B) = pp(P, D)$.

Pour répondre aux tests d'égalité, nous utilisons les algorithmes suivants, en relation avec la partie première d'un polynôme par rapport à un autre :

Algorithme 1 *Partie-première : $pp(P, Q)$*
Entrée : *$P,\ Q \in \mathbf{K}[t]$ non nuls et tels que Q divise P*
Sortie : $pp(P, Q)$
début

```
E:= P
D:= Q
while deg(D) > 0 repeat
    E:= E/D
    D:= pgcd(E, D)
return E
```

fin

Algorithme 2 *Décomposition : $dec(P, Q)$*
Entrée : $P,\ Q \in \mathbf{K}[t]$ *non nuls*
Sortie : $[pp(P,Q),\ pgcd(P,Q)]$
début

> D:= $pgcd(P,Q)$
> E:= $pp(P,D)$
> `return` $[E,\ D]$

fin

Algorithme 3 *Décomposition multiple : $decM(P_1,\ \ldots,\ P_k, Q)$*
Entrée : $P_1,\ \ldots,\ P_k,\ Q \in \mathbf{K}[t]$ *non nuls*
Sortie : $[[pp(P_1,Q),\ pgcd(P_1,Q),\ \ldots,\ pp(P_k,Q),\ pgcd(P_k,Q)]],\ pp(Q, P_1 \times \ldots \times P_k)$
début

> E:= Q
> `for` i `in` *1...k* `repeat`
> > D_i := $pgcd(P_i, Q)$
> > E_i := $pp(P_i, D_i)$
> > E := $pp(E, D_i)$
>
> L := $[E_1, D_1,\ \ldots,\ E_k, D_k]$
> `return` $[L,\ E]$

fin

Théorème 3.3 *Soit a un paramètre sur $\hat{\mathbf{K}}$ et $B \in \mathbf{K}[t]$ un polynôme unitaire et de degré > 0, alors il est possible de répondre au test "$B(a) = 0$?" de façon que la nouvelle contrainte sur a est ou bien de type* `algebraic` *ou bien de type* `exception`*.*

Démonstration
Il y a trois cas à envisager, selon le type de la contrainte sur a juste avant le test d'égalité :

- Si a est de type `anyElement` alors les deux réponses sont possibles. Précisément la réponse est `true` si a est soumis à la contrainte $B(a) = 0$ (de type `algebraic`) et `false` s'il est soumis à la contrainte $B(a) \neq 0$ (de type `exception`).

- Si a est de type `exception`, disons $P_1(a) \neq 0,\ \ldots,\ P_k(a) \neq 0$, alors soit $[L = [E_1,\ D_1,\ \ldots,\ E_k, D_k],\ E]$ le résultat de $decM(P_1,\ \ldots,\ P_k, B)$ (algorithme 3).

  Si $E = 1$ alors la réponse est `false` et la contrainte reste $P_1(a) \neq 0,\ \ldots,\ P_k(a) \neq 0$, mais nous allons maintenant l'écrire $E_1(a) \neq 0,\ D_1(a) \neq 0,\ \ldots,\ E_k(a) \neq 0,\ D_k(a) \neq 0$ en enlevant tout polynôme de degré 0 de cette liste.

Sinon la réponse est `true` si a vérifie $E(a) = 0$ (de type `algebraic`) et `false` si a vérifie $E_1(a) \neq 0,\ D_1(a) \neq 0,\ \ldots,\ E_k(a) \neq 0,\ D_k(a) \neq 0,\ E(a) \neq 0$ (de type `exception`) en enlevant aussi tout polynôme de degré 0 de cette liste.

- Si a est de type `algebraic`, disons $P(a) = 0$, alors soit $[E,\ D]$ le résultat de $dec(P,\ B)$ (algorithme 2).

  Si $D = 1$ alors la réponse est `false` et la contrainte sur a reste $P(a) = 0$. Si $E = 1$ alors la réponse est `true` et la contrainte sur a reste $P(a) = 0$.

  Enfin dans les autres cas la réponse est `true` si $D(a) = 0$ et `false` si $E(a) = 0$. Dans ce cas on obtient toujours des contraintes du type `algebraic`.

3.4 Imposition de contraintes

Nous avons vu comment fonctionne le test d'égalité dans la clôture et le changement de la contrainte sur a lors d'un test d'égalité. Voyons maintenant le fonctionnement de `areDifferent` et `areEqual`. Nous considérons ici seulement l'opérateur `areDifferent` : le rôle de `areEqual` est "symétrique".

Remarque Soit $b \in \mathbf{L} = \mathbf{K}(a)$ tel que $b \in \mathbf{K}$, alors `areDifferent(b,0)` répond `true` si $b \neq 0$ et `false` si $b = 0$.

Remarque

- Soit $b \in \mathbf{L} = \mathbf{K}(a)$ tel que $b \in \mathbf{K}$, alors `areDifferent(b,0)` ne change pas la contrainte sur a.
- Soit $b,\ c \in \mathbf{L} = \mathbf{K}(a)$, alors imposer `areDifferent(b,c)` est équivalent à imposer `areDifferent(b-c,0)`

Lemme 3.4 *Soit $b \in \mathbf{L} = \mathbf{K}(a)$ tel que $b \notin \mathbf{K}$, alors* `areDifferent(b,0)` *peut se ramener à :*

- *ou bien* `areDifferent(b',0)` *avec $b' \in \mathbf{K}$*
- *ou bien imposer la contrainte "$B(a)$* est différent de 0*" sur a avec $B \in \mathbf{K}[t]$ polynôme unitaire et de degré > 0.*

Démonstration
Soit $b \in \mathbf{L}$ de représentant $F \in \mathbf{K}(t)$. Posons $F = B_1/B_2$ avec $B_1,\ B_2 \in \mathbf{K}[t]$ et tel que $B_2(a) \neq 0$.

Si le degré de B_1 est 0, posons $b' = B_1 \in \mathbf{K}$. Alors `areDifferent(b,0)` est équivalent à `areDifferent(b',0)`.

Si le degré de B_1 est > 0, soit B le polynôme unitaire associé à B_1. Alors `areDifferent(b,0)` est équivalent à imposer la contrainte "$B(a)$ *est différent de 0*" sur a.

Théorème 3.5 *Soit a un paramètre sur $\hat{\mathbf{K}}$ et $B \in \mathbf{K}[t]$ un polynôme unitaire et de degré > 0, s'il est possible d'imposer la contrainte "$B(a)$ est différent de 0" sur a alors la nouvelle contrainte sur a soit ou bien de type* `algebraic` *ou bien de type* `exception`.

Démonstration
Pour cela nous devons considèrer trois cas, selon la contrainte sur a juste avant `areDifferent` :

- Si a est de type `anyElement` alors nous pouvons imposer $B(a) \neq 0$ (qui est une nouvelle contrainte de type `exception` sur a) et `areDifferent` répond `true`.

- Si a est de type `exception`, disons $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$, alors soit $[L = [E_1, D_1, \ldots, E_k, D_k], E]$ le résultat de $decM(P_1, \ldots, P_k, B)$ (algorithme 3).

  Si $E = 1$ alors la contrainte $B(a) \neq 0$ est déjà vérifiée, la contrainte sur a reste $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$, écrite comme $E_1(a) \neq 0$, $D_1(a) \neq 0, \ldots, E_k(a) \neq 0$, $D_k(a) \neq 0$ en enlevant tout polynôme de degré 0 de cette liste et `areDifferent` répond `true`.

  Sinon, la contrainte $B(a) \neq 0$ n'est pas vérifiée et il faut l'ajouter : la nouvelle contrainte est $E_1(a) \neq 0$, $D_1(a) \neq 0, \ldots, E_k(a) \neq 0$, $D_k(a) \neq 0$, $E(a) \neq 0$ (de type `exception`), et `areDifferent` répond `true`.

- Si a est de type `algebraic`, disons $P(a) = 0$, alors soit $[E, D]$ le résultat de $dec(P, B)$ (algorithme 2).

  Si $D = 1$ alors la contrainte $B(a) \neq 0$ est déjà vérifiée, la contrainte sur a reste $P(a) = 0$ et `areDifferent` répond `true`. Si $E = 1$ alors la contrainte $B(a) \neq 0$ ne peut pas être imposée, la contrainte sur a reste $P(a) = 0$ et `areDifferent` répond `false`.

  Dans les autres cas nous devons imposer $B(a) \neq 0$: la nouvelle contrainte est $E(a) = 0$ et `areDifferent` répond `true`.

Remarque Si l'on ne peut pas imposer une nouvelle contrainte sur un paramètre, on ne change pas la contrainte déjà existante.

Remarque Ceci nous montre qu'on peut considérer que l'instruction `areDifferent(x,y)` (*resp.* l'instruction `areEqual(x,y)`) provoque le même scindage que le test "`x = y`", mais qu'ensuite on ne s'intéresse plus qu'à la branche où la réponse au test est `false` (*resp.* où la réponse au test est `true`).

Remarque Avec cette démonstration nous avons fini la démonstration du théorème 3.1.

3.5 Arbre de scindages

Ainsi, à chaque calcul dans $\mathbf{L} = \mathbf{K}(a)$ on associe un arbre, appelé *arbre de scindages* du calcul, et construit de la manière suivante :

1. A chaque nœud de cet arbre il y a :
   - ou bien le nœud racine qui correspond à l'appel de `newElement`
   - ou bien un test d'égalité `b = c` (où b et c sont deux éléments de $\mathbf{L}$) avec trois possibilités pour la réponse :
     - `true` si $b = c$ pour toutes les valeurs possibles de a. Ce nœud a un seul fils.
     - `false` si $b \neq c$ pour toutes les valeurs possibles de a. Ce nœud a un seul fils.
     - `true` et `false` si les deux réponses sont possibles. Ce nœud a *deux* fils, un pour chaque réponse.
   - ou bien `areEqual(b,c)` avec deux possibilités pour la réponse :
     - `true` s'il est possible d'imposer la contrainte demandée, et alors le système l'impose. Ce nœud a un seul fils.
     - `false` s'il est impossible d'imposer la nouvelle contrainte. Alors le système ne modifie pas les contraintes, et c'est à l'utilisateur de gérer la situation. Ce nœud a un seul fils.
   - ou bien `areDifferent(b,c)` avec deux possibilités pour la réponse :
     - `true` s'il est possible d'imposer la contrainte demandée, et alors le système l'impose. Ce nœud a un seul fils.
     - `false` s'il est impossible d'imposer la nouvelle contrainte. Alors le système ne modifie pas les contraintes, et c'est à l'utilisateur de gérer la situation. Ce nœud a un seul fils.
2. A chaque arête de cet arbre correspond une contrainte sur a qui exprime les valeurs possibles de a à ce moment du calcul.

3.6 Calculs dans la clôture

Un calcul sur la clôture constructible dynamique `CL` *de* $\mathbf{K}$, est une combinaison d'opérations `+`, `-`, `*` et `inv`, à partir des constantes `0`, `1`, des éléments introduits par `newElement`, avec des tests d'égalité, ainsi que des instructions `areEqual` et `areDifferent`, et les "structures de contrôle" habituelles comme "`if... then... else...`" ou "`while... repeat...`".

Effectuer un calcul dans `CL` consiste essentiellement à la construction de l'arbre de scindages associé.

Par exemple :

```
a:= newElement('a)
if areDifferent(a**4,1) then
  b:= a**2 + a
  if b = 2 then return 0
  else return 1/(b-2)
else return 0
```

Effectuer ce calcul revient à construire l'arbre de scindages suivant :

$$
\begin{tikzcd}
 & \texttt{newElement('a)} \arrow[d, "any\ a"] & \\
 & \texttt{areDifferent(a**4,1)} \arrow[d, "a^4 - 1 \neq 0"] & \\
 & \texttt{a**2 + a = 2} \arrow[dl, "\substack{a^3 + a^2 + a + 1 \neq 0 \\ and\ \ a \neq 1 \\ and\ \ a \neq -2}"'] \arrow[dr, "a = -2"] & \\
\frac{1}{(a-1)(a+2)} & & 0
\end{tikzcd}
$$

Le nœud racine de cet arbre correspond à l'appel de `newElement`. Au nœud suivant, on essaie d'imposer la contrainte `areDifferent(a**4,1)`. Puisque c'est possible, le système l'impose et retourne `true`. Ensuite, on trouve un nœud où il y a un test d'égalité avec deux possibilités pour la réponse, il en part donc deux arêtes, celle de gauche correspond à `false` et celle de droite à `true`.

Enfin on obtient le résultat :

```
[ value is  0  in case  a = - 2,
                 1                  3   2
  value is  ----------  in case  a + a + a + 1/= 0, a/= 1, a/= -2 ]
          (a-1) (a+2)
```

Bien sûr le cas où $a^4 - 1 = 0$ n'est pas obtenu.

4 Implantation

L'implantation de la clôture constructible repose sur la méthode de l'évaluation dynamique, de la même manière que l'implantation de la clôture algébrique [DD].

Nous décrivons ici l'implantation réalisée et les outils dont nous avons eu besoin, entre autres l'évaluation dynamique et les ensembles dynamiques. Mais le lecteur ne trouvera pas ici une définition rigoureuse de ces concepts.

4.1 Evaluation dynamique

En général, le processus de calcul implanté dans les systèmes de calcul existants est *l'évaluation statique* : chaque question intervenant dans un calcul donné possède une seule réponse. Par exemple, la réponse à un test d'égalité est (*toujours*) vrai ou (exclusif !) (*toujours*) faux.

Nous avons montré une situation plus générale que la précédente : une question dans un calcul peut avoir plusieurs réponses, chacune correspondant à un cas différent. Par exemple, nous avons vu que la réponse à un test d'égalité peut être *quelquefois* vrai et *quelquefois* faux : les deux réponses doivent être considérées et le calcul doit continuer pour chaque réponse.

L'évaluation dynamique est le processus de calcul qui tient compte de l'existence de cette situation, et qui gère le calcul de façon que l'on considère les deux réponses, chacune dans le cas correspondant. Le *résultat* du calcul consiste en la réunion de tous les résultats possibles, chacun associé à un cas différent.

Cela nous montre que l'évaluation dynamique est quelque chose de très général, et en particulier, on y trouve le processus de calcul "traditionnel". En fait, nous étudions ici une seule application : le calcul avec des paramètres dans un corps.

Le lecteur peut trouver une définition rigoureuse de l'évaluation dynamique en utilisant la théorie des esquisses : [DR].

Puisque les systèmes de calcul existants ne tiennent pas compte de l'évaluation dynamique, son implantation n'est pas une tâche simple. Malgré cela, des implantations de l'évaluation dynamiques ont été réalisées par C. Dicrescenzo et D. Duval [DDD, DD] : une première implantation dans le système Reduce, connue sous le nom de **D5** et qui a été utilisée pour le calcul avec des nombres algébriques ; une deuxième implantation dans le système Axiom, sur laquelle repose notre programme de la clôture constructible dynamique.

Dans ces implantations l'évaluation dynamique est seulement *simulée*, ce qui nous fait perdre de l'efficacité dans les gros calculs. Une vraie implantation de ce processus demande des systèmes de calcul conçus pour tenir compte de l'évaluation dynamique.

La fonction la plus importante de l'évaluation dynamique est l'opérateur `allCases` qui, à partir d'un calcul à effectuer, gère le parcours de toutes les branches de l'arbre de scindages associé à ce calcul. C'est cet opérateur qui permet d'obtenir les solutions dans *tous* les cas. Nous voyons l'utilisation de `allCases` dans la section 5.

4.2 Ensembles dynamiques

Un *ensemble dynamique* est un ensemble avec les outils nécessaires pour effectuer l'évaluation dynamique.

Un des faits essentiels dans l'évaluation dynamique est que "*l'information dont on dispose*" à chaque moment du calcul peut changer.

En général, l'information dont on dispose sur un élément est décrite dans sa représentation, et nous avons vu en la section 3.2 que cette représentation peut changer.

Parfois, il existe aussi des éléments spéciaux (par exemple les paramètres dans le cas de la clôture constructible dynamique) pour lesquels l'information n'est pas décrite dans la représentation mais dans les contraintes qu'ils vérifient, et nous avons vu dans les sections 3.3 et 3.4 comment elles peuvent changer.

Cela nous amène à distinguer deux types d'ensembles dynamiques :

- *ensembles de dynamisme actif* : comme par exemple la clôture algébrique dynamique et la clôture constructible dynamique.

  Les ensembles de dynamisme actif sont caractérisés par l'existence de ces éléments spéciaux, appelés *noyaux*. Ces éléments possèdent la propriété de susciter le dynamisme : une question faisant intervenir des noyaux peut produire des scindages.

  Le corps premier de caractéristique arbitraire est aussi de ce type, son seul noyau est la caractéristique.

- *ensembles de dynamisme passif* : dans ce type d'ensembles nous trouvons les polynômes ou les matrices à coefficients dynamiques puisque leur test d'égalité se ramène au test d'égalité sur les coefficients.

  Les ensembles de dynamisme passif ne contiennent pas de noyaux, et les questions (s'il y en a) ne provoquent pas "directement" de scindage ; ces ensembles font appel à des ensembles de dynamisme actif où ont lieu les scindages.

  Les ensembles dynamiques que nous étudions dans la section 4.3 sont aussi des ensembles de dynamisme passif sauf la clôture constructible dynamique.

La caractéristique commune à ces deux types d'ensembles est qu'un élément peut être représenté par des expressions différentes correspondant à des moments différents du calcul. Nous devons donc fournir les opérations nécessaires pour permettre au calcul de "naviguer" entre ces représentations. Ces opérations sont *la réduction* et *l'égalité grossière* introduites dans la section 3.2 ; ce sont les outils que possèdent les ensembles dynamiques pour effectuer l'évaluation dynamique.

Pour une définition plus "détaillée" des ensembles dynamiques le lecteur peut lire [Du].

A partir des ensembles dynamiques, on peut construire des *ensembles dynamiques structurés* [Du] : nous utilisons des anneaux et des corps dynamiques.

4.3 Clôture constructible dynamique

Nous introduisons ici les domaines (au sens Axiom) principaux que nous avons construits pour le programme de la clôture constructible dynamique. Ces domaines sont utilisés pour :

- les contraintes sur les paramètres : polynômes et contraintes.
- les éléments de la clôture : polynômes, dénominateurs et fractions.

Ces domaines ont des relations étroites entre eux, qui peuvent sembler compliquées mais qui, en fait, sont très simples quand on n'oublie pas la récursion.

Nous utilisons, pour cette présentation, un peu de terminologie Axiom ...

4.3.1 Polynômes dynamiques

```
DynamicUnivariatePolynomial(R)
  where R: DynamicRingCategory
```

Ce domaine des polynômes dynamiques est très important dans le programme de la clôture constructible, de même que dans la clôture algébrique dynamique [DD, Du]. Il est utilisé pour la représentation des éléments de la clôture et aussi dans la représentation des contraintes sur les paramètres ; de plus l'opération de scindage repose entièrement sur le calcul du pgcd de ces polynômes. Donc, nous avons besoin d'un domaine le plus efficace possible.

Nous utilisons `DynamicUnivariatePolynomial` avec `CL` comme argument. (Attention à la récursion !).

La *représentation* de ses éléments consiste en la liste de ses coefficients (qui sont donc dans `CL`).

La *réduction* d'un polynôme de ce domaine est la réduction de ses coefficients, plus une présentation du polynôme avec le coefficient de tête grossièrement différent de zéro.

L'*égalité grossière* de polynômes revient aussi à l'égalité grossière de ses coefficients.

L'opération la plus importante est le calcul de pgcd, effectué par un algorithme de sous-résultants [DD, Du].

4.3.2 Contraintes dynamiques

```
DynamicConstructibleLaw(K)
  where K: DynamicFieldCategory
```

C'est le domaine crèe pour la manipulation des contraintes (`laws`) sur les paramètres. Nous avons vu dans la section 3 les types de contraintes et comment fonctionne le scindage des contraintes. Rappelons que les polynômes qui apparaissent dans les contraintes sont toujours en une variable, unitaires, de degré > 0, sans facteur commun dans le cas `exception` et enfin, sans facteur multiple lorsque la caractéristique du corps est nulle.

La *représentation* des contraintes est faite essentiellement avec des polynômes dynamiques ; mais *de plus* nous avons utilisé la représentation des contraintes pour *emmagasiner* des informations additionnelles. Ces informations proviennent des contraintes, mais elles seront utilisées ailleurs. Avant d'expliquer mieux ce point, nous devons connaître les différentes composantes de cette représentation.

Polynômes généralisés Pour la représentation des contraintes, nous avons introduit des *polynômes généralisés*. Un polynôme généralisé est un polynôme dynamique, mais pour optimiser quelques calculs, il est "accompagné" de certaines informations.

```
NNI  ==> NonNegativeInteger
Poly ==> DynamicUnivariatePolynomial(K)
Rule ==> Union(linear: K, nonlinear: Poly)
  -- Generalized Polynomial:
Gpol ==> Record(rul: Rule, deg: NNI, fla: Flag, mul: NNI)
```

Cela veut dire qu'un polynôme généralisé est composé de 4 champs :

- Lorsque `mul` $= 1$ alors :
  - `rul` est :
    - ou bien un polynôme unitaire de degré > 1
    - ou bien une constante `r` de `K` qui représente le polynôme $x - r$
  - `deg` est le degré du polynôme de `rul`
  - `fla` est une "étiquette" contenant une information sur le polynôme de `rul` :
    - `fla` est `prime` si on sait que le polynôme est premier, comme par exemple dans le cas de degré 1,
    - `fla` est `squarefree` si on sait que le polynôme est sans carré, comme par exemple dans le cas de caractéristique nulle,
    - `fla` est `noFlag` si on n'a pas d'information sur le polynôme.
- Lorsque `mul` $= 0$, le polynôme généralisé n'est pas utilisé dans les contraintes mais il garde des informations qui seront utilisées dans les dénominateurs ou les fractions (ce point est expliqué en détail ci-dessous).

Alors la *représentation* d'une contrainte est :

```
Rep:= Record(lgpol: List Gpol, typ: Boolean, sym: Symbol)
```

où :

- `lgpol` est une liste de polynômes généralisés. Nous verrons ci-dessous que l'ordre de disposition des polynômes dans `lgpol` est important.
- `typ` indique le type de contrainte :
  - `typ` est `true` si la contrainte est de type `exception` (la contrainte est de type `anyElement` si `typ` est `true` et `lgpol` est la liste vide)
  - `typ` est `false` si la contrainte est de type `algebraic`
- `sym` est le symbole qui représente le noyau (`constructibleKernel`) auquel correspond la contrainte.

Exemples

1. Si par exemple a est de type `anyElement` alors cette contrainte sur a est représenté par :

```
[[ ], true, a]
```

2. Si a est de type `exception`, par exemple $a - 1 \neq 0$, sur un corps de caractéristique 0, la représentation de la contrainte est :

```
[[[1, 1, prime, 1]], true, a]
```

3. Si a est de type `algebraic`, par exemple $a^3 - 2 = 0$, sur un corps de caractéristique 0, la représentation de la contrainte est :

```
     3
[[[t  - 2, 3, squarefree, 1]], false, a]
```

   Cette représentation pourrait aussi être donnée par :

```
     3                            2
[[[t  - 2, 3, squarefree, 1], [t   + t + 1, 1, prime, 0]],
 false, a]
```

   (cet exemple est traité en détail ci-dessous)

4. Si a est de type `exception`, par exemple $(a-1)^2(a-3)^2 \neq 0$, sur un corps de caractéristique 5 (où $(a-1)^2(a-3)^2 = a^4+2a^3+2a^2+a+4$), la représentation de la contrainte est :

```
     4      3      2
[[[t  + 2t  + 2t  + t + 4, 4, noFlag, 1]], true, a]
```

5. Si a est de type `exception`, par exemple $(a-3)^2 \neq 0$, $(a-1) \neq 0$, sur un corps de caractéristique 5, la représentation de la contrainte peut être :

```
       2
[[[t  - t + 4, 2, noFlag, 1], [1, 1, prime, 1],
  [1, 1, prime, 0]], true, a]
```

(cet exemple est traité en détail ci-dessous).

Cas avec `mul` = 0 Il y a deux situations spéciales dans lesquelles il apparaît des polynômes généralisés avec `mul` $= 0$:

- dans le scindage d'une contrainte de type `exception` (différente de `anyElement`) avec une nouvelle contrainte sur le paramètre de type `algebraic`. Par exemple si $q(a) = 0$ est la nouvelle contrainte algébrique qui vient du scindage de $p(a) \neq 0$, alors la liste des polynômes généralisés dans sa représentation est :

  ```
  [[rul(q), deg(q), fla(q), 1], [1/p mod q, deg(p), fla(p), 0]]
  ```

  L'exemple (3) précédent correspond à cette situation :

  - la première représentation de la contrainte $a^3 - 2 = 0$ sur a s'obtient dans le scindage de la contrainte `anyElement` par rapport au polynôme $t^3 - 2$.
  - la deuxième représentation s'obtient dans le scindage de la contrainte $a - 1 \neq 0$ par rapport au polynôme $t^3 - 2$, le deuxième élément de la liste de polynômes généralisés $(t^2 + t + 1)$ est l'inverse de $t - 1$ modulo $t^3 - 2$. Noter que les parties correspondant au degré et au `flag` contiennent les informations sur $t - 1$.

  Cela correspond dans le cas algébrique à ce que tout dénominateur différent de 1 se ramène au numérateur (voir le domaine des dénominateurs pour un exemple).

- dans le cas où la caractéristique est non nulle, les polynômes qui apparaissent dans les contraintes ne sont pas forcément sans carré, et le scindage des contraintes de type `exception` a des répercussions dans la représentation choisie pour les dénominateurs. Si $p(a) \neq 0$, $q(a) \neq 0$ est la nouvelle contrainte de type `exception` qui vient du scindage de $p(a) \neq 0$ par rapport à q, alors nous posons :

  $$p_1 = pp(p, q)$$
  $$p_2 = pgcd(p, q)$$
  $$c = \frac{p}{p_1 p_2}$$

$p_3 = pp(q, p_2)$

La contrainte $p(a) \neq 0,\ q(a) \neq 0$ a la liste suivante de polynômes généralisés dans sa représentation :

```
[[rul(p1), deg(p1), fla(p1), 1], [rul(p2), deg(p2), fla(p2), 1],
 [rul(c),  deg(c),  fla(c),  0], [rul(p3), deg(p3), fla(p3), 1]]
```

Ceci est la liste complète, mais s'il y avait des polynômes constants, ils n'apparaîtraient pas dans cette liste.

L'exemple (5) précédent correspond à cette situation : nous avons le paramètre a sur un corps de caractéristique 5 avec la contrainte $(a-1)^2(a-3)^2 \neq 0$ (donc $p = (a-1)^2(a-3)^2$) et on teste si "$a - 1 = 0$?" (donc $q = a - 1$). Nous considérons la branche de l'arbre des scindages où la réponse est `false`, et où la nouvelle contrainte sur a est $(a-3)^2 \neq 0,\ (a-1) \neq 0$. La représentation de cette nouvelle contrainte est :

```
      2
 [[[t  - t + 4, 2, noFlag, 1], [1, 1, prime, 1],
   [1, 1, prime, 0]], true, a]
```

où le polynôme généralisé correspondant à p_3 n'apparaît pas puisqu'il est constant. Nous montrons un exemple plus détaillé dans le domaine des dénominateurs.

Dans l'*égalité grossière* de deux contraintes nous commençons par tester si les contraintes sont du même type. En ce cas on teste l'égalité grossière des polynômes qui interviennent dans la contrainte (c'est-à-dire, avec `mul`=1 et placés dans des positions correspondantes dans les listes de chaque contrainte).

La *réduction* d'une contrainte consiste en la réduction des polynômes avec `mul`=1.

Dans ce domaine il y a des opérations, dont la plus importante est le scindage (`split`), qui changent la contrainte actuelle sur le paramètre : elles ne peuvent être utilisées que depuis le domaine de la clôture constructible dynamique.

Considérer des polynômes toujours sans carré dans le cas de caractéristique nulle nous a posé encore un autre problème avec les contraintes de type `exception`. Ce problème correspond à la situation où une expression, dont on sait qu'elle est non nulle, ne peut pas être exprimée comme un dénominateur.

Par exemple, si on a a de type `anyElement` et on pose la question "$(a-2)(a-3)^2 = 0$?", la contrainte obtenue pour le cas de réponse `false` est $(a-2)(a-3) \neq 0$. Alors l'expression $(a-2)(a-3)^2$ est non nulle, mais elle ne peut pas être exprimée comme un dénominateur, c'est-à-dire, on n'a pas accès à l'élément $\frac{1}{(a-2)(a-3)^2}$, parce que l'expression n'est pas une puissance de $(a-2)(a-3)$. Cet exemple est traité plus en détail dans le domaine des dénominateurs.

En ce cas le scindage fait du travail supplémentaire : une décomposition du polynôme dans la contrainte en facteurs sans carré, de façon qu'on évite le problème. Dans notre exemple, la contrainte sera $a-2 \neq 0, a-3 \neq 0$ au lieu de $(a-2)(a-3) \neq 0$.

Ce problème, de même que les précédents (cas de `mul=0`), est lié au domaine des dénominateurs, et il sera mieux compris après avoir étudié ce domaine.

4.3.3 Représentation dynamique : fractions et dénominateurs

```
DynamicConstructibleFraction(K)
  where K: DynamicFieldCategory
```

Les éléments de ce domaine sont utilisés pour la représentation des éléments de `CL`. Nous avons expliqué en la section 3.2 les points essentiels de cette représentation.

La *représentation* d'une fraction dynamique est composée d'un polynôme dynamique et d'un dénominateur dynamique :

```
Deno ==> DynamicConstructibleDenominator(K)
Poly ==> DynamicUnivariatePolynomial(K)

Rep:= Record(num: Poly, den: Deno)
```

Nous avons vu dans la section 4.3.1 les polynômes, voyons les dénominateurs.

Dénominateurs dynamiques

```
DynamicConstructibleDenominator(K)
  where K: DynamicFieldCategory
```

Cet ensemble est probablement l'ensemble dynamique le plus curieux que nous ayons trouvé (nous avons pensé dans un premier temps qu'il n'était pas du tout dynamique ...). En fait le dynamisme de la clôture constructible montre son influence sur cet ensemble de deux façons différentes. Commençons par dire ce qu'est un dénominateur.

Un *dénominateur* représente un élément de la clôture dont on sait qu'il est *non nul*. Cette définition dépend donc de la contrainte sur le noyau :

- Si la contrainte est `anyElement`, le seul dénominateur est 1.
- Si la contrainte est `exception`, disons $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$, alors les éléments de la clôture $e_1 = P_1(a), \ldots, e_k = P_k(a)$ engendrent l'ensemble multiplicatif
$$S = \{e_1^{n_1} \cdot \ldots \cdot e_k^{n_k} \mid (n_1, \ldots, n_k) \in \mathbf{N}^k\}$$
d'éléments non nuls dans la clôture. Un dénominateur est un élément quelconque de S.

- Si la contrainte est `algebraic`, disons $P(a) = 0$, tout polynôme Q avec $pgcd(P, Q) = 1$ représente un élément non nul de la clôture. Dans ce cas, en utilisant l'identité de Bezout, on peut se ramener à une fraction avec dénominateur 1.

  Lorsque la contrainte algébrique provient du scindage d'une contrainte de type `exception`, la représentation de la nouvelle contrainte garde des informations pour faciliter cette opération. Cela correspond au premier cas de `mul`=0. Voyons un exemple :

  Reprenons l'exemple (3) considéré dans la section des contraintes. Nous avons le paramètre a avec la contrainte $a-1 \neq 0$ et l'élément de la clôture $e = \frac{1}{(a-1)^2}$. On pose la question "$a^3 - 2 = 0$?" et on considère la branche de l'arbre des scindages où la réponse est `true` avec une nouvelle contrainte algébrique sur $a : a^3 - 2 = 0$. L'élément e a une nouvelle expression : $e = (a^2 + a + 1)^2$.

  Donc, si la contrainte est de type `algebraic` (notons c_1 cette contrainte) et provient du scindage d'une contrainte de type `exception` (que nous notons c_2), alors l'ensemble des dénominateurs de c_1 est le même que l'ensemble des dénominateurs de c_2. Lors de la réduction, ces dénominateurs sont ramenés au numérateur. Dans les autres cas l'ensemble des dénominateurs est $\{1\}$.

En conséquence un dénominateur non-trivial correspond à un polynôme de la forme $P_1^{n_1} \cdot \ldots \cdot P_k^{n_k}$ en relation avec une contrainte de type `exception`. Et donc, en tant que polynôme (à coefficients dynamiques), il a une réduction et une égalité grossière associées ; elles sont utilisées pour la présentation externe d'un dénominateur. Cela correspond à la première influence du dynamisme de la clôture.

La deuxième influence correspond au changement des contraintes au fur et à mesure du calcul, mais elle correspond plus précisément au changement de l'ensemble S lors des contraintes de type `exception`. Par exemple, si on a une contrainte de type `exception` sur le noyau, elle peut devenir algébrique par un scindage et le dénominateur doit passer au numérateur. Nous avons montré un exemple ci-dessus et, en fait, cela correspond à l'opération de réduction dans le domaine des fractions. Nous nous intéressons ici seulement aux contraintes de type `exception`. Voyons ce qu'il arrive quand la nouvelle contrainte après un scindage est aussi de type `exception`.

Pour cela supposons qu'un dénominateur correspondant à la contrainte $P_1(a) \neq 0, \ldots, P_k(a) \neq 0$ est représenté par la liste des exposants : $(n_1, \ldots, n_k)$ (nous voyons ci-dessous la représentation choisie, pas très éloignée de celle-ci), et qu'après un ou plusieurs scindages, on a la nouvelle contrainte $P_1(a) \neq 0, \ldots, P_k(a) \neq 0, P_{k+1}(a) \neq 0, \ldots, P_l(a) \neq 0$, ce qui nous donne accès à des dénominateurs de longueur $l > k$. Mais maintenant, si nous voulons faire des opérations avec des dénominateurs, il peut arriver qu'ils aient des longueurs différentes, puisqu'ils correspondent à des moments différents du calcul. Cela nous montre que même si l'on choisit une représentation “indépendante” des coefficients dynamiques, le dynamisme se manifeste de toute façon.

Pour simplifier le plus possible les opérations sur les dénominateurs (puisque c'est possible et simple), nous avons choisi de les ramener au même point du calcul, ce qui correspond à ajouter des zéros à la fin du dénominateur “le plus ancien”. Cette opération s'appelle `arrange` et prend comme arguments deux dénominateurs. Nous pouvons aussi actualiser un dénominateur par rapport à une contrainte, mais cela complique les opérations et demande, de plus, la contrainte comme argument dans les opérations.

Nous avons choisi l'implantation de la fonction `reduce` dans ce contexte. La *réduction* d'un dénominateur consiste à effacer les zéros (s'il y en a) à la fin de la représentation. Voyons un exemple.

Exemple Considérons le paramètre a avec une contrainte $a-1 \neq 0$ et le dénominateur $d_1 =$`[2]` correspondant à l'élément $e_1 = \frac{1}{(a-1)^2}$. Après scindage, nous avons la contrainte $a-1 \neq 0,\ a-3 \neq 0$ et un nouveau dénominateur $d_2 =$`[0,1]` correspondant à l'élément $e_2 = \frac{1}{a-3}$.

Effectuer $e_1 e_2$ revient à $d_1 d_2$, et pour optimiser le produit des dénominateurs, nous faisons d'abord `arrange(d1, d2)` avec pour résultat `[[2,0], [0,1]]`. Cela signifie que nous avons actualisé la représentation du dénominateur `[2]` à `[2,0]` par rapport à d_2.

Maintenant nous pouvons avoir besoin de la représentation plus simple choisie pour le dénominateur d_1 et donc nous effectuons `reduce([2,0])` qui retourne `[2]`. Cela signifie que nous revenons à l'ancienne représentation de d_1 dans le calcul, et c'est plus optimal au niveau des opérations.

Cela nous montre aussi que l'opération `arrange` est une réduction d'un dénominateur par rapport à un autre. On modifie la représentation d'un même élément, en “avançant” dans l'arbre de scindages.

De même, tester l'*égalité grossière* consiste à tester l'égalité des listes que représentent les dénominateurs ; et nous avons implanté une *égalité* (qui ne produit pas de scindage, évidemment) qui teste l'égalité des listes que représentent deux dénominateurs, mais après réduction. Cette égalité est plus précise que l'égalité grossière.

Passons maintenant à la *représentation* :

```
Rep:= List NonNegativeInteger
```

Si la contrainte de type `exception` est donnée par la liste de polynômes $P_1, \ldots, P_k$ et d_i est le degré de P_i pour $i = 1, \ldots, k$, le dénominateur $P_1^{n_1} \cdot \ldots \cdot P_k^{n_k}$ est représenté par la liste de longueur $d_1 + \ldots + d_k$:

```
[n1, ..., n1, n2, ..., n2, ..., ..., nk, ..., nk]
```

où chaque n_i est répété d_i fois.

De cette façon la représentation du dénominateur n'est pas modifiée quand la contrainte $P_i(a) \neq 0$ s'écrit, après scindage, $Q_i(a) \neq 0, D_i(a) \neq 0$ avec Q_i et D_i deux facteurs de P_i pour $i = 1, \ldots, k$. Cela reste évident quand on travaille avec des polynômes sans carré en caractéristique nulle. Dans le cas de caractéristique différente de 0, la représentation des contraintes résout ce problème ; ceci correspond au deuxième cas de `mul`=0. Voyons un exemple :

Reprenons les exemples (4) et (5) considérés dans les contraintes. Nous avons le paramètre a sur un corps de caractéristique 5 et la contrainte sur a est $a^4 + 2a^3 + 2a^2 + a + 4 \neq 0$. Supposons qu'on ait le dénominateur `[1,1,1,1]` qui signifie $(a^4+2a^3+2a^2+a+4)^1$. Après le test "$a-1 = 0$?" (avec réponse `false`) nous avons la nouvelle contrainte $a^2-a+4 \neq 0,\ a-1 \neq 0$, mais représentée par $[a^2-a+4,\ mul = 1],\ [a-1,\ mul = 1],\ [a-1,\ mul = 0]$. La représentation du dénominateur ne change pas, mais maintenant il faut l'interpréter comme : $(a^2 - a + 4)^1(a - 1)^1(a - 1)^1$.

Les opérations usuelles dans l'ensemble multiplicatif S sont maintenant très simples, principalement le calcul de pgcd et le produit. Nous montrons ici le code correspondant à ces opérations :

```
de1 * de2 ==
  lde:= arrange(de1,de2)
  nn:= #lde.1
  [ lde.1.i + lde.2.i for i in 1..nn ]

gcd(de1,de2) ==
  lde:= arrange(de1,de2)
  de:= lde.1
  for i in 1..#de repeat
    if lde.2.i < de.i then de.i:= lde.2.i
  de
```

Les opérations les plus importantes sont les opérations de passage entre le codage du dénominateur et le polynôme qu'il représente : `makeDenominator` et `makePolynomial`. Ces deux fonctions ont besoin de la contrainte comme argument. Nous avons déjà une idée sur la fonction `makePolynomial`, voyons la fonction `makeDenominator`.

La tâche de `makeDenominator` est d'exprimer un polynôme comme un dénominateur par rapport à une contrainte de type `exception` si cela est possible. Donc ses arguments sont le polynôme et la contrainte. Supposons que la contrainte soit $P_1 \neq 0, \ldots, P_k \neq 0$ et considérons P un polynôme, alors on peut toujours écrire $P = P_1^{n_1} \cdot \ldots \cdot P_k^{n_k} \cdot R$, avec $n_i \geq 0$ et R non divisible par P_i, $i = 1, \ldots, k$.

- si R est constant, le résultat est `[true,[n1,...,nk],R]`, ce qui signifie que nous pouvons exprimer P (ou plutôt P/R) comme un dénominateur $[n1, ..., nk]$.

- dans le cas contraire, le résultat est `[false,[n1,...,nk],R]`, ce qui signifie que nous ne pouvons pas exprimer P comme le dénominateur.

Par exemple, si on a a tel que $(a-2)(a-3) \neq 0$ et $P = (a-2)(a-3)^2$ alors le résultat de `makeDenominator` est :

```
[false, [1,1], a-3]
```

mais si la contrainte est $a - 2 \neq 0,\ a - 3 \neq 0$, alors le résultat est :

```
[true, [1,2], 1]
```

Cette fonction est utilisée par exemple dans la réduction des fractions (voir ce domaine pour un exemple), mais elle est aussi utilisée dans un point important de la clôture. Puisque une partie de son résultat est booléenne, elle nous fournit un test grossier d'égalité à zéro.

Par exemple, si on a a tel que $a - 2 \neq 0,\ a - 3 \neq 0$ et nous devons faire le test "$(a-2)(a-3)^2 = 0$?", nous utilisons `makeDenominator` qui répond `true` *sans lancer des scindages.*

Une autre façon d'utiliser cette fonction est la suivante : si on a a tel que $(a-2)(a-3) \neq 0$ et nous voulons faire le test "$(a-2)(a-3)^2 = 0$?", alors `makeDenominator` répond `false`, mais maintenant la question "$(a-2)(a-3)^2 = 0$?" *est équivalente* à la question "$a - 3 = 0$?".

Nous avons trouvé dans l'article [GPT] des codages similaires aux dénominateurs constructibles. Ils sont utilisés pour limiter la croissance des coefficients dans le calcul des bases standard.

Fractions dynamiques

Rappelons qu'une fraction dynamique est composée d'un polynôme dynamique et d'un dénominateur dynamique. Nous avons choisi de coder les dénominateurs en fonction des polynômes d'une contrainte de type `exception`, et donc, il y a des opérations dans ce domaine qui ont besoin de cette contrainte comme argument, par exemple la somme de fractions, dont nous montrons ici le code :

```
addition(fr1,fr2,la) ==
  if algebraicType? la then
    p:= makePolynomial(fr1.den,la)$Deno
    q:= makePolynomial(fr2.den,la)$Deno
    nume:= (fr1.num * p) + (fr2.num * q)
    deno:= 1$Deno
  else
    deno:= lcm(fr1.den,fr2.den)$Deno
    de1:= (deno exquo fr1.den)::Deno
    de2:= (deno exquo fr2.den)::Deno
    p:= makePolynomial(de1,la)$Deno
```

```
    q:= makePolynomial(de2,la)$Deno
    nume:= (fr1.num * p + fr2.num * q)$Poly
  [nume,deno]
```

Cela nous montre l'utilisation de la contrainte dans ce type d'opérations : nous effectuons, autant que possible, des opérations sur les dénominateurs (puisqu'elles sont beaucoup moins coûteuses), mais parfois nous devons ramener les dénominateurs au numérateur en utilisant `makePolynomial`, d'où l'utilisation de la contrainte.

Il y a d'autres opérations qui deviennent extrêmement simples (la multiplication, les puissances), et enfin il y a des opérations qui dépendent du type de la contrainte, et qui agissent de façon différente selon ce type (comme la somme par exemple).

Dans ce dernier cas nous trouvons l'opération de *réduction* :

- Si la contrainte est `anyElement`, la réduction d'une fraction consiste en la réduction du numérateur (le dénominateur est 1).

- Si la contrainte est `exception`, la réduction consiste en une simplification par rapport à l'ensemble multiplicatif S : on applique la fonction `makeDenominator` et on effectue une simplification du dénominateur de ce résultat avec le dénominateur de la fraction. Voyons un exemple :

  Supposons que nous avons a tel que $(a-2)(a-3) \neq 0$ et l'élément $\frac{(a-2)(a-3)^3}{[(a-2)(a-3)]^2}$. Sa réduction commence par `makeDenominator` appliqué à $(a-2)(a-3)^3$ et à $(a-2)(a-3) \neq 0$. Le résultat est $[false,\ [1,1],\ (a-3)^2]$. Le dénominateur de la fraction est $[2,2]$, et on fait une simplification des deux dénominateurs :

$$\frac{(a-2)(a-3)^3}{[(a-2)(a-3)]^2} =$$

$$\frac{(a-2)(a-3)^3}{[2,2]} = \frac{[1,1]\cdot(a-3)^2}{[2,2]} = \frac{[0,0]\cdot(a-3)^2}{[1,1]} =$$

$$\frac{(a-3)^2}{(a-2)(a-3)}$$

- Si la contrainte est `algebraic`, la réduction consiste à ramener le dénominateur (s'il y en a un) au numérateur et à réduire le numérateur. Comme exemple nous avons l'exemple (3) dans le domaine des dénominateurs.

Cela signifie que la réduction des fractions est en fait une simplification "paresseuse".

L'*égalité grossière* des fractions constructibles consiste en l'égalité des dénominateurs et l'égalité grossière des numérateurs.

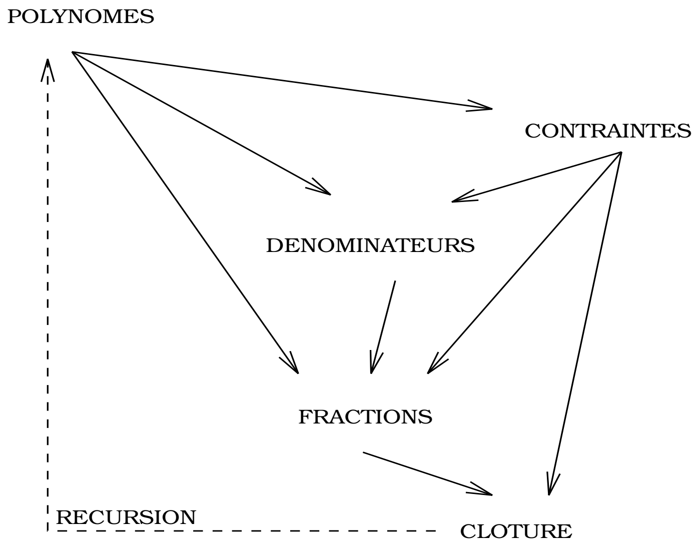
4.3.4 Clôture constructible dynamique

```
DynamicConstructibleClosure(K)
  where K: Field
```

Nous avons décrit ainsi tous les outils nécessaires pour la clôture constructible, dont la structure est similaire à celle de la clôture algébrique dynamique (mais avec une représentation un peu plus compliquée due aux fractions).

Nous avons vu que la construction de la clôture est récursive : on ajoute les noyaux les uns après les autres avec l'instruction `newElement`. Chaque noyau a un *niveau* associé : si a_1, a_2, $\ldots$, a_n sont les paramètres introduits, alors le niveau de a_i est i.

Ainsi dans l'introduction d'un nouveau noyau, on peut considérer les polynômes dynamiques avec des coefficients dans la partie de la clôture déjà construite et les utiliser dans la représentation.



L'égalité grossière et la réduction des éléments de la clôture reviennent à l'égalité grossière et la réduction des fractions.

Les opérations les plus importantes sont `newElement`, `areEqual` et `areDifferent` dont nous avons déjà parlé.

4.4 Améliorations

Beaucoup d'améliorations sont possibles, à des niveaux très différents. Quelques unes pourront se faire dans un futur (que nous attendons et souhaitons) proche. D'autres sont vues à terme.

Commençons par les améliorations "de bas niveau". Nous ne devons pas oublier que notre efficacité dépend essentiellement de l'efficacité de l'implantation de l'évaluation dynamique. Puisque cela n'est pas une tâche simple, nous pensons que cette amélioration ne pourra se faire qu'à plus long terme. Ceci est une des raisons pour lesquelles il ne nous paraît pas raisonnable de faire des comparaisons de temps de calcul avec d'autres programmes.

En ce qui concerne la programmation, il y a aussi beaucoup de choses à revoir. D'une part, nous connaissons beaucoup mieux maintenant le système Axiom et sa programmation. D'autre part, nous connaissons mieux le problème auquel nous faisons face (le calcul avec des paramètres) et quelques "petits problèmes" qu'il génère. Cela va nous permettre d'optimiser quelques points dans les programmes. Nous regardons ici de plus près quelques améliorations correspondant aux domaines décrits :

- **Polynômes.** Une première amélioration peut commencer par la reconsidération des polynômes généralisés. Il semble que construire un domaine efficace pour ces polynômes et les utiliser dans d'autres domaines (par exemple les fractions constructibles) peut optimiser beaucoup de calculs.

  Nous voulons aussi étudier différentes possibilités en relation avec le calcul de pgcd, en particulier dans le sens de [GPT] :

  the greatest between the common divisors that are easy to compute

- **Contraintes.** Nous voulons introduire deux nouvelles contraintes : le type `transcendent` et le type `failed`. Ces deux contraintes sont déjà plus ou moins considérées (de façon plutôt cachée) dans le programme, mais nous souhaitons un traitement plus propre.

- **Fractions.** Nous avons vu que dans le cas d'une contrainte algébrique qui provient d'une contrainte de type `exception`, les dénominateurs sont ramenés au numérateur par la réduction. Nous considérons la possibilité de continuer la manipulation des dénominateurs, même dans les contraintes algébriques qui ne proviennent pas des contraintes de type `exception`.

- **Clôture.** Dans tout ce programme il reste le problème de la *réduction*. De façon interne, les données utilisées pendant un calcul ne sont pas réduites en général. Cela signifie qu'on peut manipuler des données "grosses" qui ralentissent (beaucoup) les calculs, et qui en fait sont "petites" (après réduction). Cela est imposé par l'algorithme des sous-résultants implanté pour le calcul du pgcd, bien que cet algorithme soit optimal dans d'autres circonstances. Il semble qu'une bonne étude et un meilleur traitement de la réduction d'une part, et le choix d'un algorithme mieux adapté à notre problème d'autre part [AK], puissent nous aider à être plus efficaces.

5 Quelques applications et exemples

Dans cette section nous montrons quelques exemples de l'utilisation du programme de la clôture constructible dynamique. Nous avons divisé cette section en trois parties, chacune correspondant à une application :

- calculs avec des paramètres en général
- résolution des systèmes d'équations polynomiales
- démonstration automatique de théorèmes en géométrie.

Nous présentons des exemples avec les nombres rationnels comme corps de base, et donc nous considérons des paramètres complexes. Mais il est possible (grâce à la généricité d'Axiom et de notre méthode) de considérer comme corps de base un corps quelconque. D'autres exemples ont été présentés dans [Go93, DGD].

Guide d'utilisation

L'utilisation de ce programme peut être interactive, mais en général nous sommes intéressés par l'obtention des résultats complets, c'est-à-dire des résultats qui contiennent toutes les réponses possibles, ce qui est donné par la fonction `allCases`.

Pour cela nous pensons qu'il est plus simple pour les utilisateurs de construire un fichier (avec extension `.input` pour être reconnu par Axiom) en suivant celui que nous proposons ici :

```
++ Partie I
K:=    ++ corps de base
CL:= DynamicConstructibleClosure K
CC:= DynamicConstructibleCase K

++ Partie II
fonction(x:Type1):Type2 ==
  ++ Pas 1: introduction des parametres
  a:CL:= newElement('a)
  ...
  ++ Pas 2: operations a realiser
  if (a=1) then ...

++ Partie III
++ (DCTRL= DynamicControlPackage)
allCases(fonction,argument)$DCTRL(CC, Type1, Type2)
```

- **Partie I.** Dans cette partie, on introduit le corps de base souhaité et on construit la clôture constructible de ce corps. La fonction `allCases` a besoin aussi d'un autre domaine, `DynamicConstructibleCase`, pour la présentation du résultat. En général, l'utilisateur a seulement besoin de connaître son nom en tout cas on peut toujours s'informer avec les commentaires des programmes. Dans les exemples qui suivent nous avons en général :

```
RN:= Fraction Integer
CL:= DynamicConstructibleClosure RN
CC:= DynamicConstructibleCase RN
```

- **Partie II.** Dans cette partie on définit une fonction qui décrit le calcul spécifique à effectuer dans la clôture : nous devons commencer par introduire les paramètres que ce calcul va utiliser, et dans un deuxième pas nous spécifions les opérations sur ces paramètres.
- **Partie III.** Ici on fait appel à la fonction (que nous avons construite dans la partie II) d'une façon un peu spéciale, due à l'utilisation de `allCases`. Telle qu'elle est programmée actuellement, `allCases` a besoin d'un (seul) argument en entrée.

5.1 Calculs avec des paramètres

Nous montrons ici des exemples de nature très différente sur les calculs avec des paramètres.

Exemple 1 *Résolution d'un système linéaire*
Voici une des utilisations élémentaires du programme de la clôture.

Nous voulons résoudre un système linéaire où la matrice des coefficients *dépend d'un paramètre*. Pour la résolution des systèmes linéaires nous utilisons la fonction `solve` d'Axiom implantée à cette fin ; pour la gestion du paramètre nous utilisons la clôture constructible dynamique.

Prenons par exemple le système :

$$\begin{pmatrix} a & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Pour cela nous écrivons ce fichier (en suivant le guide d'utilisation) :

```
RN:= Fraction Integer
CL:= DCCLOS RN
CC:= DCCASE RN

M:= Matrix CL
V:= Vector CL
P:= LinearSystemMatrixPackage(CL,V,V,M)

PS:=  Union(V, "failed")
Sol:= Record(particular: PS, basis: List V)
```

```
dynReso(n:NNI):Sol ==
  a:CL:= newElement('a)
  m:M:= [[a,1],[1,1]]
  v:V:= [2,1]
  solve(m,v)$P

allCases(dynReso,0)$DCTRL(CC,NNI,Sol)
```

Puisque la clôture constructible dynamique est un corps, nous pouvons construire des matrices et des vecteurs avec des coefficients dans `CL`.

`P:= LinearSystemMatrixPackage(CL,V,V,M)` indique à Axiom où est la fonction `solve` que l'on souhaite utiliser, et `PS` et `Sol` spécifient le type du résultat de `solve`.

Dans le pas suivant, nous construisons notre petite fonction : nous commençons par l'introduction du paramètre, la construction de la matrice des coefficients et le vecteur du second membre, et nous appelons à la fonction `solve` d'Axiom avec les données ainsi construites.

Finalement nous faisons appel à cette nouvelle fonction avec `allCases`. Noter qu'ici l'argument de la fonction `dynReso` n'est pas utilisé. Cela n'a pas grande importance. Voici le résultat :

```
[value is [particular= [- 1,2],basis= [[0,0]]] in case a = 0,
 value is [particular= "failed",basis= [[- 1,1]]] in case a = 1,
                             1      a - 2
 value is [particular= [-------,-------],basis= [[0,0]]] in case
                        (a - 1) (a - 1)
   a /= 0 , a /= 1]

Time: 1.10 sec
```

Cela signifie que :

- Si $a \neq 0$ et $a \neq 1$, le système a une seule solution, le vecteur :

$$\begin{pmatrix} \frac{1}{a-1} \\ \\ \frac{a-2}{a-1} \end{pmatrix}$$

C'est le cas général, mais notre système a étudié aussi les cas particuliers où $a = 0$ ou $a = 1$.

- Si $a = 1$ il n'y a pas de solution (puisque `particular = "failed"`).
- Si $a = 0$ le système a une seule solution qui est $(-1, 2)^t$.

Nous remarquons que la fonction `solve` d'Axiom (que nous n'avons pas modifiée) a été implantée sans aucune notion de "paramètre".

Exemple 2 *Base de Gröbner*

Nous montrons ici le calcul d'une base de Gröbner avec des paramètres dans les coefficients. `DMP` est l'abréviation (au sens Axiom) de DistributedMultivariatePolynomial.

```
Pol:= DMP([x,y], CL)

base(n:NNI):List Pol ==
  a:CL:= newElement('a)
  p1:Pol:= x**3 - a*x*y
  p2:Pol:= x**2*y - 2*y**2 + x
  groebner [p1,p2]
```

Voici le résultat :

```
                 5      2  6
[value is [x + 4y  - 2y ,y ] in case a = 0,
                5      2  6    3
 value is [x + y  - 2y ,y  - y ] in case a = 1,
                 2  3
 value is [x - 2y ,y ] in case a = 2,
                 5      2  6    3
 value is [x + 4y  - 2y ,y  - y ] in case a = 4,
                5      2  6      3           2
 value is [x - y  - 2y ,y  + a y ] in case a  - 4a + 5 = 0,
                 2            5     2  6      a      3
 value is [x + (a  - 4a + 4)y  - 2y ,y  - -------- y ] in case
                                                 2
                                          (a - 2)
                                        2
    a /= 0 , a /= 1 , a /= 2 , a /= 4 , a  - 4a + 5 /= 0]

Time: 17.00 sec
```

Exemple 3 *Ecriture d'un polynôme*

Nous avons trouvé cet exemple très intéressant puisqu'il montre des différences fondamentales entre un système de calcul “habituel” et un système de calcul adapté à l'évaluation dynamique.

Dans les exemples précédents nous avons vu l'utilisation des fonctions standard d'Axiom avec la clôture constructible.

Mais cela ne marche pas toujours “très bien” ...Nous nous intéressons ici à l'écriture d'un polynôme avec des paramètres pour coefficients. Nous avons considéré pour cela le domaine de polynômes standard d'Axiom :

```
UPX:= UnivariatePolynomial('x,CL)
x:UPX:= monomial(1,1)

ecrit1(n:NNI):UPX ==
  a:CL:= newElement('a)
  b:CL:= newElement('b)
  c:CL:= newElement('c)
  a*x**2 + b*x + c

allCases(ecrit1,0)$DCTRL(CC,NNI,UPX)
```

Voici le résultat :

```
[value is 0 in case c = 0 and b = 0 and a = 0,
 value is c in case c /= 0 and b = 0 and a = 0,
 value is x in case c = 0 and b = 1 and a = 0,
 value is x + c in case c /= 0 and b = 1 and a = 0,
 value is b x in case c = 0 and b /= 0 , b /= 1 and a = 0,
 value is b x + c in case c /= 0 and b /= 0 , b /= 1 and a = 0,
           2
 value is x  in case c = 0 and b = 0 and a = 1,
           2
 value is x  + c in case c /= 0 and b = 0 and a = 1,
           2
 value is x  + x in case c = 0 and b = 1 and a = 1,
           2
 value is x  + x + c in case c /= 0 and b = 1 and a = 1,
           2
 value is x  + b x in case c = 0 and b /= 0 , b /= 1 and a = 1,
           2
 value is x  + b x + c in case c /= 0 and b /= 0 , b /= 1 and a = 1,
             2
 value is a x  in case c = 0 and b = 0 and a /= 0 , a /= 1,
             2
 value is a x  + c in case c /= 0 and b = 0 and a /= 0 , a /= 1,
             2
 value is a x  + x in case c = 0 and b = 1 and a /= 0 , a /= 1,
             2
 value is a x  + x + c in case c /= 0 and b = 1 and a /= 0 , a /= 1,
             2
 value is a x  + b x in case
   c = 0 and b /= 0 , b /= 1 and a /= 0 , a /= 1
```

```
            2
 value is a x  + b x + c in case
   c /= 0 and b /= 0 , b /= 1 and a /= 0 , a /= 1]

Time: 8.81 sec
```

Maintenant nous pouvons comparer avec le domaine `DynamicUnivariatePolynomial` écrit par D. Duval :

```
DUPY:= DynamicUnivariatePolynomial(CL)
y:DUPY:= monomial(1,1)$DUPY

ecrit2(n:NNI):DUPY ==
  a:CL:= newElement('a)
  b:CL:= newElement('b)
  c:CL:= newElement('c)
  a*y**2 + b*y + c

allCases(ecrit2,0)$DCTRL(CC,NNI,DUPY)
```

Voici le résultat (où le symbole ? est la représentation externe de `y`) :

```
            2
[value is a ?  + b ? + c in case any c and any b and any a]

Time: 0.38 sec
```

Il est clair que la programmation en Axiom (comme dans d'autres systèmes) ne prête pas beaucoup d'attention aux tests d'égalité, puisqu'en général il ne sont pas coûteux. Mais maintenant nous voulons utiliser ces fonctions dans un nouveau contexte, où les test d'égalité provoquent des scindages et sont plus chers que d'habitude.

Cela signifie que, au moment d'utiliser les fonctions d'axiom depuis la clôture, l'efficacité depend de l'implantation de ces fonctions, puisque nous ne pouvons pas éviter des scindages provoqués par des tests qui ne sont pas significatifs.

Exemple 4 *Résoudre* $ax^2 + bx + c$

Un autre exemple dans le même contexte que l'exemple précédent est l'utilisation de la fonction `solve` d'Axiom pour résoudre l'équation $ax^2 + bx + c = 0$, où a, b, c sont des paramètres : nous ne reproduisons pas le résultat.

Mais signalons que cet exemple nous a permis de détecter un "problème" dans Axiom : comme nous l'avons vu dans la résolution d'un système linéaire, Axiom répond `"failed"` quand il n'y a pas de solution. Cela n'est pas arrivé dans ce cas, nous n'avons pas obtenu `"failed"` comme nous l'attendions.

Voici la réponse donnée par la version actuelle d'Axiom quand il n'y a pas de solution :

```
solve(1)

   >> System error:
   Cannot take first of an empty list
   You are being returned to the top level of the interpreter.
```

Pour éviter ce problème, nous avons imposé `areDifferent(a*b,0)`. Voici la fonction :

```
Pol:= Polynomial(CL)
x:Pol:= monomial(1::CL,'x,1::NNI)
SSP:= SystemSolvePackage CL
Sol:= List Equation Fraction Pol

monReso(n:NNI):Sol ==
  a:CL:= newElement('a)
  b:CL:= newElement('b)
  c:CL:= newElement('c)
  if areDifferent(a*b,0) then
    solve(a*x**2 + b*x + c)$SSP
  else []
```

La réponse est formée de 51 cas différents et a pris `714.19 sec`. Dans cet exemple les scindages de l'écriture du résultat s'ajoutent aux scindages propres à la fonction `solve`.

Voici le même exemple traité dans la clôture constructible :

```
dynReso(n:NNI):Boolean ==
  a:CL:= newElement('a)
  b:CL:= newElement('b)
  c:CL:= newElement('c)
  y:CL:= newElement('y)
  areEqual(a*y**2 + b*y + c,0)
```

et le résultat :

```
[value is true in case any y and c = 0 and b = 0 and a = 0,
                             1
 value is true in case y = - --- c and any c and b /= 0 and a = 0,
                             (b)
                             1/2           1/4  2
 value is true in case y = - --- b and c = --- b  and any b and
                             (a)           (a)
   a /= 0,
```

```
 value is true in case
     2    1         1                   1/4  2
    y  + --- b y + --- c = 0 and c /= --- b  and any b and a /= 0]
         (a)       (a)                 (a)

Time: 15.07 sec
```

Dans cet exemple, la fonction `solve` d'Axiom est remplacée par l'imposition d'une contrainte sur les paramètres introduits : nous sommes intéressés seulement aux valeurs des paramètres qui vérifient cette contrainte. Le résultat est composé :

- d'un booléen qui nous dit que la contrainte se vérifie, c'est-à-dire que l'équation a une solution et
- d'une description des valeurs des paramètres qui vérifient la contrainte, c'est-à-dire, d'une description de la solution.

Donc, la solution de $ay^2 + by + c = 0$ est :

$$\begin{cases} a = 0 \\ b = 0 \\ c = 0 \\ y \text{ quelconque} \end{cases} \quad \begin{cases} a = 0 \\ b \neq 0 \\ c \text{ quelconque} \\ y = \frac{-c}{b} \end{cases} \quad \begin{cases} a \neq 0 \\ b \text{ quelconque} \\ c = \frac{(1/4)b^2}{a} \\ y = \frac{-(1/2)b}{a} \end{cases} \quad \begin{cases} a \neq 0 \\ b \text{ quelconque} \\ c \neq \frac{(1/4)b^2}{a} \\ y^2 + \frac{b}{a}y + \frac{c}{a} = 0 \end{cases}$$

Nous voyons la résolution des systèmes plus en détail ci-dessous.

Exemple 5 *Une autre équation*
Nous montrons ici un autre exemple de résolution d'une équation, mais ici il y a des *dénominateurs* ...

$$\frac{y+2a}{2a-y} + \frac{y-2a}{2a+y} = \frac{4a^2}{4a^2-y^2}$$

Cette équation se traduit dans la clôture constructible comme suit :

```
dynReso(n:NNI):Boolean ==
  a:CL:= newElement('a)
  y:CL:= newElement('y)
  areDifferent(2*a-y,0) and
  areDifferent(2*a+y,0) and areDifferent(4*a**2-y**2,0) and
  areEqual((y+2*a)/(2*a-y) + (y-2*a)/(2*a+y),(4*a**2)/(4*a**2-y**2))
```

et son résultat :

```
[value is true in case y /= 0 and a = 0,
                              1
 value is true in case y = - a and a /= 0]
                              2

Time: 11.76 sec
```

Nous avons pris cet exemple dans [Pa] (p.96 ex.4), où la solution décrite est :

- Si $a \neq 0$ alors $y = \frac{a}{2}$ et l'ensemble de solutions est $S = \{\frac{a}{2}\}$.
- Si $a = 0$ alors tous les nombres réels autres que 0 sont solutions de cette équation.

Exemple 6 *Matrice de passage entre A et A^t*

A partir du travail exposé dans la partie I de ce mémoire, nous nous sommes intéressés à la matrice de passage entre une matrice donnée et sa transposée. Nous nous sommes demandés s'il existe une matrice P non-singulière telle que pour toute matrice A (peut-être avec certaines conditions) on a $AP = PA^t$.

A notre surprise, la réponse est négative : la matrice P dépend de la matrice donnée A, bien que l'opération de transposition soit indépendante des données. On peut trouver la réponse complète à cette question dans [Kap] ou [TZ].

Nous avons fait un petit programme dans la clôture constructible avec des matrices 2×2 ayant des paramètres dans les coefficients. Pour simplifier un peu le problème, nous avons choisi P symétrique.

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad P = \begin{pmatrix} x & y \\ y & z \end{pmatrix}$$

$$AP = PA^t$$

Voici le programme :

```
passage(n:NNI):Boolean ==
  a:CL:= newElement('a)
  b:CL:= newElement('b)
  c:CL:= newElement('c)
  d:CL:= newElement('d)
  x:CL:= newElement('x)
  y:CL:= newElement('y)
  z:CL:= newElement('z)
  de:CL:= x*z - y**2
  e1:CL:= a*y + b*z
  e2:CL:= c*x + d*y
  areDifferent(de,0) and areEqual(e1,e2)
```

Pour le résultat nous montrons seulement la partie intéressante, c'est-à-dire celle qui correspond à `true` :

```
      1       a
z = (--- d - ---)y and y /= 0 and x = 0 and any d and
     (b)     (b)
    any c and b /= 0 and any a

         2
        c                    c
z /= -------- x and y = - ------- x and x /= 0 and d /= a and
            2             (d - a)
     (d - a)
    any c and b = 0 and any a

z = 0 and y /= 0 and x /= 0 and d = a and c = 0 and b /= 0 and any a

      1       a         2       1       a
z = (--- d - ---)y and y  + (- --- d + ---)x y /= 0 and x /= 0 and
     (b)     (b)               (b)     (b)
    d /= a and c = 0 and b /= 0 and any a

      1       a       1                    1       a
z = (--- d - ---)y + --- c x and y /= -  (--- d - ---)x and x /= 0
     (b)     (b)     (b)              2   (b)     (b)
         2                  2
    and d  - 2a d + 4b c + a  = 0 and c /= 0 and b /= 0 and any a

      1       a       1
z = (--- d - ---)y + --- c x and
     (b)     (b)     (b)
    2       1       a         1     2
   y  + (- --- d + ---)x y - --- c x  /= 0 and x /= 0 and
           (b)     (b)       (b)
    2                  2
   d  - 2a d + 4b c + a  /= 0 and c /= 0 and b /= 0 and any a

Time: 1094.68 sec
```

Exemple 7 *Matrice de Jordan*

Pour finir cette petite liste d'exemples de calculs avec des paramètres, nous voulons calculer la forme de Jordan d'une matrice avec des coefficients dans la clôture constructible. Nous utilisons pour cela le programme du calcul de la forme de Jordan que nous avons implanté [Go]. Donc nous utilisons ici les clôtures algébrique et constructible à la fois. Nous avons considéré la matrice :

$$A = \begin{pmatrix} 0 & 0 & a^3 \\ 1 & 0 & -3a^2 \\ 0 & 1 & 3a \end{pmatrix}$$

et nous avons obtenu :

```
[value is
  [value is
              +0  0  1+              +0  1  0+          +0  0  1+
              |      |               |      |           |      |
   [transition= |0  1  0|,jordanBlock= |0  0  1|,inverse= |0  1  0|]
              |      |               |      |           |      |
              +1  0  0+              +0  0  0+          +1  0  0+
   in case ev = 0]
 in case a = 0,

 value is
  [value is
               + 1     1     +
               | -    - - a  0|
               | 9     3     |
               |             |              +a  1  0+
               |2            |              |       |
    [transition= |- a    1    0|, jordanBlock= |0  a  1|,
               |3            |              |       |
               |             |              +0  0  a+
               |  1          |
               |- -   2a    1|
               +  3          +
            + 27     9a   0+
            |             |
     inverse= |- 18a   3    0|] in case ev = a]
            + - 3   - 3a  1+
         2   1
 in case a  + - = 0,
             3

 value is
  [value is
              +  4      3     +
              | a      a     0|              +a  1  0+
              |              |              |       |
   [transition= |    3      2   |, jordanBlock= |0  a  1|,
              |- 2a   - 3a   0|              |       |
              |              |              +0  0  a+
              |  2           |
              + a      2a    1+
```

```
             + 3     1     +
             | --    --   0|
             |  4     3    |
             | a     a     |
             |             |
             |  2     1    |
    inverse= |- --   - --  0|] in case ev = a]
             |   3     2   |
             |  a     a    |
             |             |
             | 1     1     |
             | --    ---  1|
             |  2   (a)    |
             + a           +
                   2   1
 in case a /= 0 , a  + - /= 0]
                       3

Time: 21.40 sec
```

C'est-à-dire, $P^{-1}AP = J$ où :

- Si $a = 0$ alors

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad J = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad P^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

- Si $a^2 + \frac{1}{3} = 0$ alors

$$P = \begin{pmatrix} \frac{1}{9} & \frac{-a}{3} & 0 \\ \frac{2a}{3} & 1 & 0 \\ \frac{-1}{3} & 2a & 1 \end{pmatrix} \qquad J = \begin{pmatrix} a & 1 & 0 \\ 0 & a & 1 \\ 0 & 0 & a \end{pmatrix} \qquad P^{-1} = \begin{pmatrix} 27 & 9a & 0 \\ -18a & 3 & 0 \\ -3 & -3a & 1 \end{pmatrix}$$

- Si $a \neq 0$ et $a^2 + \frac{1}{3} \neq 0$ alors

$$P = \begin{pmatrix} a^4 & a^3 & 0 \\ -2a^3 & -3a^2 & 0 \\ a^2 & 2a & 1 \end{pmatrix} \qquad J = \begin{pmatrix} a & 1 & 0 \\ 0 & a & 1 \\ 0 & 0 & a \end{pmatrix} \qquad P^{-1} = \begin{pmatrix} \frac{3}{a^4} & \frac{1}{a^3} & 0 \\ \frac{-2}{a^3} & \frac{-1}{a^2} & 0 \\ \frac{1}{a^2} & \frac{1}{a} & 1 \end{pmatrix}$$

5.2 Systèmes d'équations polynomiales. Systèmes constructibles

Il existe plusieurs méthodes pour la "résolution" d'un système d'équations polynomiales dans le sens de "triangulariser", par exemple la méthode de bases de Gröbner [Bu], et les algorithmes de Seidenberg [Se], Ritt-Wu [Ch], Lazard [La] ou Kalkbrener [Kal].

L'évaluation dynamique propose une nouvelle méthode : un système d'équations polynomiales (S) à coefficients dans un corps $\mathbf{K}$ peut être "triangularisé" par des calculs dans la clôture constructible de $\mathbf{K}$. Il faut d'abord considérer chaque indéterminée apparaissant dans le système comme un paramètre, qu'on introduit grâce à `newElement` (ce qui oblige à ordonner les indéterminées), et chaque équation comme un appel de `areEqual` (ce qui oblige à ordonner les équations).

Exemple 1 Reprenons l'exemple 1 du paragraphe précédent. Nous avons utilisé la fonction `solve` d'Axiom pour résoudre le système linéaire :

$$\begin{pmatrix} a & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Mais il peut être résolu en utilisant des opérations de la clôture constructible :

```
dynamicReso(n:NNI):Boolean ==
  a:CL:= newElement('a)
  x:CL:= newElement('x)
  y:CL:= newElement('y)
  areEqual(a*x + y, 2) and areEqual(x + y, 1)
```

dont le résultat est :

```
[value is false in case y = - x + 2 and any x and a = 1,
                                a - 2             1
 value is true in case y = ------- and x = ------- and a /= 1]
                               (a - 1)        (a - 1)

Time: 1.71 sec
```

Exemple 2 Nous décrivons en détail cette méthode en suivant cet exemple :

$$\begin{cases} -2a + x + y = 0 \\ yz - ay - ax + a^2 = 0 \\ z^2 - y^2 - x^2 + 2ax + 2az - a^2 = 0 \end{cases}$$

Il est traduit dans la clôture constructible par :

```
a:= newElement('a)
x:= newElement('x)
y:= newElement('y)
z:= newElement('z)
areEqual(-2*a + x + y,0) and
areEqual(y*z - a*y - a*x + a**2,0) and
areEqual(z**2 - y**2 - x**2 + 2*a*x + 2*a*z - a**2,0)
```

Au fur et à mesure de l'introduction des équations, le système met automatiquement à jour les contraintes sur les paramètres, afin qu'elles se présentent toujours sous la forme récursive détaillée dans la section 3. Ainsi on obtient une union de systèmes *triangulaires constructibles* $T_1, \ldots, T_m$.

Si on note $a_1, a_2, \ldots, a_n$ les paramètres dans l'ordre où ils ont été introduits, un système triangulaire constructible est de la forme :

$$(T) = \begin{cases} exp_1(a_1) \\ exp_2(a_2) \\ \ldots \\ exp_n(a_n) \end{cases}$$

où $exp_i(a_i)$ a trois possibilités, selon que a_i est de type `anyElement`, `exception` ou `algebraic` :

- $any\ a_i$
- $P_{i,1}(a_i) \neq 0, \ldots, P_{i,k_i}(a_i) \neq 0$
- $P(a_i) = 0$

avec $P_{i,1}(a_i), \ldots, P_{i,k_i}(a_i), P(a_i) \in \mathbf{K}(a_1, \ldots, a_{i-1})[a_i]$, unitaires de degré strictement positif en a_i, qu'on peut supposer sans facteur carré lorsque la caractéristique de $\mathbf{K}$ est 0. De plus les polynômes $P_{i,1}(a_i), \ldots, P_{i,k_i}(a_i)$ n'ont de pas de facteur commun. S'il y a des coefficients dans ces polynômes avec des dénominateurs non triviaux, ces dénominateurs ne font intervenir que les paramètres a_j de type `exception` (avec $j < i$).

Les *solutions* d'un système triangulaire constructible constituent l'ensemble $V(T) = V(a_1) \times \ldots \times V(a_n)$ où $V(a_i)$ est l'ensemble de valeurs possibles pour a_i (comme défini dans la section 3.1) pour chaque i de 1 à n.

L'ensemble des solutions du système (S) est la réunion des ensembles $V(T_j)$, pour $j = 1, \ldots, m$. En particulier on n'introduit pas de *solutions parasites* (ou "spurious solutions"). D'autre part il n'y a pas de redondance, au sens où il n'y a aucune solution commune à deux des T_j pour $1 \leq j \leq m$.

Il est à noter que les contraintes de type `exception` apparaissent lors du calcul, puisque aucun `areDifferent` n'est utilisé pour poser le problème.

Dans cet exemple le résultat est le suivant :

```
[value is true in case z = 0 and y = 0 and x = 0 and a = 0,
 value is false in case z = 0 and y = - x and x /= 0 and a = 0,
```

```
 value is true in case
        2   3    10   2
   z = - -- x  + --- x  - 17x + 8a and y = - x + 2a and
         2      (a)
        a
    4       3   37  2 2      3    15  4
   x  - 7a x  + -- a x  - 21a x + -- a  = 0 and a /= 0]
                2                 2

Time: 247.10 sec
```

Cela veut dire que l'ensemble de solutions du système initial est la réunion de $V(T_1)$ et de $V(T_2)$ où :

$$(T_1) = \begin{cases} a = 0 \\ x = 0 \\ y = 0 \\ z = 0 \end{cases} \quad \text{et} \quad (T_2) = \begin{cases} a \neq 0 \\ x^4 - 7ax^3 + \frac{37}{2}a^2x^2 - 21a^3x + \frac{15}{2}a^4 = 0 \\ y = -x + 2a \\ z = -\frac{2}{a^2}x^3 + \frac{10}{a}x^2 - 17x + 8a \end{cases}$$

Notons que l'ordre d'introduction des équations est aussi important que l'ordre d'introduction des indéterminées. Dans cet exemple, nous n'avons pas réussi à obtenir la solution après plusieurs heures de calcul en introduisant les équations dans l'ordre inverse. Nous proposons l'introduction des équations en commençant par les plus simples.

Systèmes constructibles

Nous avons montré ainsi comment résoudre un système d'équations polynomiales, mais nous pouvons considérer une classe de systèmes plus générale que nous appelons les *systèmes constructibles.*

Un système constructible est un système :

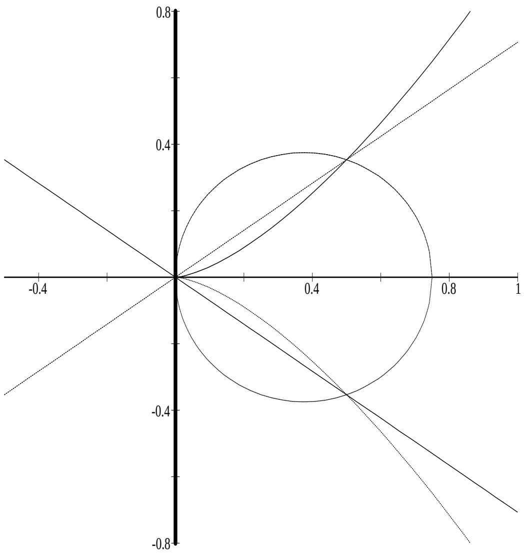
$$(S_c) = \begin{cases} P_1(x_1, \ldots, x_n) \ \varsigma_1 \ 0 \\ P_2(x_1, \ldots, x_n) \ \varsigma_2 \ 0 \\ \quad \vdots \\ P_s(x_1, \ldots, x_n) \ \varsigma_s \ 0 \end{cases}$$

où chaque ς_i est $=$ ou $\neq$.

De même que dans les systèmes polynomiaux, chaque indeterminée est introduite par `newElement` et les "équations" sont introduites par `areEqual` ou `areDifferent` selon que ς_i est $=$ ou $\neq$. L'ensemble des solutions de (S_c) est toujours exprimé comme la réunion des ensembles de solutions d'un nombre fini de systèmes triangulaires constructibles. Voyons un exemple.

$$\begin{cases} x \neq 0 \\ y^2 - x^3 = 0 \\ 2y^2 - x^2 = 0 \\ (8x-3)^2 + (8y)^2 = 9 \end{cases}$$

Cet système correspond à la figure suivante (où l'axe en gras est interdit) :



Sa traduction dans la clôture constructible est :

```
x:CL:= newElement('x)
y:CL:= newElement('y)
areDifferent(x,0)
areEqual(y**2-x**3,0) and
areEqual(2*y**2-x**2,0) and
areEqual((8*x-3)**2 + (8*y)**2, 9)
```

et le résultat obtenu est :

```
                          2   1              1
[ value is true in case y   - - = 0 and x = - ]
                              8              2
Time: 1.61 sec
```

Un autre exemple de système constructible a été traité dans l'exemple 5 du paragraphe précédent. En effet, une équation où figure des dénominateurs est équivalente à un système constructible, lorsqu'on exprime les dénominateurs avec `areDifferent`.

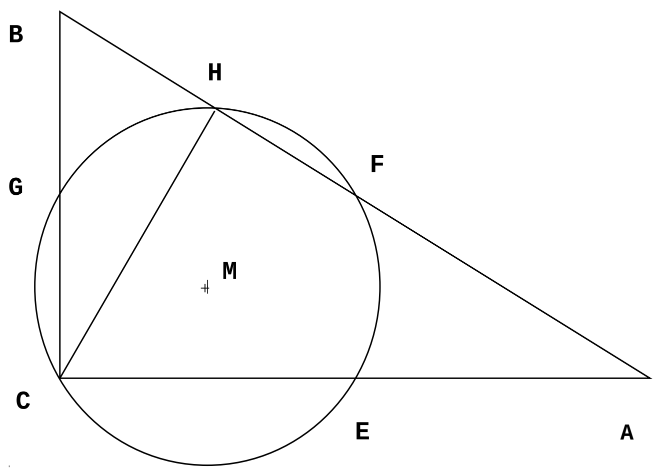
On peut rapprocher cette méthode d'une version "complexe" de la décomposition algébrique cylindrique [Co] ou de la méthode des ensembles caractéristiques de Ritt ou de Wu [Ch, BCK]. De même, elle peut être considérée comme une variante de la méthode algébrique de Seidenberg [Se]. Elle est par contre assez différente des méthodes utilisant les bases de Gröbner [Bu].

5.3 Démonstration automatique en géométrie élémentaire

Il existe plusieurs méthodes pour étudier ce type de questions [BCK]. En général, ces méthodes peuvent être divisées en deux familles, selon qu'elles utilisent les bases de Gröbner ou les ensembles caractéristiques. Mais l'utilisation de ces méthodes révèle quelques problèmes [CY].

L'évaluation dynamique propose encore une autre solution pour la démonstration automatique de théorèmes en géométrie, et nous croyons que nous sommes capables d'éviter la plupart des problèmes que les autres méthodes présentent. Cependant cette affirmation nécessiterait d'une étude plus approfondie. Nous nous limitons ici à l'étude détaillée de deux exemples.

Exemple 1 Considérons le théorème suivant : Etant donné un triangle ABC, rectangle en C, le cercle Γ passant par les milieux E, F et G respectivement des côtés CA, AB et BC (c'est le *cercle d'Euler* du triangle ABC), et le pied H de la hauteur issue de C, montrer que H est sur Γ.



Nous voulons traduire ce problème en un calcul dans la clôture. Pour cela on commence par appeler M le centre de Γ, par choisir un repère orthonormé du plan adapté au problème, et par donner un nom aux coordonnées de chacun des points. Par exemple :

$$\begin{array}{lll} C=(0,0), & A=(a_1,0), & B=(0,a_2), \\ E=(a_3,0), & F=(a_4,a_5), & G=(0,a_6), \\ M=(a_7,a_8), & H=(a_9,a_{10}). & \end{array}$$

Nous avons donc besoin de 10 paramètres dans la clôture : $a_1, a_2, \ldots, a_{10}$. Pour dire que le triangle ABC est un “vrai” triangle nous précisons :

$$a_1 \neq 0\,,\ a_2 \neq 0$$

Le fait que ce triangle soit rectangle est déjà traduit dans le choix des coordonnées. Ensuite exprimons que E, F et G sont les milieux respectifs de CA, AB et BC :

$$a_3 = \frac{1}{2}a_1\,,\ a_4 = \frac{1}{2}a_1\,,\ a_5 = \frac{1}{2}a_2\,,\ a_6 = \frac{1}{2}a_2$$

Le centre M du cercle Γ est caractérisé par le fait que EM et FM d'une part, EM et GM d'autre part, sont de la même longueur, et donc :

$$(a_7 - a_3)^2 + a_8^2 = (a_7 - a_4)^2 + (a_8 - a_5)^2\,,\ (a_7 - a_3)^2 + a_8^2 = a_7^2 + (a_8 - a_6)^2$$

Le point H peut être caractérisé comme le point de la droite AB tel que AB soit perpendiculaire à CH. Or l'équation de AB est :

$$\frac{1}{a_1}x + \frac{1}{a_2}y = 1$$

(rappelons que a_1 et a_2 sont non nuls) d'où :

$$\frac{1}{a_1}a_9 + \frac{1}{a_2}a_{10} = 1\,,\ -a_1a_9 + a_2a_{10} = 0$$

Nous pouvons maintenant traduire la conclusion du théorème, qui dit que EM et HM sont de même longueur, c'est-à-dire :

$$(a_7 - a_3)^2 + a_8^2 = (a_7 - a_9)^2 + (a_8 - a_{10})^2$$

Voici le programme correspondant :

```
geo(x:NNI):Union(Boolean,"failed") ==
  a1:CL:= newElement('a1)
  ...
  a10:CL:= newElement('a10)
  e1:CL:= (a7-a3)**2 + a8**2 - (a7-a4)**2 - (a8-a5)**2
  e2:CL:= (a7-a3)**2 + a8**2 - a7**2 - (a8-a6)**2
  e3:CL:= - a1*a9 + a2*a10
  e4:CL:=(a7-a3)**2 + a8**2 - (a7-a9)**2 - (a8-a10)**2
  if areDifferent(a1,0) and
     areDifferent(a2,0) and
     areEqual(a3,(1/2)*a1) and
     areEqual(a4,(1/2)*a1) and
     areEqual(a5,(1/2)*a2) and
```

```
        areEqual(a6,(1/2)*a2) and
        areEqual(e1,0) and
        areEqual(e2,0) and
        areEqual((1/a1)*a9 +(1/a2)*a10 - 1,0) and
        areEqual(e3,0)
     then (e4 = 0)$CL
     else "failed"
```

et le résultat :

```
[value is "failed" in case
           1                                   1
   a10 = - ---- a2 a9 + a2 and any a9 and a8 = - a2 and
          (a1)                                 4
       1             1             1             1
   a7 = - a1 and a6 = - a2 and a5 = - a2 and a4 = - a1 and
       4             2             2             2
       1         2     2
   a3 = - a1 and a2  + a1  = 0 and a1 /= 0,
       2
 value is true in case
             2                       2
           a1 a2                   a1 a2               1
   a10 = ----------- and a9 = ----------- and a8 = - a2 and
           2     2                 2     2             4
        (a2  + a1 )             (a2  + a1 )
       1             1             1             1
   a7 = - a1 and a6 = - a2 and a5 = - a2 and a4 = - a1 and
       4             2             2             2
       1                   2     2
   a3 = - a1 and a2 /= 0, a2  + a1  /= 0  and a1 /= 0]
       2

Time: 220.72 sec
```

Ce résultat est composé de deux branches qui apparaissent précisément quand le système doit imposer `areEqual(e3,0)`. A ce moment apparaît la question “$a_2^2+a_1^2 = 0$?” qui a deux réponses possibles :

- si la réponse est `true`, le système actualise les contraintes pour a_2 et a_1, ce qui donne $a_2^2 + a_1^2 = 0$ et $a_1 \neq 0$; imposer `areEqual(e3,0)` revient alors à imposer `areEqual(a1*a2**2,0)` ce qui est incompatible avec ces contraintes. Donc le système retourne `"failed"`. Cela signifie que dans ce cas on ne peut pas construire la figure. Mais ce cas ne peut pas se produire si a_1 et a_2 sont réels.

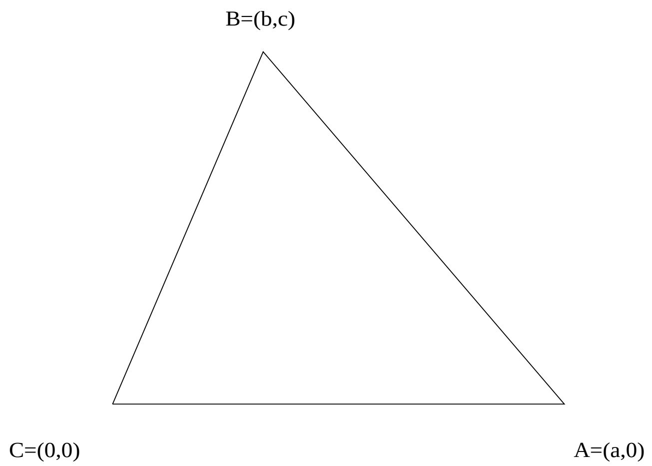
- si la réponse est `false`, le système actualise les contraintes pour a_2 et a_1, ce qui donne $a_2 \neq 0$, $a_2^2 + a_1^2 \neq 0$ et $a_1 \neq 0$; imposer `areEqual(e3,0)` est maintenant équivalent à imposer la valeur $\frac{a_1 a_2^2}{a_2^2 + a_1^2}$ pour a_9, ce qui est compatible avec la contrainte `anyElement` pour a_9. Ainsi la figure est construite. Maintenant il reste à repondre à la question "`e4 = 0 ?`", ce qui retourne `true`. Cela signifie que le théorème est toujours vrai.

Exemple 2 Nous étudions ici un exemple pris dans l'article [CY], et qui nous a été transmis par T. Recio. Dans cet article il est montré que certaines méthodes de démonstration automatique "prouvent" que *tout triangle est isocèle.*

Nous étudions donc le problème suivant :

Soit T un triangle. Est-il isocèle ?

Pour étudier cette question nous devons exprimer cet énoncé géométrique de façon algébrique. Pour cela nous choisissons un repère orthonormé du plan adapté au problème, et nous donnons un nom aux coordonnées de chacun des points. Nous considérons donc le triangle ABC ci-dessous et nous nous demandons par exemple si $CA = AB$.



Maintenant nous devons étudier l'égalité

$$a^2 = (b - a)^2 + c^2$$

ce que nous traduisons dans la clôture constructible comme suit :

```
a:CL:= newElement('a)
b:CL:= newElement('b)
c:CL:= newElement('c)
(a**2 = (b - a)**2 + c**2)$CL
```

Nous obtenons comme réponse :

```
[value is true in case c = 0 and b = 0 and a = 0,
 value is false in case c /= 0 and b = 0 and a = 0,
                        2    2
 value is true in case c   + b   = 0 and b /= 0 and a = 0,
                         2    2
 value is false in case c   + b   /= 0 and b /= 0 and a = 0,
                                  2
 value is true in case c = 0 and b   - 2a b = 0 and a /= 0,
                                    2
 value is false in case c /= 0 and b   - 2a b = 0 and a /= 0,
                        2    2                  2
 value is true in case c   + b   - 2a b = 0 and b   - 2a b /= 0
   and a /= 0,
                         2    2                   2
 value is false in case c   + b   - 2a b /= 0 and b   - 2a b /= 0
   and a /= 0]
```

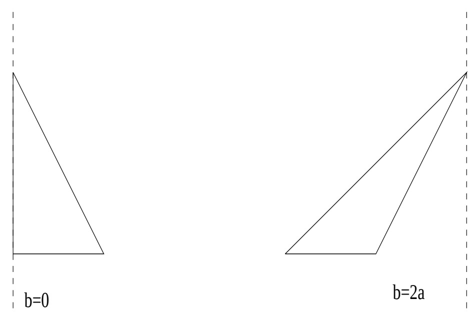
```
Time: 5.94 sec
```

Noter que $a^2 = (b-a)^2 + c^2$ est équivalent à $c^2 + b^2 - 2ab = 0$. Cela signifie que le théorème est *vrai* si :

- $a = 0$ et $b = 0$ et $c = 0$ (le triangle se réduit à un point)
- $a = 0$ et $b \neq 0$ et $c^2 + b^2 = 0$ (le triangle se réduit à un segment)
- $a \neq 0$ et $b^2 - 2ab = 0$ et $c = 0$ (le triangle se réduit à un segment)
- $a \neq 0$ et $b^2 - 2ab \neq 0$ et $c^2 + b^2 - 2ab = 0$ (c'est-à-dire pour deux valeurs précises de c en fonction de a et b)

et le théorème est *faux* si :

- $a = 0$ et $b = 0$ et $c \neq 0$ (le triangle se réduit à un segment)
- $a = 0$ et $b \neq 0$ et $c^2 + b^2 \neq 0$ (le triangle se réduit à un segment)
- $a \neq 0$ et $b^2 - 2ab = 0$ et $c \neq 0$



- $a \neq 0$ et $b^2 - 2ab \neq 0$ et $c^2 + b^2 - 2ab \neq 0$ (*cas général*)

Donc, nous pouvons déduire (comme prévu !) que *en général un triangle n'est pas isocèle.*

Nous observons dans le résultat obtenu qu'il y a plusieurs cas où le triangle se réduit à un point ou à un segment. Nous pouvons considérer que ce sont des *cas dégénérés* et peut-être voulons-nous considérer des triangles qui ne se réduisent pas à un point ou à un segment. La fonction `areDifferent` nous donne cette possibilité.

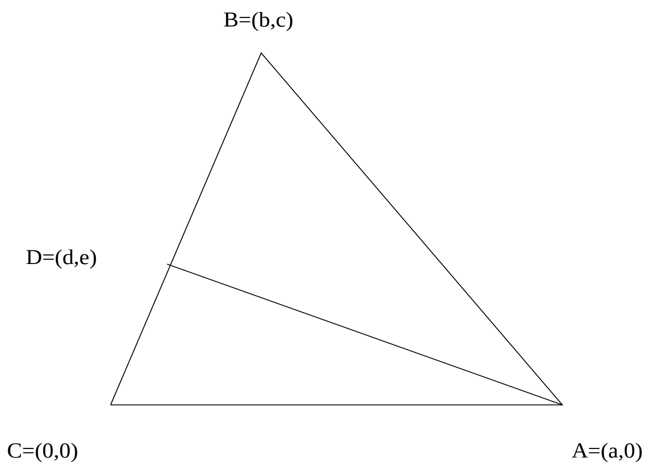
```
a:CL:= newElement('a)
b:CL:= newElement('b)
c:CL:= newElement('c)
if areDifferent(a,0) and
   areDifferent(c,0)
then (a**2 = (b-a)**2 + c**2)$CL
else "failed"
```

et nous obtenons :

```
value is false in case c /= 0 and b  - 2a b = 0 and a /= 0,
                             2    2                  2
 value is true in case c  + b  - 2a b = 0 and b  - 2a b /= 0
   and a /= 0,
 value is false in case
             2    2                  2
   c /= 0 , c  + b  - 2a b /= 0 and b  - 2a b /= 0 and a /= 0]

Time: 14.31 sec
```

Les méthodes pour la démonstration automatique qui utilisent les bases de Gröbner ou les ensembles caractéristiques travaillent dans l'idéal engendré par les hypothèses. Dans cet exemple [CY] ajoutent une hypothèse qui est *vraie en général* sur un triangle.



Soit $T = ABC$ un triangle et AD la hauteur issue de A. Montrer que $CA = AB$.

Donc les hypothèses s'expriment par :

$$\begin{array}{ll} cd - be = 0 & D \in BC \\ ce + bd = ab & AD \perp BC \end{array}$$

et la traduction dans la clôture constructible est :

```
a:CL:= newElement('a)
b:CL:= newElement('b)
c:CL:= newElement('c)
d:CL:= newElement('d)
e:CL:= newElement('e)
if areEqual(c*d - b*e,0) and
   areEqual(c*e + b*d, a*b)
then (a**2 = (b-a)**2 + c**2)$CL
else "failed"
```

et le résultat :

```
[value is true in case any e and any d and c = 0 and b = 0 and any a,
 value is false in case e = 0 and d = 0 and c /= 0 and b = 0
   and any a,
 value is "failed" in case
        1                      2    2
   e = --- c d and any d and c   + b   = 0 and b /= 0 and any a,
       (b)
                                                     2    2
 value is false in case e = 0 and d = 0 and c   + b   /= 0 and b /= 0
   and a = 0,
 value is true in case e = 0 and d = a and c = 0 and b = 2a
   and a /= 0,
 value is false in case
            2                   3
          2a c                4a            2     2
   e = ---------- and d = ---------- and c   + 4a   /= 0 , c /= 0
         2     2             2     2
       (c  + 4a )          (c  + 4a )
   and b = 2a and a /= 0,
```

```
 value is true in case
      1           1        2    2
   e = - c and d = - b and c   + b   - 2a b = 0 and b /= 0  , b /= 2a
      2           2
   and a /= 0,
 value is false in case
                                 2
        a b c               a b           2    2       2    2
   e = --------- and d = --------- and c   + b  /=0, c   + b - 2a b/= 0
        2    2              2    2
      (c   + b )         (c   + b )
   and b /= 0  , b /= 2a and a /= 0]

Time: 79.51 sec
```

Nous observons un cas où la réponse est `"failed"` ; cela signifie que les hypothèses sont *inconsistantes* dans ce cas et donc on peut en déduire tout énoncé. Sinon, le théorème est *vrai* si :

- a est quelconque, $b = 0$, $c = 0$, ... (le triangle se réduit à un segment)
- $a \neq 0$, $b = 2a$, $c = 0$, ... (le triangle se réduit à un segment)
- $a \neq 0$, $b \neq 2a$, $c^2 + b^2 - 2ab = 0$ (le triangle est isocèle et les coordonnées de D sont $(\frac{b}{2}, \frac{c}{2})$)

et le théorème est *faux* dans les autres cas, en particulier dans le cas général.

Méthode La méthode que nous proposons consiste donc en les pas suivants :

1. Introduire les indéterminées du problème avec `newElement`. En particulier, il ne faut pas diviser l'ensemble des indéterminées en *paramètres ou variables indépendantes* et *variables dépendantes* : seul l'ordre est important dans la clôture.

2. Introduire les hypothèses du théorème à démontrer avec `areEqual` et `areDifferent`. Remarquons qu'en particulier l'ensemble des hypothèses peut être vide. D'autre part l'utilisateur peut éviter des cas qu'il considère comme dégénérés, ou se restreindre à un certain type de figure.

3. Introduire avec un test d'égalité la conclusion à prouver. Notre système vérifiera automatiquement la consistance des hypothèses avant de prouver la conclusion.

Conclusion

Nous nous sommes intéressés ici aux calculs faisant intervenir des paramètres. En particulier nous avons étudié les paramètres que l'on peut appeler *constructibles*.

Pour effectuer ce type de calculs, nous utilisons l'évaluation dynamique et nous avons réalisé le programme de la clôture constructible dynamique.

C'est la première fois qu'une telle méthode de calcul est implantée de manière tout-à-fait systématique.

D'autre part, l'utilisation de ce programme est très simple, comme on peut le voir sur les exemples.

Nous avons montré des exemples d'utilisation de ce programme. Ils nous montrent l'importance et l'intérêt de ce type de calculs, et ils soulèvent des questions sur les fondements des systèmes de calcul formel.

Finalement, nous avons utilisé ce programme pour la résolution des systèmes d'équations polynomiales et pour la démonstration automatique de théorèmes en géométrie élémentaire. D'autres applications sont envisagées, comme par exemple la robotique, déjà étudiée sous cet angle (mais "à la main") dans [GR].

Références

[AK] S. A. Abramov, K. Yu. Kvashenko. – *On the Greatest Common Divisor of Polynomials which depend on a Parameter.* ISSAC'93 (1993).

[Bo] N. Bourbaki – Eléments de Mathématique. Algèbre Commutative. Ed. Hermann (1961).

[Bu] B. Buchberger. – *Gröbner Bases: an Algorithmic Method in Polynomial Ideal Theory.* Recent Trends in Multidimensional System Theory, Bose (ed.), (Reidel, 1985).

[BCK] B. Buchberger, G.E. Collins, B. Kutzler. – *Algebraic Methods for Geometry Reasoning.* Ann. Rev. Comput. Sci. 3, (1988).

[Ch] S. C. Chou – *Mechanical Geometry Theorem Proving* Reidel Publishing Company (1988).

[CY] S. C. Chou, J. G. Yang. – *On the Algebraic Formulation of Certain Geometry Statements and Mechanical Geometry Theorem Proving.* Algorithmica 4, p. 237–262 (1989).

[Co] M. Coste. – *Effective Semi-Algebraic Geometry.* Geometry and Robotics, Springer Lecture Notes in Computer Science 391, ed. J.-D. Boissonat, J.-P. Laumond, p. 1–27 (1989).

[DDD] J. Della Dora, C. Dicrescenzo, D. Duval. – *About a New Method for Computing in Algebraic Number Fields.* Eurocal'85, vol.2, Springer Lecture Notes in Computer Science 204, ed. G. Goos, J. Hartmanis, p. 289–290 (1985).

[DD] C. Dicrescenzo, D. Duval. – *Algebraic Extensions and Algebraic Closure in Scratchpad.* Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 358, ed. P. Gianni, p. 440–446 (1989).

[Du89] D. Duval. – *Computations in Fields of Arbitrary Characteristic.* Computers and Mathematics, Springer, ed. E. Kaltofen, S.M. Watt, p. 321–326 (1989).

[Du90] D. Duval. – *Calculs avec Discussion Automatique : Description et Applications.* Publications du Département de Mathématiques de Limoges p. 17–33 (1990).

[DR] D. Duval, J.-C. Reynaud. – *Sketches and Computation (Part I): Basic Definitions and Static Evaluation* et *(Part II): Dynamic Evaluation and Applications.* Mathematical Structures in Computer Science, à paraître.

[DGV] D. Duval, L. González-Vega. – *Dynamic Evaluation and Real Closure.* Proceedings Imacs 93 (1993).

[Du] D. Duval. – *Evaluation Dynamique et Clôture Algébrique en Axiom.* J. of Pure and Applied Algebra, à paraître.

[DGD] D. Duval, T. Gómez-Díaz. – *A Lazy Method for Triangularizing Polynomial Systems.* Soumis à publication.

[GPT] A. Galligo, L. Pottier, C. Traverso. – *Graeter Easy Divisor and Standar Basis Completion Algorithms.* Symbolic and Algebraic Computation, Springer Lecture Notes in Computer Science 358, ed. P. Gianni, p. 162-176 (1989).

[GR] M.J. González-López, T. Recio. – *The ROMIN Inverse Geometric Model and the Dynamical Evaluation Method.* The Scafi papers draft, p. 117-141 (1991).

[Go93] T. Gómez-Díaz. – *Exemples of Using Dynamic Constructible Closure.* Proceedings Imacs 93 (1993).

[Go] T. Gómez-Díaz. – *I. Calcul de la Forme Canonique de Jordan.* Cette thèse (1994).

[JS] Richard D. Jenks, Robert S. Sutor. – *Axiom, The Scientific Computation System.* Springer-Verlag (1992).

[Kal] M. Kalkbrener. – *A Generalized Euclidean Algorithm for Computing Triangular Representations of Algebraic Varieties.* Journal of Symbolic Computation 15, p.143-167 (1993).

[Kap] I. Kaplansky. – *Linear Algebra and Geometry.* Allyn and Bacon, Inc. (1969).

[La] D. Lazard. – *A New Method for Solving Algebraic Systems of Positive Dimension.* Discrete Applied Mathematics 33, p. 147-160 (1991).

[Pa] *Petite Encyclopedie des Mathématiques.* Ed. Pagoulatos (1980).

[Se] A. Seidenberg. – *An Elimination Theory for Differential Algebra.* Univ. of California Publications in Math 3 p. 31-65 (1954).

[TZ] O. Taussky, H. Zassenhaus. – *On the Similarity Transformation between a Matrix and its Transpose.* Pac. J. of Math. 9 p. 893-896 (1959).

Conclusion

Dans ce mémoire, nous avons traité quelques applications de l'évaluation dynamique. Nous montrons que le domaine des applications de cette technique de calcul est très large : nous l'avons utilisée pour le calcul de la forme canonique de Jordan, pour la résolution des systèmes polynomiaux et pour la démonstration de théorèmes en géométrie élémentaire.

Ce mémoire est composé de deux parties indépendantes. A chacune de ces parties correspond un programme implanté en Axiom.

La première partie est consacrée au calcul de la forme canonique de Jordan. Nous avons utilisé la clôture algébrique dynamique pour les calculs avec les valeurs propres. La réalisation de ce programme nous a conduit à une étude théorique dont le principal résultat est :

> *Il est bien connu que certaines matrices de passage peuvent être décrites par blocs, les éléments de chaque bloc s'exprimant en fonction d'une seule valeur propre d'une matrice donnée. Nous avons montré la même propriété pour l'inverse de ces matrices de passage.*

De plus, nous décrivons deux nouveaux algorithmes permettant de calculer la matrice inverse en manipulant une seule valeur propre à tout moment du calcul. En particulier, ce résultat s'applique aux matrices de passage entre une matrice donnée et sa forme canonique de Jordan. De plus, nous décrivons en ce cas une version plus efficace de chacun de nos deux algorithmes.

La deuxième partie est consacrée au calcul avec des paramètres à valeurs dans un corps. Pour cela, nous avons réalisé le programme de la clôture constructible dynamique. Grâce à lui :

> *Etant donné un calcul faisant intervenir des paramètres, nous sommes capables d'obtenir, en utilisant l'évaluation dynamique, les diverses solutions en fonction des valeurs des paramètres.*

C'est la première fois qu'un tel calcul est implanté de manière tout-à-fait systématique.

C'est par l'intermédiaire de ce programme que l'évaluation dynamique propose de nouvelles méthodes pour la résolution des systèmes polynomiaux et pour la démonstration de théorèmes en géométrie élémentaire.

Les méthodes de calcul proposées ici sont valables sur un corps de base quelconque, de même que les programmes implantés. Nous avons vu par exemple comment l'on peut calculer la forme canonique de Jordan d'une matrice avec des paramètres dans ses coefficients.

Finalement, nous avons présenté l'état actuel de notre travail, déjà dépassé par les projets pour l'avenir.

Des améliorations sur ces travaux sont toujours possibles, nous les espérons pour un futur proche.

Des nouveaux projets sont déjà envisagés, en particulier :

pouvons-nous automatiser des calculs avec des paramètres de type entier ?

Résumé

Cette thèse est consacrée à l'étude et la programmation de quelques applications de l'évaluation dynamique. Elle est composée de deux travaux indépendants.

Dans la première partie de ce mémoire, nous étudions le calcul de la forme canonique de Jordan d'une matrice. Nous présentons ici les résultats théoriques obtenus et quelques algorithmes implantés.

Dans la deuxième partie, nous décrivons le programme de la clôture constructible dynamique que nous avons réalisé pour effectuer des calculs avec des paramètres. Nous utilisons ce programme pour la résolution des systèmes polynomiaux et pour la démonstration automatique de théorèmes en géométrie élémentaire.

Abstract

This thesis is devoted to the study and programmation of some applications of dynamic evaluation. It is made of two different parts.

In the first part we study the computation of the Jordan canonical form of a matrix. We present here the theorical results obtained and some of the algorithms that we have implemented.

In the second part we describe the program of dynamic constructible closure for computations with parameters. We use this program for solving polynomial systems and for automatic proving of geometric theorems.

Resumen

Esta tesis está dedicada al estudio y la programación de algunas aplicaciones de la evaluación dinámica. Está compuesta de dos trabajos independientes.

En la primera parte de esta memoria, estudiamos el cálculo de la forma canónica de Jordan de una matriz. En ella presentamos los resultados teóricos obtenidos y algunos de los algoritmos programados.

En la segunda parte, describimos el programa de la clausura constructible dinámica que hemos realizado para efectuar cálculos con parámetros. Este programa es utilizado para la resolución de sistemas de ecuaciones polinómicas y para la demostración automática de teoremas en geometría.