INRIA
SOPHIA ANTIPOLIS

STAGE DE
MAGISTÈRE
DEUXIÈME ANNÉE
Juin-Juillet 1997

rapport de stage

Anneau local d'un point épais

effectué par

Benoît BERTRAND

sous la responsabilité de Bernard MOURRAIN

Table des matières

1 Présentation du Stage

1.1 L'INRIA

L'Institut National de Recherche en Informatique et en Automatique, est un établissement public à caractère scientifique et technologique (EPST) placé sous la double tutelle du ministère de l'éducation nationale, de l'enseignement supérieur et de la recherche et du ministère de l'industrie, de la poste et des télécommunications.

A Sophia Antipolis, l'unité de recherche INRIA représente un potentiel de 400 personnes, dont 140 sur poste budgétaire. Cette unité entretient de nombreuses collaborations avec les laboratoires de la région ainsi qu'avec le milieu industriel comme peuvent en témoigner les quatre sociétés de technologie créées par d'anciens chercheurs du site. Les projets sont répartis sur quatre thèmes principaux de recherche :

-Thème 1 : Réseaux et systèmes.

-Thème 2 : Génie logiciel et calcul symbolique.

-Thème 3 : Interaction homme-machine, images, données, connaissances.

-Thème 4 : Simulation et optimisation de systèmes complexes.

1.2 Le projet SAFIR

L'équipe SAFIR (Systèmes Algébriques Formels pour l'Industrie et la Recherche) travaille sur le deuxième thème de recherche de l'INRIA sous la direction de Stephen WATT qui est un des créateurs de Maple et d'Axiom. Elle développe et promeut des méthodes symboliques nouvelles, en particulier pour le calcul scientifique et met au point des outils logiciels pour le mathématicien et l'ingénieur. Ce projet a été créé à l'initiative de Jacques Morgenstern, décédé en avril 1994. SAFIR est un projet commun avec l'université de Nice-Sophia Antipolis et le CNRS.

Ces axes de recherche sont :

-Les langages de programmation pour le calcul formel, la conception et l'architecture des systèmes de calcul formel (systèmes Sisyphe et Ulysse), la conception d'environnement pour le calcul scientifique : coopération entre systèmes (numériques et formels) et interfaces utilisateurs (CAS/PI, Central Control, Izic), l'animation d'algorithmes (Agat).

-L'étude des équations polynomiales et des méthodes de géométrie algébrique effectives, les applications aux problèmes de robotique et de vision, l'étude d'équations diophantiennes linéaires et de la complexité algébrique : analyses fines de l'influence de la représentation des données.

-Le lien entre calcul formel et calcul numérique, la transformation de programmes numériques : dérivation automatique de programmes FORTRAN (système Odyssée), l'application à l'automatique : identification de fonctions de transfert (programme Hyperion).

2 Introduction

Le but de ce travail est d'implémenter un algorithme efficace de calcul de la multiplicité (et de l'anneau local) de points épais, c'est-à-dire multiples et isolés dans des variétés algébriques. Ces dernières sont des ensemble de points définis par des systèmes de polynômes $\mathbf{P} = \{p_1, \dots, p_m\}$ à plusieurs indéterminées. Pour trouver la multiplicité d'un de ces points épais nous allons être amenés à considérer le dual $\widehat{R}$ de R où $R = \mathbb{K}[x_1, ..., x_n]$ et $\mathbb{K}$ est le corps de base de caractéristique nulle (par exemple $\mathbb{Q}$). Nous nous intéresserons alors à l'orthogonal de l'idéal engendré par $\mathbf{P}$ (ou plus précisément à l'orthogonal des composantes primaires des points considérés) dont le nombre de générateurs en tant qu'espace vectoriel nous donnera la multiplicité du point épais de la variété.

3 Dualité et systèmes inverses

Dans cette section, nous allons nous intéresser aux formes linéaires sur l'anneau des polynômes. Les objets que nous étudierons seront donc des éléments du dual de $R = \mathbb{K}[x_1, \cdots, x_n] = \mathbb{K}[\mathbf{x}]$. Dans toute la suite, $\mathbb{K}$ sera un corps de caractéristique nulle.

3.1 Le dual de $R = \mathbb{K}[\mathbf{x}]$

La forme linéaire non nulle la plus « simple » est l'*évaluation en un point* $\zeta \in \mathbb{K}^n$,

$$\begin{aligned} \partial_\zeta^{\mathbf{0}} \, : \, R &\to \mathbb{K} \\ p &\mapsto \partial_\zeta^{\mathbf{0}}(p) = p(\zeta). \end{aligned}$$

On peut aussi, pour tout multi-indice $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{N}^n$, considérer la forme linéaire

$$\begin{aligned} \partial_\zeta^{\mathbf{a}} : R &\to \mathbb{K} \\ p &\mapsto \partial_\zeta^{\mathbf{a}}(p) = d_{x_1}^{a_1} \cdots d_{x_n}^{a_n}(p)(\zeta), \end{aligned}$$

où d_{x_i} désigne la dérivation par rapport à la variable x_i. Nous notons, $\partial_{i,\zeta} = \partial_\zeta^{\mathbf{a}}$, avec $a_i = 1$ et $a_j = 0, j \neq i$. Avec ces notations, $\partial_\zeta^{\mathbf{a}} = \partial_{1,\zeta}^{a_1} ... \partial_{n,\zeta}^{a_n}$. Nous travaillerons plutôt avec la forme linéaire $\mathbf{d}_\zeta^{\mathbf{a}} = \frac{1}{\prod_{i=1}^n a_i!} \partial_\zeta^{\mathbf{a}}$, dont l'utilisation est plus simple. Lorsque $\zeta = 0, \mathbf{d}_\zeta^{\mathbf{a}}$ sera noté $\mathbf{d}^{\mathbf{a}}$.

Via ce formalisme, l'algèbre $\mathbb{K}[[\mathbf{d}_\zeta^{(1,0,\dots,0)}, \dots, \mathbf{d}_\zeta^{(0,\dots,0,1)}]]$ des séries formelles, encore notée $\mathbb{K}[[\mathbf{d}_\zeta^{\mathbf{e}_1}, \dots, \mathbf{d}_\zeta^{\mathbf{e}_n}]]$ (où $(\mathbf{e}_i)_i$ est la base canonique des multi-indices $\mathbf{a}$), **s'identifie** à $\widehat{R}$. Cette identification est réalisée par le développement de Taylor en ζ. En effet, étant donné $f \in \mathbb{K}[\mathbf{x}]$, on peut le décomposer en

$$f(\mathbf{x}) = \sum_{\mathbf{a}\in\mathbb{N}^n} \mathbf{d}_\zeta^{\mathbf{a}}(f)\,(\mathbf{x}-\zeta)^{\mathbf{a}},$$

où $(\mathbf{x}-\zeta)^{\mathbf{a}} = \prod_{i=1}^n (x_i-\zeta_i)^{a_i}$. Par conséquent, pour toute forme linéaire Λ sur R,

$$\Lambda(f) = \sum_{\mathbf{a}\in\mathbb{N}^n} \Lambda((\mathbf{x}-\zeta)^{\mathbf{a}})\,\mathbf{d}_\zeta^{\mathbf{a}}(f).$$

Nous pouvons donc identifier Λ à la série formelle S en les $\mathbf{d}_\zeta^{\mathbf{e}_1}, \dots, \mathbf{d}_\zeta^{\mathbf{e}_n}$ où

$$S = \sum_{\mathbf{a}\in\mathbb{N}^n} \Lambda((\mathbf{x}-\zeta)^{\mathbf{a}})\,\mathbf{d}_\zeta^{\mathbf{a}}.$$

Comme

$$\mathbf{d}_\zeta^{a_i}((x_i-\zeta)^{a_i}) = 1,$$

nous avons

$$\mathbf{d}_\zeta^{\mathbf{a}}((x-\zeta)^{\mathbf{b}}) = \begin{cases} 1 \text{ si } \ \mathbf{a}=\mathbf{b}, \\ 0 \text{ sinon.} \end{cases}$$

La base $(\mathbf{d}_\zeta^{\mathbf{a}})_{\mathbf{a}\in\mathbb{N}^n}$ de $\widehat{R}$ est donc la base duale de la base monomiale $((\mathbf{x}-\zeta)^{\mathbf{a}})_{\mathbf{a}\in\mathbb{N}^n}$ de R. Cette notation n'est pas multiplicative : $\mathbf{d}_\zeta^{\mathbf{a}}\,\mathbf{d}_\zeta^{\mathbf{a}'} \neq \mathbf{d}_\zeta^{\mathbf{a}+\mathbf{a}'}$.

À partir de maintenant, nous allons **identifier** $\widehat{R}$ avec $\mathbb{K}[[\mathbf{d}_\zeta^{\mathbf{e}_1}, \dots, \mathbf{d}_\zeta^{\mathbf{e}_n}]]$. Les formes linéaires seront donc vues *comme des séries formelles de* $\mathbb{K}[[\mathbf{d}_\zeta^{\mathbf{e}_1}, \dots, \mathbf{d}_\zeta^{\mathbf{e}_n}]]$. On notera aussi leur espace $\mathbb{K}[[\mathbf{d}_\zeta]]$. Lorsque $\zeta = 0$, $\mathbb{K}[[\mathbf{d}_0]]$ sera noté $\mathbb{K}[[\mathbf{d}_1, \dots, \mathbf{d}_n]]$ ou $\mathbb{K}[[\mathbf{d}]]$. Si c'est clair dans le contexte, on pourra aussi utiliser $\mathbf{d}_i$ pour $\mathbf{d}^{\mathbf{e}_i}$. Ce sera le cas quand on ne considérera plus que le cas $\zeta = 0$.

L'espace vectoriel $\widehat{R}$ (identifié à $\mathbb{K}[[\mathbf{d}]]$) est muni d'une structure de R-module de la façon suivante :

$$\begin{aligned} p\cdot\Lambda : R &\to \mathbb{K} \\ q &\mapsto \Lambda(p\,q). \end{aligned}$$

On a en particulier:

$$x_i \cdot \mathbf{d}^{a_i}(p) = \mathbf{d}^{a_i}(x_i p) = \frac{1}{\prod_{j=1}^n a_j!}\partial^{a_i}(x_i p) = \frac{1}{\prod_{j=1}^n a_j!} d_{x_1}^{a_1} \cdots d_{x_n}^{a_n}(x_i p)(0),$$

et comme

$$d_{x_i}^{a}(x_i\, p) = a\, d_{x_i}^{a-1}(p) + x_i\, d_{x_i}^{a}(p),$$

on en déduit que

$$x_i \cdot \mathbf{d}^{\mathbf{a}}(p) = \frac{1}{\prod_{j=1}^n a_j!} d_{x_1}^{a_1} \cdots d_{x_{i-1}}^{a_{i-1}} d_{x_{i+1}}^{a_{i+1}} \cdots d_{x_n}^{a_n}(a_i d_{x_i}^{a_i-1}(p) + x_i d_{x_i}^{a_i}(p))(0).$$

Or $x_i q(0) = 0$ ce qui implique que :

$$\begin{aligned} x_i \cdot \mathbf{d}^{\mathbf{a}}(p) &= \frac{1}{\prod_{j=1}^n a_j!} d_{x_1}^{a_1} \cdots d_{x_{i-1}}^{a_{i-1}} d_{x_{i+1}}^{a_{i+1}} \cdots d_{x_n}^{a_n}(a_i d_{x_i}^{a_i-1}(p))(0) \\ &= \frac{a_i}{\prod_{j=1}^n a_j!} d_{x_1}^{a_1} \cdots d_{x_{i-1}}^{a_{i-1}} d_{x_i}^{a_i-1} d_{x_{i+1}}^{a_{i+1}} \cdots d_{x_n}^{a_n}(p)(0) \\ &= \mathbf{d}^{\mathbf{a}-\mathbf{e}_i}(p). \end{aligned}$$

Par abus de langage, on dira que

$$\begin{aligned} x_i \cdot \Lambda : R &\to \mathbb{K} \\ q &\mapsto \Lambda(x_i\, q) \end{aligned}$$

est la i-ème « dérivée » de Λ. En faisant cet abus, on peut donc dire que la multiplication par x_i correspond à une « dérivation » de $\mathbb{K}[[\mathbf{d}]]$. On appellera alors i-ème dérivation de $\mathbb{K}[[\mathbf{d}]]$ et on notera δ_i l'application

$$\begin{aligned} \delta_i : \widehat{R} &\to \widehat{R} \\ \Lambda &\mapsto x_i \cdot \Lambda\,. \end{aligned}$$

Dans toute la suite, pour plus de simplicité, nous nous placerons au point $\zeta = 0$. Une translation suffit alors pour se replacer en ζ et la structure de l'anneau local n'est en rien modifiée. Le traitement en un point ζ quelconque est fait dans la première partie de [3] dont cette section et la section 5 ne sont que des adaptations comprenant quelques explications supplémentaires.

3.2 L'orthogonal d'un idéal et l'orthogonal d'un espace vectoriel

Définition 3.2.1 — *Pour tout idéal I de R, on définit le sous-espace vectoriel de $\widehat{R}$:*

$$I^{\perp} = \{\Lambda \in \widehat{R};\ \forall p \in I,\ \Lambda(p) = 0\}.$$

Pour tout sous-espace vectoriel D de $\widehat{R}$, on définit le sous-espace vectoriel de R :

$$D^{\perp} = \{p \in R;\ \forall \Lambda \in D,\ \Lambda(p) = 0\}.$$

Remarque : L'orthogonal d'un idéal I de R n'est pas un idéal de $\mathbb{K}[[\mathbf{d}]]$.(En effet soit I, un idéal de $\mathbb{K}[\mathbf{x}]$ inclus dans $(x_1, \cdots, x_n)$, alors $1 \in \mathbb{K}[[\mathbf{d}]]$ est dans $I^{\perp}$ et si $I \neq \mathbb{K}[\mathbf{x}]$, $I^{\perp} \neq \mathbb{K}[[\mathbf{d}]]$)

Définition 3.2.2 — *L'espace vectoriel $I^{\perp}$ est appelé le **système inverse** de I.*
(voir [6]).

Ces éléments peuvent se voir comme des formes linéaires sur $B = R/I$: Soient p_1 et p_2 des éléments de R ayant la même projection dans B, et soit Λ un élément de $I^{\perp}$. On a alors $\Lambda \cdot p_1 = \Lambda \cdot p_2$ et Λ passe au quotient en $\tilde{\Lambda}$.

Plus précisément, la projection $\pi : R \to B$ induit une application

$$\begin{aligned} \pi_* : \widehat{B} &\to I^{\perp} \\ \tilde{\Lambda} &\mapsto \tilde{\Lambda} \circ \pi \end{aligned}$$

et nous avons la propriété suivante :

Proposition 3.2.3 — *L'application π_* est un isomorphisme entre $\widehat{B}$ et $I^{\perp}$.*

Preuve. Soit $\Lambda \in I^{\perp}$, Λ passe au quotient en $\tilde{\Lambda}$ et $\tilde{\Lambda} \circ \pi = \Lambda$. Donc π_* est surjective. Soient $\tilde{\Lambda}_1$ et $\tilde{\Lambda}_2 \in \widehat{B}$ telles que $\tilde{\Lambda}_1 \circ \pi = \tilde{\Lambda}_2 \circ \pi$. Alors $\Lambda_1 - \Lambda_2 \in I^{\perp}$ donc $\tilde{\Lambda}_1 = \tilde{\Lambda}_2$.

Dans la suite, on **identifie** $I^{\perp}$ avec $\widehat{B}$.

Proposition 3.2.4 — *Soit $I = (p_1, \dots, p_s)$ un idéal de R; alors son système inverse est*

$$I^\perp = \{\Lambda \in \mathbb{K}[[\mathbf{d}]];\ p_i(\delta_1, \dots, \delta_n)(\Lambda) = 0, 1 \leq i \leq s\}.$$

Preuve. Par convention pour tout $\Lambda \in \mathbb{K}[[\mathbf{d}]]$, $x_i \cdot \Lambda = \delta_i(\Lambda)$. Donc, pour tout $p \in R$, $p \cdot \Lambda = P(\delta_1, \dots, \delta_n)(\Lambda)$. Ceci montre que $\Lambda \in I^\perp$ si et seulement si $\forall q \in R$, $\forall i = 1, \dots, s$,

$$\Lambda(p_i\, q) = p_i \cdot \Lambda(q) = p_i(\delta_1, \dots, \delta_n)(\Lambda)(q) = 0,$$

ou encore $\forall i = 1, \dots, s$, $p_i(\delta_1, \dots, \delta_n)(\Lambda) = 0$.

Le système inverse $I^\perp$ est stable par dérivation car I est stable par multiplication. En fait, il y a une correspondance entre les idéaux de R et certains sous-espaces vectoriels de $\mathbb{K}[[\mathbf{d}]]$ stables par dérivation.

Théorème 3.2.5 — *Les idéaux de R sont en bijection avec les sous-espaces vectoriels de $\mathbb{K}[[\mathbf{d}]]$ stables par dérivation et fermés pour la topologie $(\mathbf{d}_1, \dots, \mathbf{d}_n)$-adique.*

Voir [4].

La topologie $\mathfrak{p}$-adique est celle pour laquelle la convergence est définie comme suit :

$$lim_{n->\infty}(p_n) = p \text{ si } \forall k, \exists N_k \text{ tel que } p_n - p \in \mathfrak{p}^k.$$

Cette bijection consiste à prendre l'orthogonal dans le dual et dans le bidual, c'est-à-dire l'espace lui-même. Pour tout idéal I de R et pour tout sous-espace vectoriel fermé D de $\widehat{R}$, on a en effet les propriétés

$$I^{\perp\perp} = I,\ D^{\perp\perp} = D.$$

Donnons maintenant quelques propriétés des systèmes inverses.

Définition 3.2.6 — *Soient $\Lambda_1, \dots, \Lambda_s \in \mathbb{K}[[\mathbf{d}]]$, on note*

$$\langle\langle \Lambda_1, \dots, \Lambda_s \rangle\rangle$$

le sous-espace vectoriel de $\mathbb{K}[[\mathbf{d}]]$ engendré par les éléments Λ_i et leurs dérivées.

Proposition 3.2.7 — *Soient I et J deux idéaux de R, alors*

- $I \subset J \Leftrightarrow J^\perp \subset I^\perp$
- $(I \cap J)^\perp = I^\perp + J^\perp$
- $(I + J)^\perp = I^\perp \cap J^\perp$.

3.2.1 Division d'idéaux

Certaines propriétés des idéaux, difficiles à décrire ou à calculer dans R, se traduisent particulièrement bien sur les systèmes inverses. La division en est un exemple.

Définition 3.2.8 — *Soient A un anneau et $\mathfrak{a}$ un idéal de A et $\{z_1, ..., z_r\}$ un ensemble fini d'éléments de A, par définition $(\mathfrak{a} : \{z_1, ..., z_r\})$ est l'ensemble des $y \in A$ tels que quelque soit $i \in \{1, ..., r\}$ $yz_i \in \mathfrak{a}$.*

On remarque que $(\mathfrak{a} : \{z_1, ..., z_r\})$ est un idéal.

Proposition 3.2.9 — *Pour tout $\Lambda_1, \ldots, \Lambda_s \in \mathbb{K}[[\mathbf{d}]]$, et $p_1, \ldots, p_t \in R$,*

$$\langle\langle \Lambda_1, \ldots, \Lambda_s \rangle\rangle^\perp : (p_1, \ldots, p_t) = \langle\langle p_j \cdot \Lambda_i \rangle\rangle^\perp_{1 \le i \le s, 1 \le j \le t}$$

Preuve. Comme $\langle\langle \Lambda_1, \ldots, \Lambda_s \rangle\rangle^\perp$ est un idéal de R,

$$\begin{aligned} P \in \langle\langle \Lambda_1, \ldots, \Lambda_s \rangle\rangle^\perp : (p_1, \ldots, p_t) &\Leftrightarrow \forall j,\ p_j\, P \in \langle\langle \Lambda_1, \ldots, \Lambda_s \rangle\rangle^\perp \\ &\Leftrightarrow \forall i, j, \forall Q \in R,\ \Lambda_i(p_j\, P\, Q) = 0 \\ &\Leftrightarrow \forall i, j, \forall Q \in R, p_j \cdot \Lambda_i(P\, Q) = 0 \\ &\Leftrightarrow P \in \langle\langle p_j \cdot \Lambda_i \rangle\rangle^\perp. \end{aligned}$$

4 Décomposition d'un idéal, composante isolée.

4.1 Cas général

Dans toute cette section A est un anneau. On notera $\sqrt{\mathfrak{a}}$ la racine de l'idéal $\mathfrak{a}$, c'est-à-dire $\sqrt{\mathfrak{a}} = \{x \in A,\ \exists n\ :\ x^n \in \mathfrak{a}\}$.

Définition 4.1.1 — *On dit qu'un idéal $\mathfrak{a}$ d'un anneau A est* **primaire** *si $\mathfrak{a} \neq A$ et si pour tout couple d'éléments (x, y) de A,*

$$xy \in \mathfrak{a} \Rightarrow x \in \mathfrak{a} \text{ ou } \exists n \in \mathbb{N}^* \text{ tq. } y^n \in \mathfrak{a}.$$

On a aussi la définition équivalente :

Définition 4.1.2 — *Un idéal $\mathfrak{a}$ d'un anneau A est* **primaire** *si $A/\mathfrak{a} \neq 0$ et, dans $A/\mathfrak{a}$, tout diviseur de zéro est nilpotent.*

Proposition 4.1.3 — *Soit $\mathfrak{a}$ un idéal primaire de A alors $\sqrt{\mathfrak{a}}$ est le plus petit idéal premier contenant $\mathfrak{a}$.*

Preuve. $\sqrt{\mathfrak{a}}$ est l'intersection des idéaux premiers contenant $\mathfrak{a}$. Il suffit donc de montrer qu'il est premier. Soit x, y dans A tels que $xy \in \sqrt{\mathfrak{a}}$ alors il existe $n \in \mathbb{N}$ tel que $(xy)^n \in \mathfrak{a}$ donc $x^n \in \mathfrak{a}$ ou il existe $m \in \mathbb{N}$ tel que $y^{nm} \in \mathfrak{a}$. Alors $x \in \sqrt{\mathfrak{a}}$ ou $y \in \sqrt{\mathfrak{a}}$, ce qui prouve que $\sqrt{\mathfrak{a}}$ est premier.

Si $\mathfrak{p} = \sqrt{\mathfrak{a}}$, on dit que $\mathfrak{a}$ est $\mathfrak{p}$ primaire.

Proposition 4.1.4 — *Si les $\mathfrak{b}_i$ sont $\mathfrak{p}$-primaires $\mathfrak{b} = \cap_{i=1}^n \mathfrak{b}_i$ est $\mathfrak{p}$-primaire.*

Preuve. $\sqrt{\mathfrak{b}} = \cap_{i=1}^n \sqrt{\mathfrak{b}_i}$ car l'intersection d'idéaux radicaux est un idéal radical et que le passage à la racine préserve l'inclusion. Donc $\sqrt{\mathfrak{b}} = \mathfrak{p}$. De plus si $xy \in \mathfrak{b}$ et $y \notin \mathfrak{b}$ alors il existe i tel que $y \notin \mathfrak{b}_i$ donc $x \in \sqrt{\mathfrak{b}_i} = \mathfrak{p}$ et $\mathfrak{b}$ est $\mathfrak{p}$-primaire.

Proposition 4.1.5 — *si $\mathfrak{a}$ est un idéal $\mathfrak{p}$-primaire de A et $x \in \mathfrak{a}$ alors,* *(i) si $x \in \mathfrak{a}$, $(\mathfrak{a} : x) = A$,*

(ii) si $x \notin \mathfrak{a}$, $(\mathfrak{a} : x)$ est $\mathfrak{p}$-primaire et donc $\sqrt{(\mathfrak{a} : x)} = \mathfrak{p}$,

(iii) si $x \notin \mathfrak{p}$, $(\mathfrak{a} : x) = \mathfrak{a}$.

Preuve.

(i) $1 \in (\mathfrak{a} : x)$.
(ii) $x \notin \mathfrak{a}$ donc si $y \in (\mathfrak{a} : x)$ alors $y \in \mathfrak{p}$ car $\mathfrak{a}$ est $\mathfrak{p}$-primaire. Donc $\mathfrak{a} \subset (\mathfrak{a} : x) \subset \mathfrak{p}$ et $\sqrt{(\mathfrak{a} : x)} = \mathfrak{p}$. Soit $yz \in (\mathfrak{a} : x)$ avec $y \notin \mathfrak{p}$ alors $xyz \in \mathfrak{a}$ et, comme $\mathfrak{a}$ est $\mathfrak{p}$-primaire, $xz \in \mathfrak{a}$ donc $z \in (\mathfrak{a} : x)$, ce qui montre que (a:x) est $\mathfrak{p}$-primaire
(iii) Si $y \notin \mathfrak{a}$ et $xy \in \mathfrak{a}$ alors $x \in \mathfrak{p}$ donc si $x \notin \mathfrak{p}$ et $xy \in \mathfrak{a}$ alors $y \in \mathfrak{a}$. Donc $(\mathfrak{a} : x) = \mathfrak{a}$

Définition 4.1.6 — *On appelle* **décomposition primaire** *d'un idéal $\mathfrak{a}$, l'écriture de $\mathfrak{a}$ comme intersection finie d'idéaux primaires $\mathfrak{b}_i$:*

$$\mathfrak{a} = \bigcap_{i=1}^{n} \mathfrak{b}_i \, .$$

Si de plus,

(i) les $\sqrt{\mathfrak{b}_i}$ sont tous distincts,

(ii) quelque soit i, $\mathfrak{b}_i \nsubseteq \cap_{j=1, j\neq i}^{n} \mathfrak{b}_j$,

on dit que la décomposition est **minimale.**

Il n'existe pas toujours de décomposition primaire, mais si elle existe on peut toujours trouver une décomposition minimale.

Théorème 4.1.7 — *Soit $\mathfrak{a}$ un idéal décomposable et soit $\mathfrak{a} = \cap_{i=1}^{n} \mathfrak{b}_i$ une décomposition primaire de $\mathfrak{a}$. Alors les $\sqrt{\mathfrak{b}_i}$ sont les idéaux premiers de la forme $\sqrt{(\mathfrak{a} : x)}$ où $x \in A$ et sont donc indépendants de la décomposition de A.*

Preuve. Soit $x \in A$, on a $(\mathfrak{a} : x) = \cap_{j=1}^{n} (\mathfrak{b}_j : x)$ $(xy \in \mathfrak{a} \Leftrightarrow \forall i, xy \in \mathfrak{b}_i)$. Donc $\sqrt{(\mathfrak{a} : x)} = \cap_{j=1}^{n} \sqrt{(\mathfrak{b}_j : x)} = \cap_{j=1, x \notin \mathfrak{b}_j}^{n} \sqrt{(\mathfrak{b}_j : x)}$ car si $x \in \mathfrak{b}_i, (\mathfrak{b}_i : x) = A$ (d'après 4.1.5). Si $\sqrt{(\mathfrak{a} : x)}$ est premier alors il existe un i tel que $\sqrt{(a : x)} = \sqrt{(\mathfrak{b}_i : x)}$ car un idéal premier qui est l'intersection d'idéaux premiers est égal à l'un de ceux-ci. Donc tout idéal premier de la forme $\sqrt{(\mathfrak{a} : x)}$ est un des $\sqrt{\mathfrak{b}_j}$ car, comme $\mathfrak{b}_i$ est primaire , $\sqrt{(\mathfrak{b}_i : x)} = \sqrt{\mathfrak{b}_i}$ si $x \notin \mathfrak{b}_i$ etăsinon $\sqrt{(\mathfrak{b}_i : x)} = A$ (d'après 4.1.5). Réciproquement, pour tout i, il existe $x_i \notin \mathfrak{b}_i$ tel que $x_i \in \cap_{j=1}^{n} \mathfrak{b}_j$ sinon la décomposition ne serait pas minimale. Et on a donc $\sqrt{(\mathfrak{a} : x_i)} = \sqrt{\mathfrak{b}_i}$.

Définition 4.1.8 — *On dit que les $\sqrt{\mathfrak{b}_i}$ appartiennent à $\mathfrak{a}$.*

Définition 4.1.9 — *Avec les notations précédentes, on dit que $\mathfrak{b}_i$ est* **isolé** *si*

$$\sqrt{\mathfrak{b}_j} \subset \sqrt{\mathfrak{b}_i} \Rightarrow i = j \,.$$

4.2 Cas des anneaux noethériens

L'anneau A est maintenant noethérien. On va voir que la notion d'irréductibilité entraîne celle d'idéal $\mathfrak{p}$-primaire.

Définition 4.2.1 — *Un idéal $\mathfrak{a}$ est dit* **irréductible** *si*

$$\mathfrak{a} = \mathfrak{b} \cap \mathfrak{c} \Rightarrow (\mathfrak{a} = \mathfrak{b} \text{ ou } \mathfrak{a} = \mathfrak{c}) \,.$$

Proposition 4.2.2 — *Dans un anneau noethérien A, tout idéal $\mathfrak{a}$ est une intersection finie d'idéaux irréductibles.*

Preuve. Raisonnons par l'absurde : supposons que ce ne soit pas le cas. Soit $\mathfrak{E}$ l'ensemble des idéaux qui ne sont pas intersection d'idéaux irréductibles. Par hypothèse $\mathfrak{E}$ est non vide donc il admet un élément maximal $\mathfrak{c}$. Comme $\mathfrak{c}$ est réductible, il existe $\mathfrak{b}_1$ et $\mathfrak{b}_2$ tel que $\mathfrak{c} = \mathfrak{b}_1 \cap \mathfrak{b}_2$ avec $\mathfrak{c} \nsubseteq \mathfrak{b}_1$ et $\mathfrak{c} \nsubseteq \mathfrak{b}_2$ donc $\mathfrak{b}_1$ et $\mathfrak{b}_2$ sont des intersections finies d'idéaux irréductibles et $\mathfrak{c}$ aussi, ce qui contredit l'hypothèse.

Proposition 4.2.3 — *Dans un anneau noethérien, tout idéal irréductible est primaire.*

Preuve. Soit $\mathfrak{a}$ un idéal de A, si $\mathfrak{a}$ est irréductible, (0) est irréductible dans $A/\mathfrak{a}$. En effet si $(0) = \mathfrak{b} \cap \mathfrak{c}$ et si Π est la projection canonique, $\Pi^{-1}(\mathfrak{b} \cap \mathfrak{c}) = \Pi^{-1}(\mathfrak{b}) \cap \Pi^{-1}(\mathfrak{c}) = \mathfrak{a}$. Si $(0) \subsetneq \mathfrak{b}$ et $(0) \subsetneq \mathfrak{c}$ alors $\mathfrak{a} \subsetneq \Pi^{-1}(\mathfrak{b})$ et $ag \subsetneq \Pi^{-1}(\mathfrak{c})$. Supposons donc (0) irréductible. Soient x et y tels que $(y) \neq (0)$ et $xy = 0$. Comme $Ann(x^i)$ est une suite stationnaire car $A/\mathfrak{a}$ est noethérien, il existe n tel que $Ann(x^n) = Ann(x^{n+1})$. On a alors $(x^n) \cap (y) = 0$. En effet, si $a \in (x^n) \cap (y)$ alors il existe b et c tels que $a = bx^n = cy$ alors $ax = cyx = 0$ donc $bx^{n+1} = 0$ ce qui implique que $b \in Ann(x^n) = Ann(x^{n+1})$ donc $bx^n = 0$ et $a = 0$. Comme $(y) \neq (0)$ et que (0) est irréductible, $(x^n) = (0)$ c'est-à-dire $x^n = 0$. D'après la définition4.1.2, ceci entraîne que $\mathfrak{a}$ est primaire.

L'existence d'une décomposition en idéaux irréductibles entraîne donc celle d'une décomposition primaire. D'où le résultat suivant.

Théorème 4.2.4 — *Dans un anneau Noethérien tout idéal admet une décomposition primaire.*

Nous aurons aussi besoin du propriété suivante :

Proposition 4.2.5 —*Si A est noethérien, tout idéal contient une puissance de sa racine.*

Preuve. Soit $\mathfrak{a}$ un idéal, $\sqrt{\mathfrak{a}}$ est engendré par un nombre fini de générateurs $(x_1, \cdots, x_n)$. Il existe donc N tel que quelque soit $i \in \{1, \cdots, n\}$, $x_i^N \in \mathfrak{a}$.Donc pour tous scalaires λ_i, $(\sum_{i=1}^n \lambda_i\, x_i)^{n \cdot N+1} \in \mathfrak{a}$. Comme tout élément x de $\sqrt{\mathfrak{a}}$ s'écrit $x = \sum_{i=1}^n \lambda_i\, x_i$, $(\sqrt{\mathfrak{a}})^{n \cdot N+1} \subset \mathfrak{a}$

4.3 Lien avec la géométrie

À partir de maintenant $\mathbb{K}$ est un corps algébriquement clos.

En effet, soit Σ une variété algébrique définie par un idéal I. Comme $\mathbb{K}[\mathbf{x}]$ est noethérien, I admet une décomposition en idéaux irréductibles : $I = \cap_{i=1}^m Q_i$. On note $\mathfrak{p}_i$ la racine de Q_i et $V_i = V(Q_i) \subset \mathbb{K}^n$

Définition 4.3.1 — *On dit que V_i est* **isolé** *si $\mathfrak{p}_j$ ne contient aucun des autres $\mathfrak{p}_i$.*

Lorsque V_i est de dimension 0 c'est un point ζ. L'idéal I engendré par les polynômes définissant la variété a alors ζ comme composante $\mathbf{m}_\zeta$-primaire isolée. Où $\mathbf{m}_\zeta = ((x_1 - \zeta_1) \dots (x_n - \zeta_n))$ est l'idéal maximal définissant ζ.

Géométriquement le point ζ est isolé s'il ne se trouve pas sur une des composantes définies par les $\mathfrak{p}_i$ pour $i \neq j$.

5 Composante en un point isolé

Définition 5.0.1 — *Soit ζ un point dans une variété algébrique définie par l'idéal I. On appelle multiplicité de ζ le nombre*

$$\mu = dim((\mathbb{K}[\mathbf{x}]/I)_{\mathbf{m}_\zeta}),$$

où $(\mathbb{K}[\mathbf{x}]/I)_{\mathbf{m}_\zeta}$ est le localisé de $\mathbb{K}[\mathbf{x}]/I$ en $\mathbf{m}_\zeta$.

5.1 Points isolés

5.1.1 Un idéal $\mathbf{m}_\zeta$-primaire

Nous nous plaçons, dans ce paragraphe, dans le cas où l'idéal I définit un seul point $\zeta \in \mathbb{K}^n$, c'est-à-dire que I est $\mathbf{m}_\zeta$-primaire où $\mathbf{m}_\zeta$ est l'idéal maximal définissant ζ. Nous prendrons encore, et toujours dans un souci de simplification, $\zeta = 0$.

Proposition 5.1.1 — *Supposons que I soit $\mathbf{m}_0$-primaire; alors $I^\perp \subset \mathbb{K}[\mathbf{d}]$.*

Preuve. D'après 4.2.5 il existe un entier N tel que $\mathbf{m}_0^N \subset I \subset \mathbf{m}_0$, $\mathbf{x}^\mathbf{a} \in I$ dès que $|\mathbf{a}| = a_1 + \cdots + a_n \geq N$. Soit $\Lambda \in I^\perp$, alors

$$\Lambda = \sum_{\mathbf{a} \in \mathbb{N}^n : |\mathbf{a}| < N} \Lambda(\mathbf{x}^\mathbf{a})\, \mathbf{d}^\mathbf{a}.$$

Dans ce cas, les éléments du système inverse de I sont des polynômes en $\mathbf{d}$. De plus, comme I ne définit qu'un seul point et que $\mathbb{K}$ est algébriquement clos, l'anneau de la variété $B = R/I$ est de dimension finie.

Montrons que cette dimension est la multiplicité μ du point 0 :

Comme I est $\mathbf{m}_0$-primaire, $B = R/I$ est un anneau local dont l'idéal maximal est $\mathbf{m}_0$. Les éléments qui ne sont pas dans $\mathbf{m}_0$ sont inversibles dans B. Donc $B = R/I \simeq (R/I)_{\mathbf{m}_0}$ et B est dimension μ .

Ceci établit *une bijection entre les idéaux* $\mathbf{m}_0$*-primaires et les sous-espaces vectoriels de* $\mathbb{K}[\mathbf{d}]$*, stables par dérivation et de dimension finie* (voir [4], [6][p. 65], [5]).

5.1.2 La composante $\mathbf{m}_0$*-primaire*

Dans la pratique, il est rare de traiter directement un idéal $\mathbf{m}_0$-primaire; on a souvent affaire à des idéaux dont une composante est $\mathbf{m}_0$-primaire. Dans ce cas, nous allons voir comment le systeme inverse permet de travailler sur cette composante $\mathbf{m}_0$-primaire : Q_0.

Théorème 5.1.2 — *Soit I un idéal de R et Q_0 sa composante* $\mathbf{m}_0$*-primaire que l'on suppose isolée. Alors*

$$(I^{\perp} \cap \mathbb{K}[\mathbf{d}])^{\perp} = Q_0.$$

Preuve. Notons $D = I^{\perp} \cap \mathbb{K}[\mathbf{d}]$; nous allons montrer que $D = Q_0^{\perp}$. Comme $I \subset Q_0$, on a $Q_0^{\perp} \subset I^{\perp}$ et $Q_0^{\perp} \subset \mathbb{K}[\mathbf{d}]$ (d'après la proposition (5.1.1)), par suite, $Q_0^{\perp} \subset I^{\perp} \cap \mathbb{K}[\mathbf{d}]$.

Pour montrer l'inclusion inverse, nous utiliserons les deux propriétés suivantes

- La composante $\mathbf{m}_0$-primaire Q_0 de I est l'ensemble des polynômes f de R tels qu'il existe $g \in R$ avec $f\,g \in I$ et $g(0) \neq 0$.
  En effet d'après [1] p.54 si $S = \mathbb{K}[\mathbf{x}] \setminus \mathbf{m}_0$, $S^{-1}I = S^{-1}Q_0$. Donc pour tout $q \in Q_0$ il existe $s_1, s_2 \in S$ et $p \in I$ tels que $\frac{q}{s_1} = \frac{p}{s_2}$. Alors $s_2\, q = s_1\, p$ avec $s_1\, p \in I$ et $s_2(0) \neq 0$.
- Pour tout $\Lambda \in \mathbb{K}[\mathbf{d}]$ et tout $g \in R$,

$$\begin{aligned}(g \cdot \Lambda)(f) &= g(\delta_1, \ldots, \delta_n)(\Lambda)(f) \\ &= g(0)\Lambda(f) + (g - g(0))(\delta_1, \ldots, \delta_n)(\Lambda)(f).\end{aligned} \tag{1}$$

Montrons par récurrence sur le degré de Λ (en $\mathbf{d}$) que $D \subset Q_0^{\perp}$.

Si $\Lambda \in D$ est de degré 0, alors Λ est, à un scalaire près, l'évaluation en 0. Pour tout $f \in Q_0, g \in R$ tels que $g(0) \neq 0$ et $f\,g \in I$, $\Lambda(f\,g) = 0 = f(0)g(0)$, donc $\Lambda(f) = f(0) = 0$, et $\Lambda \in Q_0^{\perp}$.

Supposons maintenant que tous les éléments de D de degré $< d$ sont dans $Q_0^{\perp}$. Soit $\Lambda \in D$ de degré d; d'après la formule (1), pour tout $f \in Q_0$, $g \in R$ tels que $g(0) \neq 0$ et $f g \in I$,

$$\begin{aligned}\Lambda(f\,g) = 0 &= g(0)\Lambda(f) + (g - g(0))(\delta_1, \ldots, \delta_n)(\Lambda)(f) \\ &= g(0)\Lambda(f) + \rho(f),\end{aligned}$$

Comme $(g - g(0))(\delta_1, \ldots, \delta_n)$ est de degré supérieur ou égal à 1 et sans terme constant, $\rho = (g - g(0))(\delta_1, \ldots, \delta_n)(\Lambda)$ est de degré $< d$ en $\mathbf{d}$. De plus $\rho \in D$ car D est stable par dérivation puisque $I^{\perp}$ l'est. Par hypothèse de récurrence, $\rho(f) = 0$. Il en découle que $\Lambda(f) = 0$, et $\Lambda \in Q_0^{\perp}$.

Soit $N \in \mathbb{N}$ tel que $\mathbf{m}_0^N \subset Q_0$ alors $Q_0^{\perp} \subset (\mathbf{m}_0^N)^{\perp} = \mathbb{K}[\mathbf{d}]_{N-1}$ où $\mathbb{K}[\mathbf{d}]_{N-1}$ est l'espace vectoriel des polynômes de degré inférieur ou égal à $N - 1$. Le degré des éléments de D est donc au plus $N - 1$.

Définition 5.1.3 — *On appelle indice de nil-potence de Q_0, l'entier N_0 égal au maximum des degrés des éléments de D.*

Proposition 5.1.4 — *N_0 est le plus grand entier N tel que $\mathbf{m}_0^N \not\subset Q_0$.*

Preuve. Soit $\Lambda \in D$ de degré N_0, Λ possède un monôme de degré d, $\alpha\,\mathbf{d}^{\mathbf{a}}$ où α est un scalaire non nul. Alors $\Lambda(\mathbf{x}^{\mathbf{a}}) \neq 0$, donc $\mathbf{m}_0^{N_0} \not\subset Q_0$. Par contre, pour tout monôme $m = \mathbf{x}^{\mathbf{a}}$ tel que $|\mathbf{a}| > N_0$ et tout $\Lambda \in D$, $\Lambda(m) = 0$, donc $\mathbf{x}^{\mathbf{a}} \in Q_0$, ce qui implique que $\mathbf{m}_0^{N_0+1} \subset Q_0$.

Théorème 5.1.5 — *Soient $I = (p_1, \ldots, p_s)$ un idéal de R ayant une composante isolée Q_0 en 0, et $q_1, \ldots, q_s$ des polynômes homogènes de degrés $> N_0+1$. Alors l'idéal*

$$\tilde{I} = (p_1 + q_1, \ldots, p_s + q_s)$$

a pour composante $\mathbf{m}_0$-primaire isolée Q_0.

Preuve. Notons $\tilde{p}_i = p_i + q_i, 1 \leq i \leq s$. Un élément $\Lambda \in \mathbb{K}[\mathbf{d}]$ est dans $I^{\perp}$ (resp. $\tilde{I}^{\perp}$) si et seulement si pour tout $\mathbf{a} \in \mathbb{N}^n$, $\Lambda(\mathbf{x}^{\mathbf{a}}\,p_i) = 0$ (resp. $\Lambda(\mathbf{x}^{\mathbf{a}}\,\tilde{p}_i) = 0$), $1 \leq i \leq s$. Or si le degré de Λ est $\leq N_0 + 1$,

$$\Lambda(\mathbf{x}^{\mathbf{a}}\,\tilde{p}_i) = \Lambda(\mathbf{x}^{\mathbf{a}}\,p_i).$$

Donc $I^\perp$ et $\tilde{I}^\perp$ coïncident jusqu'au degré N_0+1. Par conséquent, il n'existe pas d'éléments de degré exactement N_0+1 dans $\tilde{I}^\perp$ (car N_0 est l'indice de nil-potence de Q_0), et a fortiori pas d'éléments de degré $> N_0$ (car $\tilde{I}^\perp$ est stable par dérivation). Il en résulte que

$$\tilde{I}^\perp \cap \mathbb{K}[\mathbf{d}] = I^\perp \cap \mathbb{K}[\mathbf{d}].$$

La composante primaire Q_0 en 0 reste inchangée par déformation en degré suffisamment élevé. Ceci est vrai par translation en tout autre point ζ de $\mathbb{K}^n$.

5.2 Passage du système inverse au quotient

La construction d'une base de $I^\perp$ peut se faire facilement si on sait réduire tout polynôme en une forme normale modulo I, c'est-à-dire si on sait trouver son expression sur une base choisie de R/I. Comme nous allons le voir maintenant, le système inverse de I capture, d'une certaine manière, tous les éléments qui se réduisent à un même terme. Soient I un idéal de R et $(b_i)_i$ une base de $B = R/I$. Alors pour tout monôme $\mathbf{x}^\mathbf{a}$, il existe des scalaires uniques $(\lambda_{\mathbf{a},i})_i$ presque tous nuls, tels que

$$\mathbf{x}^\mathbf{a} - \sum_i \lambda_{\mathbf{a},i}\, b_i \in I.$$

Proposition 5.2.1 — *La famille $(\sum_{\mathbf{a}\in\mathbb{N}^n} \lambda_{\mathbf{a},i}\, \mathbf{d}^\mathbf{a})_i$ forme une base du système inverse de I.*

Preuve. Soit $(\Lambda_i)_i \subset \widehat{B}$ (que l'on identifie avec $I^\perp$) la base duale de $(b_i)_i \subset B$. Les éléments Λ_i s'écrivent sous la forme

$$\Lambda_i = \sum_{\mathbf{a}\in\mathbb{N}^n} \mu_{\mathbf{a},i}\, \mathbf{d}^\mathbf{a}, \mu_{\mathbf{a},i} \in \mathbb{K}.$$

et comme $\Lambda_i(b_j) = \delta_{i,j}$ où $\delta_{i,j}$ est le symbole de Kronecker,

$$\Lambda_i(\mathbf{x}^\mathbf{a}) = \sum_j \lambda_{\mathbf{a},j}\, \Lambda_i(b_j) = \lambda_i = (\sum_\mathbf{b} \mu_{\mathbf{b},i}\, \mathbf{d}^\mathbf{b})\,(\mathbf{x}^\mathbf{a}) = \mu_{\mathbf{a},i}.$$

Alors

$$\mu_{\mathbf{a},i} = \lambda_{\mathbf{a},i}.$$

Cette construction peut se reformuler de la façon suivante. Soit

$$\Delta = \sum_{\mathbf{a}\in\mathbb{N}^n} \mathbf{x}^\mathbf{a} \otimes \mathbf{d}^\mathbf{a}\ ,$$

l'élément diagonal de $\mathbb{K}[[\mathbf{x},\mathbf{d}]] \simeq \mathbb{K}[[\mathbf{x}]] \bigotimes \mathbb{K}[[\mathbf{d}]]$. Pour $p \in R$, on définit $\Delta(p) = \sum_{\mathbf{a}\in\mathbb{N}^n} \mathbf{x}^{\mathbf{a}}\mathbf{d}^{\mathbf{a}}(p)$, et pour $\lambda \in \widehat{R}, \lambda(\Delta) = \sum_{\mathbf{a}\in\mathbb{N}^n} \lambda(\mathbf{x}^{\mathbf{a}})\mathbf{d}^{\mathbf{a}}$. On remarque que

$$\Delta(p) = p,\ \lambda(\Delta) = \lambda.$$

Nous nous plaçons dans le cas où B est un $\mathbb{K}$-espace vectoriel de dimension finie.

Proposition 5.2.2 — *Les propriétés suivantes sont équivalentes*

(1) Il existe une décomposition de Δ sous la forme

$$\Delta = \sum_{i=1}^{\infty} a_i \otimes \beta_i$$

où (a_i) est une base et (β_i) est une famille linéairement indépendante sur $\mathbb{K}$ avec
- *$a_j \in I$ pour $j > \mu$,*
- *$\beta_i \in I^\perp$ pour $1 \leq i \leq \mu$.*

(2) $(a_i)_{1\leq i\leq\mu}$ est une base de B .
(3) $(\beta_i)_{1\leq i\leq\mu}$ est une base de $I^\perp$.

Si ces points sont satisfaits, alors $(a_i)_{1\leq i\leq\mu}$ et $(\beta_i)_{1\leq i\leq\mu}$ sont des bases duales .

Preuve. Supposons que Δ se redécompose suivant *1*. Puisque (a_i) engendre $\mathbb{K}[\mathbf{x}]$ (comme $\mathbb{K}$-espace vectoriel), $(a_i)_{1\leq i\leq\mu}$ engendre B.

Nous avons de plus $\beta_i(\Delta) = \beta_i = \sum_{j=1}^{\mu} \beta_i(a_j)\,\beta_j$; comme $(\beta_i)_i$ est une famille libre, ceci implique

$$\beta_i(a_j) = \delta_{i,j} \text{ pour } 1 \leq i,j \leq \mu. \tag{2}$$

Soit $\lambda_l \in \mathbb{K}$ tel que $\sum_{l=1}^{\mu} \lambda_l\, a_l \in I$, alors $\beta_i(\sum_l \lambda_l\, a_l) = 0$ et $\lambda_l = 0$ pour $1 \leq l \leq \mu$. Donc $(a_i)_{1\leq i\leq\mu}$ est une famille libre de B, comme elle est génératrice, c'est une base.

La relation (2) montre que $(\beta_i)_{1\leq i\leq\mu}$ est la base duale de $(a_i)_{1\leq i\leq\mu}$ et les points *2* et *3* sont vérifiés.

Supposons maintenant que *2* soit vérifié. Alors $(a_i)_{1\leq i\leq\mu}$ est aussi libre dans $\mathbb{K}[\mathbf{x}]$ et il existe une famille libre $(a_j)_{j>\mu} \subset I$ telle que l'espace vectoriel engendré par $(a_1,\ldots,a_\mu,a_{\mu+1},\ldots)$ soit une base de $\mathbb{K}[\mathbf{x}]$. Ceci permet de réécrire

Δ sous la forme

$$\Delta = \sum_{i=1}^{\mu} a_i \otimes \beta'_i + \sum_{j>\mu} a_j \otimes \beta'_j,$$

avec les (β'_j) linéairement indépendants. En effet supposons que $\sum_i \nu_i \beta'_i = 0$ avec les ν_i presque tous nuls. Soit $(\beta_i)_i$ la base duale de $(a_i)_i$,

$$\sum_i \nu_i \beta_i = \sum_i \nu_i \beta_i\,(\Delta) = \sum_i \nu_i \beta_i (\sum_j a_j \otimes \beta'_j) = 0.$$

Alors, comme $(\beta_i)_i$ est une base, pour tout i, $\nu_i = 0$.

$(\beta_i)_{1\le i\le\mu} \subset I^{\perp}$ est la base duale de $(a_i)_{1\le i\le\mu}$ donc on a :

$$\beta_i(\Delta) = \beta_i = \sum_{j=1}^{\mu} \beta_i(a_j)\beta'_j = \beta'_i \ , \quad 1 \le i \le \mu$$

ce qui montre *1*.

Si *3* est vérifié, alors on choisit pour (a_i) la base duale de (β_i) et on utilise le *2*.

Cette propriété peut être utilisée dans les deux sens. Si nous savons réduire à une forme normale tout monôme (par exemple, en utilisant une base de Gröbner) alors nous pouvons calculer une base du système inverse (au moins en tronquant à un certain degré). Inversement, si nous connaissons le système inverse alors nous pouvons construire les générateurs de l'idéal. Nous illustrons ceci sur deux exemples très simples, où les éléments du système inverse sont des polynômes.

Exemple 1 :

Nous considérons l'idéal I engendré par

$$p_1 :\ = y - x^2,\ p_2 :\ = y^2 - x^3,\ p_3 :\ = x^4.$$

Pour construire $I^{\perp}$, nous utilisons les relations $x\,y \equiv x^3 \equiv y^2 \equiv x^4 \equiv 0$ modulo I et $x^2 = y - p_1$. On réécrit Δ sous la forme

$$\begin{aligned}\Delta &= \mathbf{d}^0 + x\,\mathbf{d}_1 + y\,\mathbf{d}_2 + x^2\,\mathbf{d}_1^2 + \cdots\\ &= \mathbf{d}^0 + x\,\mathbf{d}_1 + y\,\mathbf{d}_2 + (y - p_1)\,\mathbf{d}_1^2 + \cdots\\ &= \mathbf{d}^0 + x\,\mathbf{d}_1 + y\,(\mathbf{d}_2 + \mathbf{d}_1^2) + \cdots\end{aligned}$$

les points $\ldots$ désignant des éléments de $I\,\widehat{R} = I \otimes \widehat{R}$ c'est-à-dire $x\,y\,\mathbf{d}_1, y^2\,\mathbf{d}_2$ et tout les termes faisant intervenir une puissance de $\mathbf{x}$ supérieure ou égale à 3. Les produits tensoriels sont implicites dans cette notation. Donc, d'après la proposition 5.2.2, $I^{\perp} = \langle \mathbf{d}_2 + \mathbf{d}_1^2, \mathbf{d}_1, \mathbf{d}^0 \rangle = \langle\!\langle \mathbf{d}_2 + \mathbf{d}_1^2 \rangle\!\rangle$.

Exemple 2 :
Considérons maintenant le système inverse

$$\langle\!\langle \mathbf{d}_1^2 + \mathbf{d}_2^2 + \mathbf{d}_1\mathbf{d}_2 \rangle\!\rangle$$

qui est engendré (comme $\mathbb{K}$-espace vectoriel) par les éléments $\Lambda_2 = \mathbf{d}_1^2 + \mathbf{d}_2^2 + \mathbf{d}_1\mathbf{d}_2, \Lambda_1 = \mathbf{d}_1 + \mathbf{d}_2, \Lambda_0 = \mathbf{d}^0$.

Nous voulons construire l'idéal I, on réécrit donc Δ sous la forme

$$\begin{aligned}
\Delta &= \mathbf{d}^0 + x\,\mathbf{d}_1 + y\,\mathbf{d}_2 + x^2\,\mathbf{d}_1^2 + x\,y\,\mathbf{d}_1\,\mathbf{d}_2 + y^2\,\mathbf{d}_2^2 + \cdots \\
&= \Lambda_0 + x\,(\Lambda_1 - \mathbf{d}_2) + y\,\mathbf{d}_2 + x^2\,(\Lambda_2 - \mathbf{d}_2^2 - \mathbf{d}_1\,\mathbf{d}_2) + x\,y\,\mathbf{d}_1\,\mathbf{d}_2 + y^2\,\mathbf{d}_2^2 + \cdots \\
&= \underbrace{\Lambda_0 + x\,\Lambda_1 + x^2\,\Lambda_2}_{(1)} + \underbrace{(y-x)\,\mathbf{d}_2 + (x\,y - x^2)\,\mathbf{d}_1\,\mathbf{d}_2 + (y^2 - x^2)\,\mathbf{d}_2^2 + \cdots}_{(2)}
\end{aligned}$$

D'après la proposition 5.2.2 les éléments de $\mathbb{K}[x_1, x_2]$ apparaissant au dessus de l'accolade (1), $(a_0 = 1, a_1 = x, a_2 = x^2)$, forment une base de $B = R/I$ et ceux qui apparaissent au dessus de l'accolade (2) ainsi que ceux de degré supérieur ou égal à 3, $(a_i)_{i\geq 3}$, appartiennent à I. Comme la réunion $(a_i)_i \in \mathbb{N}$ de ces éléments est une base de R, $(a_i)_{i\geq 3}$ engendre l'idéal I. Or $(x\,y - x^2) = x\,(y-x)$ et $y^2 - x^2 = (x+y)\,(y-x)$ donc

$$I = \langle\!\langle \mathbf{d}_1^2 + \mathbf{d}_2^2 + \mathbf{d}_1\mathbf{d}_2 \rangle\!\rangle^{\perp} = (x-y) + (x,y)^3,$$

et que $(1, x, x^2)$ est une base du quotient B.

Dans le cas où B est un espace vectoriel de dimension finie, la structure multiplicative du quotient peut se retrouver directement, comme le montre la proposition suivante. Soit $(\beta_1, \ldots, \beta_\mu)$ une base de l'espace vectoriel $I^{\perp}$. Pour tout $k \in \{1, \ldots, n\}$,

$$\delta_k(\beta_i) = \sum_{j=1}^{\mu} \lambda_{i,j}^k \beta_j \quad , \quad \lambda_{i,j}^k \in \mathbb{K}.$$

Notons U_k la matrice $(\lambda_{i,j}^k)_{1\leq i,j\leq\mu}$.

Proposition 5.2.3 — *Les matrices $U_k, 1 \leq k \leq n$, sont les matrices de multiplication par x_k dans B dans la base duale de $(\beta_i)_{1\leq i\leq\mu}$.*

Preuve. Soit $(b_1, \ldots, b_\mu)$ la base duale de $(\beta_1, \ldots, \beta_\mu)$ dans B. Le coefficient d'indices i, j de la matrice de multiplication par x_k dans B dans la base (b_j) est donné par

$$\beta_i(x_k\, b_j) = (x_k \cdot \beta_i)(b_j)$$
$$= \delta_k(\beta_i)(b_j) = \sum_{l=1}^{\mu} \lambda_{i,l}^k \beta_l(b_j) = \lambda_{i,j}^k.$$

5.3 Construction du système inverse dans le cas local

Soit I un idéal de R et $D = I^{\perp} \cap \mathbb{K}[\mathbf{d}]$ le système inverse de la composante primaire isolée au point $\zeta = 0$.

Définition 5.3.1 — *$\mathbb{K}[\mathbf{d}]_d$ est l'ensemble des polynômes en $\mathbf{d}$ de degré au plus d et $D \cap \mathbb{K}[\mathbf{d}]_d$ sera noté D_d et nous noterons s_d sa dimension.*

Nous allons voir comment on peut construire D_d à partir de D_{d-1}. Comme on sait que les éléments de I s'annulent en 0, D_0 est engendré par $\mathbf{d}^0$ ($\mathbf{d}^0$ est la forme linéaire telle que $\mathbf{d}^0(p) = p(0)$), nous pourrons alors construire tous les D_j par récurrence.

Notons pour $p \in \mathbb{K}[\mathbf{d}], p_{|\mathbf{d}_i=0} = p(\mathbf{d}_1, \ldots, \mathbf{d}_{i-1}, 0, \mathbf{d}_{i+1}, \ldots, \mathbf{d}_n)$.

Définition 5.3.2 — *On appelle i-primitive (sans terme constant) de $p \in \mathbb{K}[\mathbf{d}]$, le polynôme q, noté $\int_i p$, tel que $\delta_i q = p$ et $q_{|\mathbf{d}_i=0} = 0$.*

remarque: En fait on a $\int_i p = \mathbf{d}_i\, p$ ce qui est très simple pour les calculs.

5.3.1 Passage de D_{d-1} à D_d

Comme $D_{d-1} \subset D_d$ et que D_d est stable par dérivation, les éléments de D_{d-1} sont les dérivées des éléments de D_d; donc pour obtenir les éléments de D_d, l'idée est d'intégrer les éléments de D_{d-1}. La construction de D_d à partir de D_{d-1} est basée sur le théorème suivant.

Théorème 5.3.3 — *Supposons que l'idéal I soit engendré par $p_1, \ldots, p_m$ et $d > 1$. Soit $(\beta_1, \ldots, \beta_s)$ une base de D_{d-1}. Les éléments de D_d sans terme constant sont les Λ de la forme*

$$\Lambda = \sum_{j=1}^{s} \lambda_j^1 \int_1 \beta_j|_{\mathbf{d}_2=0,\ldots,\mathbf{d}_n=0} + \sum_{j=1}^{s} \lambda_j^2 \int_2 \beta_j|_{\mathbf{d}_3=0,\ldots,\mathbf{d}_n=0} + \cdots + \sum_{j=1}^{s} \lambda_j^n \int_n \beta_j \quad , \quad \lambda_j^k \in \mathbb{K} \ , \tag{3}$$

tels que

(1) $\sum_{j=1}^{s} \lambda_j^k \delta_l \beta_j - \sum_{j=1}^{s} \lambda_j^l \delta_k \beta_j = 0$ *pour* $1 \leq k < l \leq n$,
(2) $\Lambda(p_i) = 0$ *pour* $1 \leq i \leq m$.

voir [3].

La condition *1* traduit seulement le fait que les *dérivations* $\delta_i, 1 \leq i \leq n$, *commutent ou de manière équivalente que la multiplication dans* B *est commutative.*

5.3.2 Exemple

On considère le point isolé $0 \in \mathbb{K}^2$ du système

$$p_1 = 2\,x_1\,x_2^2 + 5\,x_1^4,\ \ p_2 = 2\,{x_1}^2\,x_2 + 5\,x_2^4.$$

Pour tout $i, j \in \mathbb{N}$, on note $\mathbf{d}_i^j = \frac{1}{j!}\,\mathrm{d}_i^j$. On vérifie facilement que $I^\perp$ contient $1, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_1^2, \mathbf{d}_1\mathbf{d}_2, \mathbf{d}_2^2, \mathbf{d}_1^3, \mathbf{d}_2^3$. Pour trouver les autres éléments de D, on intègre ceux-ci suivant la formule (3) en ne gardant que les éléments qui apportent de nouveaux termes

$$\Lambda = \lambda_1\,\mathbf{d}_1^4 + \lambda_2\,\mathbf{d}_1^2\mathbf{d}_2 + \lambda_3\,\mathbf{d}_1\mathbf{d}_2^2 + \lambda_4\,\mathbf{d}_2^4 + \lambda_5\,\mathbf{d}_1^3\mathbf{d}_2 + \lambda_6\,\mathbf{d}_1\mathbf{d}_2^3,\ \ \lambda_i \in \mathbb{K}.$$

Les conditions $\Lambda(p_1) = \Lambda(p_2) = 0$, entraînent que

$$\Lambda = \lambda_1(2\,\mathbf{d}_1^4 - 5\,\mathbf{d}_1\mathbf{d}_2^2) + \lambda_2(2\,\mathbf{d}_2^4 - 5\,\mathbf{d}_1^2\mathbf{d}_2)$$

Un nouvel élément de $I^\perp$ sera (d'après le théorème précédent) de la forme $\Lambda = \lambda_1\mathbf{d}_1^5 + \lambda_2(2\,\mathbf{d}_1^4\mathbf{d}_2 - 5\,\mathbf{d}_1\mathbf{d}_2^3) + \lambda_3(2\,\mathbf{d}_2^5 - 5\,\mathbf{d}_1^2\mathbf{d}_2^2)$ et ses dérivées doivent être dans l'espace vectoriel engendré par les éléments précédents, ce qui impose que

$$\Lambda = \lambda\,(5\,\mathbf{d}_1^2\mathbf{d}_2^2 - 2\,\mathbf{d}_1^5 - 2\,\mathbf{d}_2^5),\ \ \lambda \in \mathbb{K}.$$

Une nouvelle intégration ne fournit pas d'autre élément dans D qui est alors engendré par

$$1, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_1^2, \mathbf{d}_1\mathbf{d}_2, \mathbf{d}_2^2, \mathbf{d}_1^3, \mathbf{d}_2^3,$$
$$2\,\mathbf{d}_1^4 - 5\,\mathbf{d}_1\mathbf{d}_2^2, 2\,\mathbf{d}_2^4 - 5\,\mathbf{d}_1^2\mathbf{d}_2, 5\,\mathbf{d}_1^2\mathbf{d}_2^2 - 2\,\mathbf{d}_1^5 - 2\,\mathbf{d}_2^5.$$

Le point 0 est donc de multiplicité 11.

5.3.3 *Description de l'algorithme*

Ceci nous conduit naturellement à un algorithme qui construit étape par étape, les générateurs de D. Nous obtiendrons par la même occasion la structure du quotient, c'est-à-dire les matrices de multiplications par les variables x_l ou les matrices de dérivations par δ_l, $1 \leq l \leq n$.

Entrée : $(p_1, \ldots, p_m) \in R^m$ et $\zeta \in \mathbb{K}^n$ tels que
$I = (p_1, \ldots, p_m)$ a une composante $\mathbf{m}_\zeta$-primaire
isolée Q_ζ.
Sortie : Une base de $Q_\zeta^\perp$ dans $\mathbb{K}[\mathbf{d}]$ après translation en 0 et les matrices de multiplication par $x_k - \zeta_k$ dans B après translation en 0.

▷ $D_0 : = 1$; d := 0; s_0 := 1; test := vrai;
  Pour k de 1 à n faire $U_k[1]$:= [0];
▷ Tant que teste faire
  1) S := système d'équations 1,2 en λ_j^k décrit dans le théorème précédent ;
  2) résoudre le système S;
  3) S'il n'y a pas de nouvelle solution alors teste := faux
    sinon
      soit $(\gamma_1, \ldots, \gamma_s)$ une base des nouvelles solutions
        telle que $\delta_k(\delta_i) = \sum_{j=1}^{s_d} \lambda_{j,s_d+i}\, \beta_j$;
      $s_{d+1} : = s_d + s$;
      $D_{d+1} : = D_d, \delta_1, \ldots, \delta_s = \beta_1, \ldots, \beta_{s_{d+1}}$;
      Pour k de 1 à n faire
        Pour i de $s_d + 1$ à s_{d+1} faire
          $U_k[i] : = [\lambda_{1,i}, \ldots, \lambda_{s_d,i}]$;
      $d : = d + 1$;
▷ D_d et U^k pour $1 \leq k \leq n$;

Pour simplifier les calculs, on commence donc par faire une translation qui déplace le point ζ en zéro. Les polynômes et les matrices trouvés seront donc ceux correspondant à la variété résultant de cette translation.

En pratique, si le minimum des ordres des $p_1, \ldots, p_m$ est ord, on sait que $\mathbb{K}[\mathbf{d}]_{ord-1}$ est inclus dans $Q^\perp$. On choisit la base canonique de $\mathbb{K}[\mathbf{d}]_{ord-1}$ et on commence donc les itérations en intégrant uniquement $D_{ord-1} \setminus D_{ord-2}$ plutôt que de commencer avec $\mathbf{d}^0$. On n'initialise alors les matrices U_k que lors de la première intégration (la partie des matrices de multiplication qui n'est pas dans les matrices U_k est alors évidente à écrire). Le gain de temps ainsi obtenu peut être assez conséquent si l'ordre est important car cela permet de manipuler des matrices de plus petites dimensions.

On prendra bien sur soin de ne construire à chaque étape que la nouvelle partie des matrices U_k et des matrices A_k exprimant la propriété 2 (c'est ce qui sera appelé les *nouvelles contraintes*).

Une étape importante de cet algorithme est le point (2) que nous allons détailler. Supposons que nous soyons au rang d et que nous avons calculé une base $\beta_1, \ldots, \beta_{s_d}$ de D_d. Posons $U_k = (u_{i,j}^k)_{1 \leq i,j \leq s_d}$ tel que

$$d_{\partial_k}(\beta_j) = \sum_{i=1}^{s_d} u_{i,j}^k \, \beta_i \quad , \quad 1 \leq j \leq s_d.$$

Soient $v_l = (\lambda_1^l, \ldots, \lambda_{s_d}^l), 1 \leq l \leq n$, des vecteurs tels que $V = [v_1, \ldots, v_n]$ soit une solution du système *1, 2* du théorème (5.3.3). Les équations *1* se réécrivent sous la forme

$$U_k \, v_l - U_l \, v_k = 0, \ 1 \leq k < l \leq n.$$

Les équations *2* correspondent à

$$[A_1, \ldots, A_n]. \, V = 0,$$

où les matrices $A_1, \ldots, A_n$ sont de taille $m \times s_d$. Soit $\Lambda_{i,j} = \textstyle\int_i \beta_j|_{\mathbf{d}_{i+1}=0,\ldots,\mathbf{d}_n=0}$, on a alors

$$A_i = \begin{bmatrix} \Lambda_{i,1}(p_1) & \cdots & \Lambda_{i,n}(p_1) \\ \vdots & \vdots & \vdots \\ \Lambda_{i,1}(p_m) & \cdots & \Lambda_{i,n}(p_m) \end{bmatrix}.$$

Le système général est donc de la forme

$$\begin{bmatrix} U_n & & & & -U_1 \\ & U_n & & & -U_2 \\ & & \ddots & & \vdots \\ & & & U_n & -U_{n-1} \\ U_{n-1} & & & -U_1 & \\ & \ddots & & \vdots & \\ & & U_{n-1} & -U_{n-2} & \\ \vdots & \vdots & \vdots & & \\ U_2 & -U_1 & & & \\ A_1 & \cdots & \cdots & \cdots & A_n \end{bmatrix} . \, V = 0 \ ,$$

où les blancs désignent des 0. C'est un système linéaire de taille $(\frac{1}{2} \, n \, (n-1) \, s_d + m) \times n \, s_d$. On remarque cependant que, comme les matrices U_k expriment les dérivées des vecteurs d'une base de D_d sur cette même base, il est inutile de faire apparaître dans la matrice les dernières lignes qui sont forcément nulle. En effet elles correspondent aux polynômes de degré d qui ne peuvent pas apparaître dans la décomposition d'une dérivée d'un polynôme de degré au plus d. Les matrices U_k seront donc en pratique de taille $s_{d-1} \times s_d$. La matrice GM sera alors de taille $(\frac{1}{2}\, n\,(n-1)\, s_{d-1} + m) \times n\, s_d$.

Si, pour résoudre, on ne triangularise à chaque étape que la nouvelle partie de la matrice, et qu'on ne cherche que les nouvelles solutions on a la propriété suivante :

Proposition 5.3.4 — *Le nombre total d'opérations pour l'étape (2) dans tout l'algorithme est majoré par* $\mathcal{O}(n^2\mu^3 + m\,\mu^3)$ *où* $\mu - 1 = s_\nu - s_0 = (s_1 - s_0) + \cdots + (s_\nu - s_{\nu-1})$.
Voir [3].

6 Les différentes implémentations

6.1 Solutions grossières

Une première façon de faire est de ne pas tenir compte des éléments que l'on a déjà trouvés et de résoudre à chaque fois entièrement le système. On retrouve alors à chaque fois les éléments que l'on connaît déjà : $\mathfrak{A}$. Comme les nouveaux polynômes trouvés sont nécessairement de degré d plus grand que celui des précédents, on peut tout de suite éliminer les polynômes de degré inférieur à d. On doit alors construire, à partir des solutions trouvées, une base $\mathfrak{N}$ des solutions telle que *vect* $(\mathfrak{A})$ et *vect* $(\mathfrak{N})$ soient en somme directe. Voici comment on procède :

1 On écrit les anciennes solutions $(\mathfrak{A})$ et les candidats à être de nouvelles solutions sur une base des monômes qui apparaissent et on met les anciennes solutions ainsi décomposées dans une matrice M.
2 On calcule ensuite le rang de la matrice M' obtenue en accolant un candidat à M.
3 Si ce rang est égal à celui de M, on passe au candidat suivant. Sinon le rang de M est supérieur à celui de M, le candidat est effectivement une nouvelle solution, on le rajoute à la base des solutions et M' devient la nouvelle matrice M, on passe alors au candidat suivant.

C'est ce que fait la version 0 du programme décrite en annexe 1.

Pour optimiser cette méthode on peut, au fur et à mesure que l'on construit la base des solutions, triangulariser la matrice M. On évite ainsi de refaire la triangularisation à chaque fois et on calcule plus facilement le rang de la matrice M'. C'est ce qui est fait dans la version 1 de l'algorithme décrite en Annexe 2.

6.2 L'algorithme le plus efficace

Dans les implémentations précédentes, à chaque étape on recalculait des solutions que l'on connaissait déjà. Pour éviter cette grande perte de temps, on est amené à construire la matrice à résoudre un peu différemment. On veut en effet la triangulariser étape par étape, c'est-à-dire conserver le bénéfice de ce qui a déjà été fait et ne triangulariser que la partie qu' on lui rajoute.

L'idée est donc de construire la nouvelle matrice GM en utilisant la précédente de la façon suivante :

$$\begin{bmatrix} \begin{bmatrix} \text{ancienne matrice GM} \end{bmatrix} & \begin{bmatrix} \text{début des} \\ \text{nouvelles} \\ \text{contraintes} \end{bmatrix} \\ \begin{bmatrix} \text{zéros} \end{bmatrix} & \begin{bmatrix} \text{fin des} \\ \text{nouvelles} \\ \text{contraintes} \end{bmatrix} \end{bmatrix}$$

Pour cela il est beaucoup plus simple de placer les $A_1 \ldots A_n$ au dessus de la partie faisant intervenir les U_k. On initialise donc la matrice GM comme suit :

$$
\begin{bmatrix}
A_1 & \cdots & \cdots & \cdots & A_n \\
U_n & & & & -U_1 \\
& U_n & & & -U_2 \\
& & \ddots & & \vdots \\
& & & U_n & -U_{n-1} \\
U_{n-1} & & & -U_1 & \\
& \ddots & & \vdots & \\
& & U_{n-1} & -U_{n-2} & \\
\vdots & \vdots & \vdots & & \\
U_2 & -U_1 & & &
\end{bmatrix}
$$

Lorsque que l'on construit les nouvelles colonnes des matrices U_k, on construit deux sous matrices U_k^1 et U_k^2 qui contiennent les nouvelles colonnes de U_k. Dans U_k^1 on met les premières lignes des nouvelles colonnes de U_k. U_k^2 contient les dernières lignes(c'est-à-dire celles qui correspondent aux polynômes de degré $d-1$) des nouvelles colonnes de U_k. Les A_i^1 sont les nouvelles colonnes qui interviennent dans les A_i.

Si l'on reprend la forme de la matrice GM précédemment décrite,

$$
\begin{bmatrix}
\begin{bmatrix} \\ \quad oldGM \quad \\ \\ \end{bmatrix} & \begin{bmatrix} \\ \quad newGM_1 \quad \\ \\ \end{bmatrix} \\
\begin{bmatrix} \\ \quad z\acute{e}ro \quad \\ \\ \end{bmatrix} & \begin{bmatrix} \\ \quad newGM_2 \quad \\ \\ \end{bmatrix}
\end{bmatrix}
$$

les matrices $newGM_1$ et $newGM_2$ sont construites sur le modèle de l'initialisation de GM :

$$
newGM_1 = \begin{bmatrix}
A_1^1 & \cdots & \cdots & \cdots & A_n^1 \\
U_n^1 & & & & -U_1^1 \\
 & U_n^1 & & & -U_2^1 \\
 & & \ddots & & \vdots \\
 & & & U_n^1 & -U_{n-1}^1 \\
U_{n-1}^1 & & & -U_1^1 & \\
 & \ddots & & \vdots & \\
 & & U_{n-1}^1 & -U_{n-2}^1 & \\
\vdots & \vdots & \vdots & & \\
U_2^1 & -U_1^1 & & &
\end{bmatrix}
$$

et

$$
newGM_2 = \begin{bmatrix}
U_n^2 & & & & -U_1^2 \\
 & U_n^2 & & & -U_2^2 \\
 & & \ddots & & \vdots \\
 & & & U_n^2 & -U_{n-1}^2 \\
U_{n-1}^2 & & & -U_1^2 & \\
 & \ddots & & \vdots & \\
 & & U_{n-1}^2 & -U_{n-2}^2 & \\
\vdots & \vdots & \vdots & & \\
U_2^2 & -U_1^2 & & &
\end{bmatrix}
$$

En fait la matrice $newG_1$ réellement utilisée correspond à celle décrite ci-dessus sur laquelle on a fait des permutations de lignes pour que le système reste cohérent quand on accole $newG_1$ à $oldGM$. Comme on a triangularisé $oldGM$ (en faisant des opérations sur les lignes), il faut appliquer ces transformations à $newGM_1$. Il reste ensuite à triangulariser $newGM_2$ pour obtenir la nouvelle matrice GM.

Pour ne pas retrouver les solutions déjà connues, on ne va pas résoudre le système entièrement mais plutôt appliquer la méthode suivante :

En faisant des permutations de colonnes, on met la matrice GM sous la forme :

$$\begin{bmatrix} T & R \\ 0 & 0 \end{bmatrix},$$

où T est une matrice triangulaire supérieure inversible et R une matrice rectangulaire ayant le même nombre de lignes que T. Le nombre de colonnes de R est alors la dimension de l'espace des solutions. En effet, ce système $[T|R] \cdot [V] = 0$ est équivalent au système

$$[T][V_1] = -[R][V_2],$$

où

$$V = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

On fixe successivement $V_2 = (0, \ldots, 0, 1, 0, \ldots, 0)$, 1 étant à la i-ème place, et on calcule le vecteur V_1 correspondant. On obtient ainsi une base des solutions du système.

Le profil de la matrice GM après quelques itérations est en fait celui de la figure 1. On peut aussi voir ce profil dans l'exemple 6.2.1

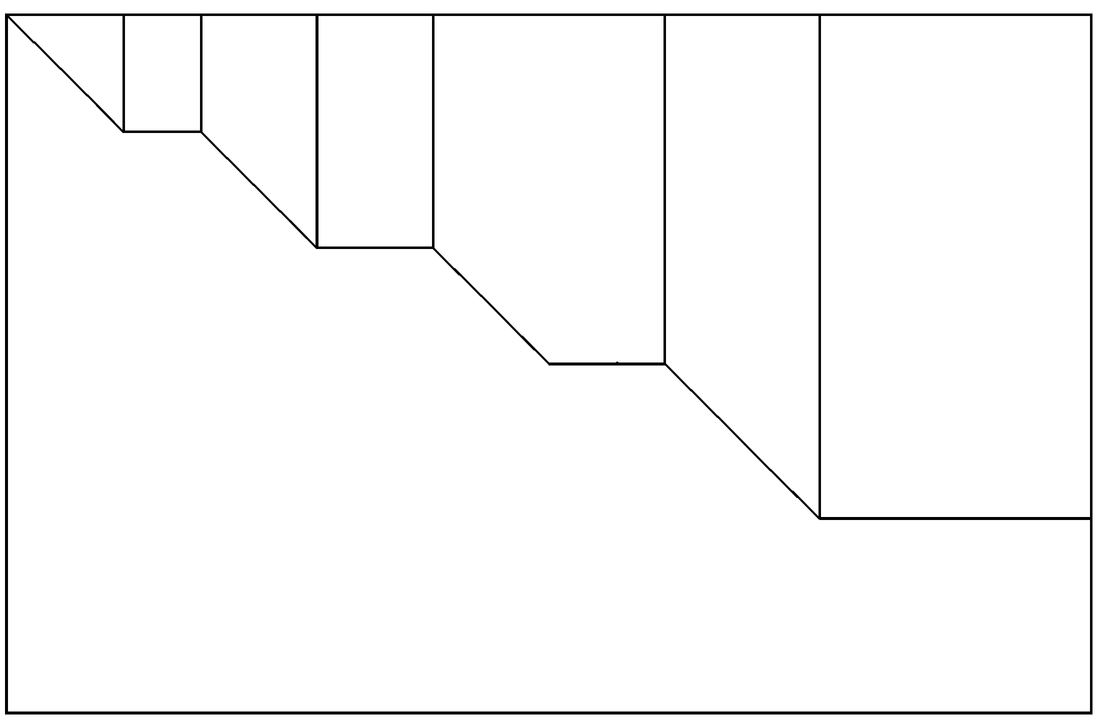


FIG. 1. matrice GM

La matrice T est obtenue en accolant les blocs trapèzoïdaux et, pour n'avoir que les nouvelles solutions, on ne prend comme matrice R que le dernier bloc rectangulaire. On résout le système

$$[T][V_1] = -[R][V_2]$$

et on insère dans les solutions des zéros correspondant aux autres blocs rectangulaires.

Comme cette matrice est élargie de $s_d - s_{d-1}$ à chaque étape de l'algorithme, on peut supposer que la triangulation de la matrice GM ci-dessus est faite jusqu'au rang s_{d-1}.

C'est cet algorithme qui permet d'obtenir une complexité majorée par $\mathcal{O}(n^2\mu^3 + m\,\mu^3)$ où $\mu - 1 = s_\nu - s_0 = (s_1 - s_0) + \cdots + (s_\nu - s_{\nu-1})$. . Voir([3]).

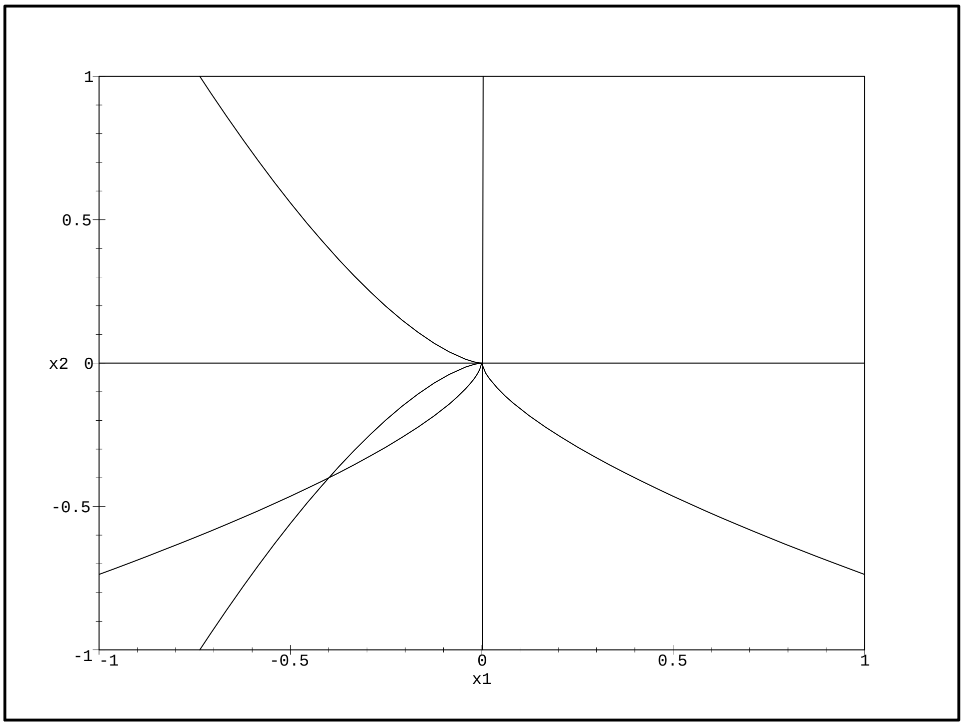
6.2.1 Un exemple de résultat

Nous avons traité avec l'algorithme décrit ci-dessus l'exemple résolu à la main dans le paragraphe 5.3.2. La base trouvée est :

$$\Big[[\,1\,], [d_2, d_1]\,, \Big[{d_1}^2, {d_2}^2, d_1\,d_2\Big]\,, \Big[{d_1}^3, {d_2}^3\Big]\,,$$
$$\Big[-\frac{5}{2}\,d_2\,{d_1}^2 + {d_2}^4, {d_1}^4 - \frac{5}{2}\,{d_2}^2\,d_1\Big]\,, \Big[{d_1}^5 - \frac{5}{2}\,{d_2}^2\,{d_1}^2 + {d_2}^5\Big]\Big]\,,$$

où les polynômes sont rangés dans des listes par degré croissant. On constate que l'espace vectoriel engendré par ces vecteurs est bien le même que celui engendré par la base construite à la main.

La figure suivante représente les courbes C_1 et C_2 définies respectivement par les polynômes p_1 et p_2. C_1 est le cusp inclus dans le demi-plan des x_1 négatifs et l'axe des x_2. C_2 est le cusp inclus dans le demi-plan des x_2 négatifs et l'axe des x_1.



Les matrices de multiplication dans le quotient sont:

$$U_1 = \begin{bmatrix} 0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&\frac{-5}{2}&0\\ 0&0&0&0&0&0&0&0&\frac{-5}{2}&0&0\\ 0&0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \qquad U_2 = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&\frac{-5}{2}&0&0\\ 0&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&\frac{-5}{2}&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

La dernière matrice GM construite est :

$$\begin{bmatrix}
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{5}{2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & \frac{-5}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-5}{2} & 0 & \frac{5}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

On voit bien les palliers des lignes 4, 6 et 9 et on remarque que la longueur des palliers, respectivement 2, 2 et 1, correspond bien au nombre de solutions trouvées en degré respectivement 3, 4 et 5.

6.3 Comparaison sur un exemple un peu plus consistant

On considère l'idéal de $\mathbb{K}[x_1, x_2]$ engendré par les polynômes suivants :

$$\begin{cases} p_1 = (2\,x_1\,x_2^2 + 5\,x_1^4)^3 \\ p_2 = (2\,{x_1}^2\,x_2 + 5\,x_2^4)^3 \end{cases}$$

p_1 et p_2 sont les polynômes de l'exemple 5.3.2 élevés à la puissance 3. La multiplicité de zéro est donc $11 \times 9 = 99$. Le tableau suivant donne les temps d'exécutions pour cet exemple avec les différents algorithmes.

algorithme	temps d'exécution
algorithme 0	1237 secondes
algorithme 1	1258 secondes
algorithme 2	643 secondes

Les polynômes p_1 et p_2 sont d'ordre 9 et l'on sait donc a priori que $\mathbb{K}[\mathbf{d}]_8$ est dans le système inverse. Comme le polynômes que l'orthogonal est de degré 20, on ne fait donc « que » 13 itérations de la résolution. En fait on pourrait s'attendre, compte tenu du nombre élevé de solutions et d'itérations, à un avantage beaucoup plus important en faveur de l'algorithme le plus efficace. Il n'en reste pas moins plus rapide que les deux autres et cela sera d'autant plus vrai que μ est grand. Sa lenteur relative s'explique par les rigidités de l'implémentation en Maple : quand on veut faire des manipulations sur les matrices, coller ou extraire une sous-matrice par exemple, Maple fait systématiquement des copies des matrices existentes, il ne fait pas l'opération sur place. Or, nous faisons beaucoup plus d'opérations dans le dernier algorithme. Une autre cause probable de cette lenteur est la procédure de triangularisation des matrices: dans les deux premières implémentations nous avons utilisé les fonctions Maple *linalg[linsolve]* et *linalg[kernel]* qui résolvent le système avec une procédure d'élimination de Gauss qui est écrite en C. Mais dans le dernier algorithme, nous avons du refaire la procédure de triangularisation pour y intégrer des permutations de colonnes. Cette procédure est beaucoup moins bonne que la procédure Maple à cause des permutations de colonnes, qui prennent un temps considérable quand la taille de la matrice est importante, mais aussi à cause de l'écriture de la procédure en Maple qui, par nature, donne une exécution moins rapide.

On peut constater sur le tableau suivant qui est un profil de l'exécution (réalisée sur une autre machine) que la procédure de triangularisation, bbtrig, prend un temps important. On remarque aussi que les trois procédures constUk, constGM, et constnewH2, où l'on fait principalement des manipulations matricielles, sont de très grandes consommatrices de temps. Le temps d'exécution était 1131 secondes le programmme principal (SI) prend quasiment tout le temps qui n'est pas passé dans les procédures dont on a fait le profil.

```
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
HG1                    1       12   50.130     6.74      39197944     7.94
bbtrig                 1       13   68.090     9.15      46572508     9.43
constAk                1       24    4.720      .63       2927260      .59
constGM2               1       12  210.370    28.28     134474348    27.24
constT                 1       11   33.621     4.52      19331824     3.91
constUk                1       24  144.070    19.37      91004596    18.44
constnewH2             1       11  147.060    19.77      97390356    19.73
goodvect               1       54   55.121     7.41      40944624     8.29
nderoul                1       54    3.340      .44       2423576      .49
resol                  1       54   27.130     3.64      19231684     3.89
---------------------------------------------------------------------------
total:                10      269  743.652   100.00     493498720   100.00
```

Comparons avec l'algorithme 1 dont le temps d'exécution est ici de 2017 secondes.

function	depth	calls	time	time%	bytes	bytes%
constAk	1	20	4.290	.22	2931516	.25
constGM	1	11	145.210	7.72	88215684	7.68
constUk	1	20	109.140	5.80	69045360	6.01
makebase	1	10	351.161	18.67	221644916	19.31
sol	1	11	43.420	2.30	23937356	2.08
trigrang	1	44	1226.759	65.25	741820728	64.64
total:	6	116	1879.980	100.00	1147595560	100.00

Ici c'est plutôt l'élimination des anciennes solutions par la triangularisation étape par étape de la matrice M (procédures makebase et triangrang) qui consomme beaucoup de temps. C'est le défaut que l'on a voulut corriger en implémentant l'algorithme 2.

Enfin voyons ce que donne l'algorithme 0 :

function	depth	calls	time	time%	bytes	bytes%
constAk	1	24	4.310	.25	3647752	.29
constGM	1	13	160.609	9.57	118574360	9.50
constUk	1	24	118.110	7.04	90603436	7.25
makebase	1	12	678.440	40.44	489456804	39.21
sol	1	13	715.840	42.67	545851636	43.73
total:	5	86	1677.309	100.00	1248133988	100.00

Le temps d'exécution (1839 secondes) est assez nettement inférieur à celui de l'algorithme 1 sur la même machine. Même si il est rare de trouver une telle différence en faveur de l'algorithme 0 Cela prouve que la triangularisation de la matrice M n'est pas assez efficace pour être valable pour les grandes valeurs.

L'exemple suivant où la dimension est 16 mais où le degré maximal des éléments du système inverse n'est que 4 (c'est-à-dire que l'on n'itère que 4 fois la résolution du système) est critique : le deuxième algorithme donne la solution un peu plus vite que le dernier. Le nombre d'itérations est en effet trop faible pour que le temps gagné sur la résolution compense celui perdu dans la triangularisation et dans les manipulations matricielles.

On prend les 5 polynômes de $\mathbb{K}[x_1, \ldots, x_5]$ suivants :

$$\begin{cases} p_1 = x_1^2 + x_1 + x_2 + x_3 + x_4 + x_5 \\ p_2 = x_2^2 + x_2 + x_1 + x_3 + x_4 + x_5 \\ p_3 = x_3^2 + x_3 + x_1 + x_2 + x_4 + x_5 \\ p_4 = x_4^2 + x_4 + x_1 + x_2 + x_3 + x_5 \\ p_5 = x_5^2 + x_5 + x_1 + x_2 + x_3 + x_4 \end{cases}$$

Et on trouve un résultat difficilement apréhendable. On imagine le travail que représenterait le calcul à la main, même aidé d'un logiciel de calcul formel...

$$\Big[[1], [-d_1 + d_4, -d_1 + d_2, -d_1 + d_3, -d_1 + d_5], \Big[$$
$$d_2\, d_1 + d_3\, d_1 - d_3\, d_2 - d_4\, d_1 - d_5\, d_1 + d_5\, d_4,$$
$$d_3\, d_1 - d_3\, d_2 - d_5\, d_1 + d_5\, d_2, d_2\, d_1 - d_3\, d_2 - d_5\, d_1 + d_5\, d_3, -d_1 + {d_1}^2$$
$$+ \frac{1}{2}\, d_2\, d_1 + {d_2}^2 + \frac{1}{2}\, d_3\, d_1 - \frac{3}{2}\, d_3\, d_2 + {d_3}^2 - d_4\, d_1 + {d_4}^2 - d_5\, d_1 + {d_5}^2,$$
$$d_3\, d_1 - d_3\, d_2 - d_4\, d_1 + d_4\, d_2, d_2\, d_1 - d_3\, d_2 - d_4\, d_1 + d_4\, d_3\Big], \Big[\%3 - \%2$$
$$+ d_4\, d_1 - {d_2}^2\, d_1 - {d_4}^2\, d_1 - d_4\, {d_1}^2 - d_4\, {d_2}^2 - d_4\, {d_3}^2 + {d_4}^2\, d_2 + {d_4}^2\, d_3$$
$$- {d_3}^2\, d_1 + d_3\, {d_1}^2 + d_3\, {d_2}^2 + d_2\, {d_1}^2 + {d_3}^2\, d_2 - {d_5}^2\, d_1 + {d_5}^2\, d_2 - d_3\, d_2$$
$$+ {d_2}^3 - {d_1}^3 + {d_3}^3 - {d_4}^3 + {d_5}^2\, d_3 - {d_5}^2\, d_4 - \frac{1}{2}\, d_4\, d_3\, d_2 + \frac{1}{2}\, \%1$$
$$- \frac{1}{2}\, d_3\, d_2\, d_1 + \frac{1}{2}\, d_4\, d_2\, d_1, \frac{1}{2}\, d_4\, d_2\, d_1 - d_5\, d_1 + d_4\, d_1 - d_4\, {d_1}^2 - d_4\, {d_2}^2$$
$$- d_4\, {d_3}^2 + \frac{1}{2}\, \%1 + \frac{1}{2}\, d_4\, d_3\, d_2 + d_5\, {d_4}^2 - {d_5}^2\, d_4 + d_5\, {d_1}^2 + d_5\, {d_2}^2$$
$$+ d_5\, {d_3}^2 + {d_5}^3 - {d_4}^3 - \frac{1}{2}\, d_5\, d_3\, d_1 - \frac{1}{2}\, \%2 - \frac{1}{2}\, d_5\, d_2\, d_1, 2\, {d_1}^3 + 2\, {d_2}^2\, d_1$$
$$- 2\, d_2\, {d_1}^2 - 2\, {d_2}^3 - 2\, d_3\, d_1 + 2\, d_3\, d_2 + 2\, {d_3}^2\, d_1 - 2\, {d_3}^2\, d_2 - \%1$$
$$+ d_4\, d_3\, d_2 + 2\, {d_4}^2\, d_1 - 2\, {d_4}^2\, d_2 - d_5\, d_3\, d_1 + \%2 + 2\, {d_5}^2\, d_1 - 2\, {d_5}^2\, d_2$$
$$- \%3 + d_5\, d_4\, d_2, -2\, d_2\, d_1 + 2\, d_2\, {d_1}^2 + 2\, {d_2}^3 - d_3\, d_2\, d_1 + 2\, {d_3}^2\, d_2$$
$$+ 2\, d_4\, d_1 - 2\, d_4\, {d_1}^2 - 2\, d_4\, {d_2}^2 + \%1 - 2\, d_4\, {d_3}^2 - 2\, {d_4}^3 + 2\, {d_4}^2\, d_2$$
$$- d_5\, d_2\, d_1 + \%3 - 2\, {d_5}^2\, d_4 + 2\, {d_5}^2\, d_2 - \%2 + d_5\, d_4\, d_3\Big], \Big[- \frac{5}{3}\, \%3$$
$$+ \frac{1}{3}\, \%2 - \frac{4}{3}\, d_2\, d_1 - \frac{4}{3}\, d_3\, d_1 - \frac{2}{3}\, d_4\, d_1 + \frac{10}{3}\, {d_2}^2\, d_1 + \frac{10}{3}\, {d_4}^2\, d_1$$
$$+ \frac{2}{3}\, d_4\, {d_1}^2 + \frac{2}{3}\, d_4\, {d_2}^2 + \frac{2}{3}\, d_4\, {d_3}^2 + \frac{10}{3}\, {d_3}^2\, d_1 + \frac{10}{3}\, {d_5}^2\, d_1 + d_5\, {d_3}^2\, d_1$$
$$+ d_5\, d_3\, {d_1}^2 + {d_5}^2\, d_4\, d_3 + {d_5}^2\, d_4\, d_2 + d_5\, d_3\, {d_2}^2 + d_5\, d_2\, {d_1}^2 + d_5\, {d_3}^2\, d_2$$
$$+ {d_5}^2\, d_3\, d_1 + {d_5}^2\, d_2\, d_1 + {d_5}^2\, d_4\, d_1 + {d_5}^2\, d_3\, d_2 + d_5\, {d_2}^2\, d_1 + d_5\, {d_4}^2\, d_1$$
$$+ d_5\, d_4\, {d_1}^2 + d_5\, d_4\, {d_2}^2 + d_5\, d_4\, {d_3}^2 + d_5\, {d_4}^2\, d_2 + {d_3}^2\, d_2\, d_1 + d_3\, {d_2}^2\, d_1$$
$$+ d_3\, d_2\, {d_1}^2 + d_4\, {d_2}^2\, d_1 + d_4\, {d_3}^2\, d_1 + d_4\, d_3\, {d_1}^2 + d_4\, d_3\, {d_2}^2 + d_4\, d_2\, {d_1}^2$$
$$+ d_4\, {d_3}^2\, d_2 + {d_4}^2\, d_3\, d_1 + {d_4}^2\, d_2\, d_1 + {d_4}^2\, d_3\, d_2 + {d_5}^3\, d_1 + {d_5}^3\, d_4$$
$$+ {d_5}^3\, d_3 + {d_5}^3\, d_2 + \frac{4}{3}\, d_3\, d_2 - 4\, {d_3}^4 + d_5\, {d_4}^2\, d_3 + d_4\, {d_3}^3 - \frac{4}{3}\, d_5\, d_2\, d_1$$
$$+ \frac{10}{3}\, {d_1}^3 + \frac{2}{3}\, {d_4}^3 + \frac{2}{3}\, {d_5}^2\, d_4 + d_5\, {d_4}^3 + d_5\, {d_2}^3 + d_5\, {d_1}^3 + d_5\, {d_3}^3$$
$$+ d_3\, {d_2}^3 + d_3\, {d_1}^3 + {d_3}^3\, d_1 + {d_4}^3\, d_3 + d_2\, {d_1}^3 + {d_2}^3\, d_1 - 4\, {d_5}^2\, {d_4}^2$$
$$- 4\, {d_5}^2\, {d_1}^2 - 4\, {d_5}^2\, {d_2}^2 - 4\, {d_5}^2\, {d_3}^2 - 4\, {d_3}^2\, {d_1}^2 - 4\, {d_3}^2\, {d_2}^2 - 4\, {d_4}^2\, {d_1}^2$$
$$- 4\, {d_4}^2\, {d_2}^2 - 4\, {d_4}^2\, {d_3}^2 - 4\, {d_2}^2\, {d_1}^2 + {d_3}^3\, d_2 - \frac{4}{3}\, d_5\, d_3\, d_1 + d_4\, {d_2}^3 - 4\, {d_4}^4$$
$$+ {d_4}^3\, d_1 - \frac{5}{3}\, \%1 - \frac{4}{3}\, d_3\, d_2\, d_1 + {d_4}^3\, d_2 - \frac{5}{3}\, d_4\, d_2\, d_1 - 4\, {d_1}^4 - 4\, {d_2}^4$$
$$- 4\, {d_5}^4 + d_4\, {d_1}^3 - \frac{3}{2}\, d_5\, d_3\, d_2\, d_1 - \frac{3}{2}\, d_5\, d_4\, d_3\, d_2 - \frac{3}{2}\, d_5\, d_4\, d_2\, d_1$$
$$- \frac{3}{2}\, d_5\, d_4\, d_3\, d_1 - \frac{3}{2}\, d_4\, d_3\, d_2\, d_1\Big]\Big]$$
$$\%1 := d_4\, d_3\, d_1$$
$$\%2 := d_5\, d_3\, d_2$$
$$\%3 := d_5\, d_4\, d_1$$

Le tableau suivant donne les temps d'exécutions des trois algorithmes :

algorithme	temps d'exécution
algorithme 0	217 secondes
algorithme 1	83 secondes
algorithme 2	103 secondes

Voici le profil de l'exécution de l'algorithme 2 :

```
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
HG1                    1        4   13.970     8.57      11008712    10.15
bbtrig                 1        5   29.879    18.35      17956560    16.56
constAk                1       20    7.111     4.36       5621436     5.18
constGM                1        4   58.281    35.79      38013708    35.06
constT                 1        3    5.930     3.64       3642480     3.35
constUk                1       20    9.870     6.06       6784592     6.25
constnewH2             1        3   27.040    16.60      17313444    15.96
deroul                 1        4     .050      .03         49584      .04
goodvect               1       11    7.050     4.32       5296396     4.88
nderoul                1       11     .080      .04         53520      .04
resol                  1       11    3.560     2.18       2682552     2.47
---------------------------------------------------------------------------
total:                11       96  162.821   100.00     108422984   100.00
```

on remarque que le pourcentage de temps passé dans la procédure bbtrig est bien plus important que dans l'exemple précédent. Cela est dû au faible nombre d'itérations (4) et montre bien que cet algorithme n'est efficace que quand le nombre d'itérations est grand.

le tableau suivant donne le profile de l'algorithme 1.

```
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
constAk                1       20   11.020     6.66       7535008     7.54
constGM                1        4   53.490    32.35      31590216    31.62
constUk                1       20   10.920     6.60       6577264     6.58
makebase               1        4   31.670    19.15      19953216    19.97
sol                    1        5   12.980     7.85       7791232     7.80
trigrang               1       11   45.240    27.36      26430036    26.46
---------------------------------------------------------------------------
total:                 6       64  165.320   100.00      99876972   100.00
```

La triangularisatioin dans cet algorithme est réalisée par Maple en même temps qu'il resout dans la procédure sol. On note que le pourcentage de temps pris par sol est très inférieur à celui pris pas bbtrig dans l'algoritme 2. Cela montre bien que la procedure bbtrig est moins bonne que *linalg[kernel]*.

Et enfin le profil de l'algorithme 0

function	depth	calls	time	time%	bytes	bytes%
constAk	1	20	6.290	2.37	6766140	2.98
constGM	1	4	36.970	13.96	31793464	14.04
constUk	1	20	6.920	2.61	6617368	2.92
makebase	1	4	45.280	17.10	37056060	16.37
sol	1	5	169.200	63.93	144108032	63.66
total:	5	53	264.660	100.00	226341064	100.00

Ici on voit quand même que, dans des cas où la dimension n'est pas trop grande, l'algorithme 1 est beaucoup plus efficace que algorithme 0 : les manipulations matricielles se font sur des matrices de taille raisonnable et n'handicapent pas encore l'algorithme 1.

7 Conclusion

On a vu que l'implémentation de l'algorithme sensé être le plus efficace, même si elle permettait lorsqu'il y avait de gros calculs, un gain de temps important ne donnait pas entièrement satisfaction. L'implémentation que j'ai faite n'est peut-être pas optimale, et j'aurais probablement réussit à l'améliorer un peu si j'avais disposé de plus de temps, mais l'ordre serait resté le même. Même s'il est intéressant d'avoir, en Maple, un outil qui fonctionne correctement pour trouver la multiplicité de points épais et la structure de l'anneau local, Maple n'est pas le bon outil quand on veut optimiser un algorithme. Il faudrait, ou il aurait fallut, implémenter l'algorithme dans un véritable langage informatique par exemple C. Mais cela aurait pris aussi beaucoup plus de temps compte tenu du nombre de fonctions, équivalentes à celles disponible sous Maple, qu'il aurait fallu implémenter et je ne me sentais pas assez à l'aise en C au début du stage pour entreprendre ce travail.

Il aurait aussi été intéressant de regarder si on pouvait étendre le principe de l'algorithme au calcul du système inverse entier ou, du moins, des termes de bas degré des séries qui le composent. De façon expérimentale, en modifiant un peu l'algorithme 0 (le moins complexe), on pouvait en tout cas essayer de voir ce que cela donnait mais, là encore, le temps m'a manqué.

8 Remerciements

Je tiens à remercier Bernard Mourrain pour son aide patiente et ses nombreuses relectures de ce rapport.

Références

[1] M.F. Atiyah and I.G. MacDonald. *Introduction to Commutative Algebra.* Addison-Wesley, 1969.

[2] N. Bourbaki. *Eléments de Mathématiques, Algèbre, Ch. 3.* Hermann, Paris, 1970.

[3] M. Elkadi and B. Mourrain. Approche Effective des Résidus Algébriques. Rapport de Recherche 2884, INRIA, 1996.

[4] J. Emsalem. Géométrie des points épais. *Bull. Soc. Math. France*, 106:399–416, 1978.

[5] W. Gröbner. *Algebrische Geometrie II*, volume 737 of *Bib. Inst. Mannheim.* Hochschultaschenbücher, 1970.

[6] F.S. Macaulay. *The Algebraic Theory of Modular Systems.* Cambridge Univ. Press, 1916.

9 Annexe 1

```
#1############################################################################




#DESCRIPTION#

#si I=(Q1,...,Qn) n'a pas de composante m-pt primaire ou si la multiplicite est simple, le programme ne fonctionne
pas.


#ENTREES#

# Q est une liste  de polynomes [Q1,...,Qm] (en u[i] I=1..n).
# pt est un point de K^n que l'on donne aussi sous forme de liste.

#SORTIES#
# orth est la trace du Systeme Inverse de I sur les polynomes.
# Uks contient les matrices de multiplication dans le quotient.




SI:=proc(Q,pt)

local i,j,p,m,orth,som,aux,o,D,P,bin,usedorth,B,L,lambda,dlambda,
U,s,st,ds,oldds,GM,dimsol,Uks,Uk,Aks,Ak,k,V,Vb,Vaux,i1,b,b1,nbiter,b0,
L0,coefb,termb,orthu,ns,zero,Laux,Llaux;

global n;

#initialisation#

# orth est une liste qui contient les elements de L'orthogonal a I deja trouves
# D=[d[1],d[2],...,d[n]]

m:=nops(Q);
n:=nops(pt);
P:=[];

#on se ramene en zero par une translation

for p in Q do
   for i from 1 to n do
    p:=subs(u[i]=u[i]+pt[i],p);
   od;
   P:=[op(P),expand(p)]
od;




#ldegree(Pi) est l'ordre de Pi

n {ordres dans I}
o:=ldegree(P[1]);

for i from 2 to m do o:=min(ldegree(P[i]),o) od;
```

```
som:=0;
D:=[];
for  i from 1 to n do
   som:=som +d[i];
   D:=[op(D),d[i]]
od;
orth:=[[1]];
bin:=1;
for i to o-1 do
   bin:=expand(som*bin);
   coeffs(bin,D,'aux');
   orth:=[op(orth),[aux]];
od;



usedorth:=orth[o];

# on integre les elements de usedorth suivant [Mourrain] p17-18
# il faut savoir pour apres d'ou viennent les lambda qui apparaissent
# on les stoque (y compris quand ils sont nuls)
# dans une matrice de taille n x nops(usedorth)=:B.

B:=linalg[matrix](n,nops(usedorth));
for i from n by -1 to 1 do
   usedorth:=subs(d[i+1]=0,usedorth);
   for j from 1 to nops(usedorth)do
      B[i,j]:= d[i]*usedorth[j]
  od;
od;

# construction des matrices U pour le premier tour

s:=nops(orth);
ds:=nops(usedorth);
st:=sum('nops(orth[sumi])','sumi'=1..s);


Uks:=[];

if s=1 then
   for i to n do
      Uk:=linalg[matrix](1,1,0);
      print(`Uki`,i,Uk);
      Uks:=[op(Uks),evalm(Uk)]
   od;
else
   for i to n do
      Uk:=initUk(orth,i,D,ds,st);
      Uks:=[op(Uks),evalm(Uk)]
   od;
fi;

print(Uks);

# construction des matrices A pour le premier tour


Aks:=[];
for k to n do
   Ak:=initAk(B,ds,k,P,m);
   Aks:=[op(Aks),evalm(Ak)]
od;

# resolution au premier tour

zero0:=[op(Uks)];
```

```
if s=1 then
   for i to n*(n-1)/2 do zero0:=[op(zero0),op(Uks)] od;
   GM:=linalg[blockmatrix](n*(n-1)/2,n,op(zero0));
   LA:=linalg[blockmatrix](1,n,op(Aks));
   GM:=linalg[stack](GM,LA);
else

   GM:=constGM(Uks,Aks,st,ds,ds);
fi;

V:=sol(evalm(GM),ds,'dimsol');

#on construit une base de l'espace des solutions:


Vb:=[];
for i to dimsol do
   Vaux:=[];for k to linalg[rowdim](V) do Vaux:=[op(Vaux),V[k,1]] od;
   for j to i-1 do Vaux:=subs(v[j]=0,Vaux) od;
   Vaux:=subs(v[i]=1,Vaux);
   for j from i+1 to dimsol do Vaux:=subs(v[j]=0,Vaux) od;
   Vb:=[op(Vb),Vaux];
od;

L0:=[];
for i1 to dimsol do L0:=[op(L0),linalg[matrix](n,ds,Vb[i1])] od;

L:=[L0];


b0:=[];
for i1 to dimsol do
   b1:=0;
   for i from 1 to n do
      for j from 1 to ds do
         b1:= b1+B[i,j]*L[1][i1][i,j]
      od;
   od;
   b0:=[op(b0),b1];
od;

b:=[b0];




orth:=[op(orth),b[1]];
s:=nops(orth);



orthu:=[];
for i from o to s do
   orthu:=[op(orthu),op(orth[i])]
od;


ns:=nops(orthu);
oldds:=ds;
ds:=dimsol;
st:=st+ds;


#### initialisation de M et mon

# mon est l'ensemble des monomes qui apparaissent dans les polynomes que l'on integre .
```

```
mon:=[1];
M:=linalg[matrix](1,1,1);

for i from 2 to nops(orthu) do
   vect:=linalg[matrix](nops(mon),1,0);

   c:=[coeffs(expand(orthu[i]),D,'t')];t:=[t];
   for j to nops(t) do
      if member(t[j],mon,'place') then
         vect[place,1]:=c[j];
      else
         mon:=[op(mon),t[j]];
         vect:=linalg[stack](vect,linalg[matrix](1,1,c[j]));
      fi;
   od;
      if linalg[rowdim](vect)-linalg[rowdim](M)>0 then

         M:=linalg[stack](M,linalg[matrix](linalg[rowdim](vect)-linalg[rowdim](M),linalg[coldim](M),0));
      fi;
         M:=linalg[concat](M,vect);
od;




#la premiere fois on trouve de vrais nouveaux polynomes qui forment une base



#################################################################
##### regime permanent ###########################################
#################################################################



for nbiter to 15 while ds<>0 do

   # construction des matrices U


   for i to n do
      Uk2:=constUk(evalm(Uks[i]),orth,i,L[nbiter],s,st,ds,ns,oldds);
      Uks:=subsop(i=evalm(Uk2),Uks);
   od;


   # construction des A


   B:=linalg[matrix](n,nops(orthu));
   usedorth:=orthu;
   for i from n by -1 to 1 do
      usedorth:=subs(d[i+1]=0,usedorth);\char"09
      for j from 1 to nops(orthu) do
         B[i,j]:= d[i]*usedorth[j]
      od;
   od;

   for k to n do
      Ak2:=evalm(constAk(evalm(Aks[k]),B,ds,k,P,m));
      Aks:=subsop(k=evalm(Ak2),Aks)
   od;

   # resolution
```

```
GM:=constGM(Uks,Aks,st,ns,ds);
V:=sol(evalm(GM),ns,'dimsol');

# on construit une base de l'espace des solutions:
# ATTENTION comme il y a des redondances avec avant il ne s'agit
pas d'une base de l'espace que l'on va rajouter



Vb:=[];
for i to dimsol do
   Vaux:=[];
   for k to linalg[rowdim](V) do Vaux:=[op(Vaux),V[k,1]] od;
   for j to i-1 do Vaux:=subs(v[j]=0,Vaux) od;
   Vaux:=subs(v[i]=1,Vaux);
   for j from i+1 to dimsol do Vaux:=subs(v[j]=0,Vaux) od;
   Vb:=[op(Vb),Vaux];
od;

L0:=[];
for i1 to dimsol do L0:=[op(L0),linalg[matrix](n,ns,Vb[i1])] od;


zero:=linalg[matrix](linalg[rowdim](L0[1]),linalg[coldim](L0[1])-linalg[coldim](L[1][1]),0);
for i to nbiter do
   Llaux:=[];
   for j to nops(L[i]) do
      Laux:=linalg[concat](L[i][j],zero);
      Llaux:=[op(Llaux),evalm(Laux)];
   od;

   L:=subsop(i=Llaux,L);
od;

L:=[op(L),L0];



b0:=[];
for i1 to dimsol do
   b1:=0;
   for i from 1 to n do
      for j from 1 to ns do
         b1:= b1+B[i,j]*L[nbiter+1][i1][i,j]
      od;
   od;
   b0:=[op(b0),b1]

od;

b0:=firstsort(b0,L0,'newL0',o+nbiter);

L:=subsop(nops(L)=newL0,L);




b:=[op(b),b0];

b:=subsop(nops(b)=makebase(M,'newM',b[nops(b)],orthu,mon,'newmon',D,L[nops(L)],'newL'),b);


mon:=newmon;
L:=subsop(nops(L)=newL,L);
M:=evalm(newM);
```

```
   orth:=[op(orth),b[nbiter+1]];
   orthu:=[op(orthu),op(b[nbiter+1])];
   ns:=nops(orthu);
   oldds:=ds;
   ds:=nops(b[nbiter+1]);print('pds',ds);
   st:=st+ds;
   s:=nops(orth);


od;


orth;


end:



#2#############################################################################



# attention n est global

# procedure definissant l'action des derivations sur les polynomes

`&.` := proc(a,b)

local i,D;
global n;

D:=[];
for i from 1 to n do D:=[op(D),d[i]=u[i]^(-1)] od;
select(u_->evalb(degree(denom(u_))=0),expand(subs(D,a)*b));
for i from 1 to n do subs(u[i]=0,") od;
end:



#3############################################################################




# procedure definissant l'action des polynomes sur les derivations

`&%`:= proc(a,b)

local i,U;
global n;

U:=[];
for i from 1 to n do U:=[op(U),u[i]=d[i]^(-1)] od;
select(d_->evalb(degree(denom(d_))=0),expand(subs(U,a)*b));
  end:



#4###########################################################################



# procedure derive
# donne la liste des derivees par rapport aux d[K] (dans l'ordre)
```

```
derive:=proc(L)

local i,res;

res:=[];
for i to n do
   res:=[op(res),expand((L-subs(d[i]=0,L))/d[i])];
od;

end:


#5##########################################################################

# n est global

# procedure d'initialisation des matrices Uk



initUk:=proc(orth,k,D,ds,st)


local i,j,s,U,c,T,l;



s:=nops(orth);
U:=linalg[matrix](st-ds,ds,0);


   for i to ds do
      c:=coeffs(expand((orth[s][i]-subs(d[k]=0,orth[s][i]))/d[k]),D,'T');
      c:=[c];
      T:=[T];
      for l to nops(T) do
         for j while T[l]<>orth[degree(T[l])+1][j] do od;
         U[sum('nops(orth[sumj])','sumj'=1..degree(T[l]))+j,i]:=c[l];
      od
   od;

evalm(U)
end:




#6##############################################################################


# construction des matrices Uk grace aux coefficients lamda resultant de la
# resolution du systeme GM.V=0
# oldds=ancien ds


constUk:=proc(Uk,orth,k,listL,s,st,ds,ns,oldds)

local newU,i,j,zero,col;



newU:=linalg[matrix](st-ds,1,0);
for j to ns-ds do
   newU[j+st-ns,1]:=listL[1][k,j];
od;

col:=linalg[matrix](st-ds,1,0);
```

```
for i from 2 to ds do
   for j to ns-ds do
      col[j+st-ns,1]:=listL[i][k,j];
   od;
   newU:=linalg[concat](newU,col);
od;



zero:=linalg[matrix](oldds,linalg[coldim](Uk),0);
if st-ds-linalg[rowdim](Uk)<>0 then
   Uk2:=linalg[stack](evalm(Uk),evalm(zero));
   Uk2:=linalg[concat](evalm(Uk2),evalm(newU));
else
   Uk2:=linalg[concat](evalm(Uk),evalm(newU));
fi;

evalm(Uk2);

end:




#7#########################################################################

initAk:=proc(B,ds,k,P,m)

local A,j,l;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j] &. P[l] od;
od;


evalm(A);


end:





#7bis#########################################################################

constAk:=proc(Ak,B,ds,k,P,m)

local A,j,l;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j+linalg[coldim](B)-ds] &. P[l] od;
od;


Ak2:=linalg[concat](evalm(Ak),evalm(A));

evalm(Ak2);


end:


#8##########################################################################
```

```
#procedure construisant la matrice correspondant au systeme a resoudre

constGM:=proc(listU,listA,st,ns,ds)


local i,j,l,LA,GM,zero,listaux;



zero:=linalg[matrix](st-ds,ns,0);
listaux:=[];
for i to n do
   for j to n-i do
      for l to j-1 do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),listU[n+1-i]];
      for l from j+1 to n-i do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),-listU[j]];
      for l from n+2-i to n do listaux:=[op(listaux),zero] od;
   od;
od;

GM:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));

LA:=evalm(linalg[blockmatrix](1,n,op(listA)));
GM:=evalm(linalg[stack](GM,LA));


GM;

end:



#9#########################################################################

#resolution du systeme
# rend un vecteur parametre en v[i]

sol:=proc(GM,ns,dimsol)

local zero,res,V,i,rang;

zero:=linalg[matrix](linalg[rowdim](GM),1,0);

res:=linalg[linsolve](evalm(GM),zero,'rang',v);



dimsol:=linalg[coldim](GM)-linalg[rank](GM);



res;



end:



#10########################################################################

# procedure eliminamt les polynomes de degree inferieur a d

firstsort:=proc(b0,L0,newL0,deg)


local i,nb0,nL0;
```

```
nb0:=[];
nL0:=[];

for i to nops(b0) do

   if degree(b0[i])=deg then

      nL0:=[op(nL0),L0[i]];
      nb0:=[op(nb0),b0[i]];

   fi;

od;

newL0:=nL0;

nb0;

end:





#11####################################################################


# procedure prenant en entree les candidats a etre de nouveaux vecteurs
de la base des solutions candid
# et rendant une base des solutions vraiment nouvelles.

# on passe aussi en entree les monomes qui interviennent dans la
decomposition des polynomes de orthu
et la matrice correspondant aux vecteurs de orthu.



makebase:=proc(M,newM,candid,orthu,mon,newmon,D,candidL,newL)

local i,j,vect,Laux,listv,nM,c,t,northu,nmon,nMaux;

Laux:=[];
nmon:=mon;
listv:=[];
northu:=orthu;
nM:=M;
bd:=[];


for i to nops(candid) do
   vect:=linalg[matrix](nops(nmon),1,0);

   c:=[coeffs(expand(candid[i]),D,'t')];
   t:=[t];
   for j to nops(t) do
      if member(t[j],nmon,'place') then
         vect[place,1]:=c[j];
      else
         nmon:=[op(nmon),t[j]];
         vect:=linalg[stack](vect,linalg[matrix](1,1,c[j]))
      fi;
   od;
      nMaux:=evalm(nM);


      if linalg[rowdim](vect)-linalg[rowdim](nM)>0 then
```

```
        nMaux:=evalm(linalg[stack](nM,linalg[matrix](linalg[rowdim](vect)-linalg[rowdim](nM),linalg[coldim](nM)
,0)));
      fi;
        nMaux:=evalm(linalg[concat](nMaux,vect));
      if linalg[rank](nMaux)<>nops(northu) then
        nM:=evalm(nMaux);
        northu:=[op(northu),candid[i]];
        bd:=[op(bd),candid[i]];
        Laux:=[op(Laux),candidL[i]];
      fi;
od;

newmon:=nmon;
newL:=Laux;
newM:=nM;

bd;

end:
```

10 Annexe 2

```
#cette version de l'algorithme triangularise la base des vecteurs au fur et a mesure de sa construction.
#Cela permet de calculer plus facilement le rang quand on veut savoir si une solution potentielle est vraiment une nouvelle solution.


#1############################################################################




#DESCRIPTION#

#si I=(P1,...,Pn) n'a pas de composante m-pt primaire ou si la multiplicite est simple le programme ne fonctionne pas.

#ENTREES#

# Q est une liste  de polynomes [Q1,...,Qm] (en u[i] I=1..n)
# pt est un point de K^n que l'on donne aussi sous forme de liste

#SORTIES#

# orth est la base de l'intersection du systeme inverse et de l'espace des polynomes en d .
# Uks est une liste qui contient les matrices de multiplication dans le quotient.





SI:=proc(Q,pt)

local i,j,p,m,orth,som,aux,o,D,P,bin,usedorth,B,L,lambda,dlambda,
U,s,st,ds,oldds,GM,dimsol,Uks,Uk,Aks,Ak,k,V,Vb,Vaux,i1,b,b1,nbiter,b0,
L0,coefb,termb,orthu,ns,zero,Laux,Llaux,nbwithoutL;

global n;

#initialisation#

# orth est une liste qui contient les elements de L'orthogonal a I deja trouves
# D=[d[1],d[2],...,d[n]]

m:=nops(Q);
n:=nops(pt);
P:=[];

#on se ramene en zero par une translation

for p in Q do
   for i from 1 to n do
    p:=subs(u[i]=u[i]+pt[i],p);
   od;
   P:=[op(P),expand(p)]
od;




#ldegree(Pi) est l'ordre de Pi

o:=ldegree(P[1]);
for i from 2 to m do o:=min(ldegree(P[i]),o) od;
som:=0;
```

```
D:=[];
for  i from 1 to n do
   som:=som +d[i];
   D:=[op(D),d[i]]
od;
orth:=[[1]];
bin:=1;
for i to o-1 do
   bin:=expand(som*bin);
   coeffs(bin,D,'aux');
   orth:=[op(orth),[aux]];
od;



usedorth:=orth[o];

# on integre les elements de usedorth suivant [Mourrain] p17-18
# il faut savoir pour apres d'ou viennent les lambda qui apparaisent.
# On les stoque donc dans une matrice de taille n x nops(usedorth)=:B.

B:=linalg[matrix](n,nops(usedorth));
for i from n by -1 to 1 do
   usedorth:=subs(d[i+1]=0,usedorth);
   for j from 1 to nops(usedorth)do
      B[i,j]:= d[i]*usedorth[j]
  od;
od;

# construction des matrices U pour le premier tour

s:=nops(orth);
ds:=nops(usedorth);
st:=sum('nops(orth[sumi])','sumi'=1..s);


#nombre de polynomes de la base pour lesquels il n'existe pas de matrice L calculee .

nbwithoutL:=ds;



Uks:=[];

if s=1 then
   for i to n do
      Uk:=linalg[matrix](1,1,0);
      Uks:=[op(Uks),evalm(Uk)]
   od;
else
   for i to n do
      Uk:=initUk(orth,i,D,ds,st);
      Uks:=[op(Uks),evalm(Uk)]
   od;
fi;


# construction des matrices A pour le premier tour


Aks:=[];
for k to n do
   Ak:=initAk(B,ds,k,P,m);
   Aks:=[op(Aks),evalm(Ak)]
od;

# resolution au premier tour
```

```
zero0:=[op(Uks)];

if s=1 then
   for i to n*(n-1)/2 do zero0:=[op(zero0),op(Uks)] od;
   GM:=linalg[blockmatrix](n*(n-1)/2,n,op(zero0));
   LA:=linalg[blockmatrix](1,n,op(Aks));
   GM:=linalg[stack](GM,LA);
else

   GM:=constGM(Uks,Aks,st,ds,ds);
fi;

V:=sol(evalm(GM),ds,'dimsol');

#on construit une base de l'espace des solutions:



L0:=[];
for i1 to dimsol do L0:=[op(L0),linalg[matrix](n,ds,V[i1])] od;

L:=[L0];


b0:=[];
for i1 to dimsol do
   b1:=0;
   for i from 1 to n do
      for j from 1 to ds do
         b1:= b1+B[i,j]*L[1][i1][i,j]
      od;
   od;
   b0:=[op(b0),b1];
od;

b:=[b0];




orth:=[op(orth),b[1]];
s:=nops(orth);

#en fait s:=s+1 sauf si y'a pas de sol et la je sais pas faut voir.

orthu:=[];
for i from o to s do
   orthu:=[op(orthu),op(orth[i])]
od;


ns:=nops(orthu);
oldds:=ds;
ds:=dimsol;
st:=st+ds;


#### initialisation de M et mon






c:=[coeffs(expand(orthu[1]),D,'t')];
t:=[t];
mon:=t;
vect:=linalg[matrix](nops(mon),1,0);
```

```
for j to nops(t) do
   member(t[j],mon,'place');
   vect[place,1]:=c[j]
 od;
M:=evalm(vect);

for i from 2 to nops(orthu) do
   vect:=linalg[matrix](nops(mon),1,0);

   c:=[coeffs(expand(orthu[i]),D,'t')];t:=[t];
   for j to nops(t) do
      if member(t[j],mon,'place') then
         vect[place,1]:=c[j];
      else
         mon:=[t[j],op(mon)];
         vect:=linalg[stack](linalg[matrix](1,1,c[j]),vect);
      fi;
   od;
      if linalg[rowdim](vect)-linalg[rowdim](M)>0 then

         M:=linalg[stack](linalg[matrix](linalg[rowdim](vect)-linalg[rowdim](M),linalg[coldim](M),0),M);
      fi;
         M:=linalg[concat](vect,M);
od;




#la premiere fois on trouve de vrais nouveaux polynomes qui forment une base

################################################################
##### regime permanent###########################################
################################################################



for nbiter to 10 while ds<>0 do

   # construction des matrices U


   for i to n do
      Uk2:=constUk(evalm(Uks[i]),orth,i,L[nbiter],s,st,ds,ns,oldds);
      Uks:=subsop(i=evalm(Uk2),Uks);
   od;


   # construction des A


   B:=linalg[matrix](n,nops(orthu));
   usedorth:=orthu;
   for i from n by -1 to 1 do
      usedorth:=subs(d[i+1]=0,usedorth);
      for j from 1 to nops(orthu) do
         B[i,j]:= d[i]*usedorth[j]
      od;
   od;

   for k to n do
      Ak2:=constAk(evalm(Aks[k]),B,ds,k,P,m);
      Aks:=subsop(k=evalm(Ak2),Aks)
   od;

   # resolution
```

```
GM:=constGM(Uks,Aks,st,ns,ds);
V:=sol(evalm(GM),ns,'dimsol');

# on construit une base de l'espace des solutions:
# ATTENTION comme il y a des redondances avec avant il ne s'agit pas d'une base de l'espace que l'on va rajouter




L0:=[];
for i1 to dimsol do L0:=[op(L0),linalg[matrix](n,ns,V[i1])] od;


zero:=linalg[matrix](linalg[rowdim](L0[1]),linalg[coldim](L0[1])-linalg[coldim](L[1][1]),0);
for i to nbiter do
   Llaux:=[];
   for j to nops(L[i]) do
      Laux:=linalg[concat](L[i][j],zero);
      Llaux:=[op(Llaux),evalm(Laux)];
   od;

   L:=subsop(i=Llaux,L);
od;

L:=[op(L),L0];



b0:=[];
for i1 to dimsol do
   b1:=0;
   for i from 1 to n do
      for j from 1 to ns do
         b1:= b1+B[i,j]*L[nbiter+1][i1][i,j]
      od;
   od;
   b0:=[op(b0),b1]

od;

b0:=firstsort(b0,L0,'newL0',o+nbiter);

L:=subsop(nops(L)=newL0,L);




b:=[op(b),b0];


b:=subsop(nops(b)=makebase(M,'newM',b[nops(b)],orthu,mon,'newmon',D,L[nops(L)],'newL',L,nbiter+o,nbwithoutL),b);


mon:=newmon;

L:=subsop(nops(L)=newL,L);
M:=evalm(newM);


orth:=[op(orth),b[nbiter+1]];
orthu:=[op(orthu),op(b[nbiter+1])];
ns:=nops(orthu);
oldds:=ds;
ds:=nops(b[nbiter+1]);
st:=st+ds;
```

```
   s:=nops(orth);

od;

print(orth);

end:

#2##########################################################################

# attention n est global

# procedure definissant l'action des derivations sur les polynomes

`&.` := proc(a,b)

local i,D;
global n;

D:=[];
for i from 1 to n do D:=[op(D),d[i]=u[i]^(-1)] od;
select(u_->evalb(degree(denom(u_))=0),expand(subs(D,a)*b));
for i from 1 to n do subs(u[i]=0,") od;
end:

#3##########################################################################

# procedure definissant l'action des polynomes sur les derivations

`&%`:= proc(a,b)

local i,U;
global n;

U:=[];
for i from 1 to n do U:=[op(U),u[i]=d[i]^(-1)] od;
select(d_->evalb(degree(denom(d_))=0),expand(subs(U,a)*b));

  end:

#4##########################################################################

# procedure derive
# donne la liste des derivees par rapport aux d[K] (dans l'ordre)

derive:=proc(L)

local i,res;

res:=[];
for i to n do
   res:=[op(res),expand((L-subs(d[i]=0,L))/d[i])];
```

```
od;

end:


#5##############################################################################

# n global utilise mais pas passe en argument!

# procedure d'initialisation des matrices Uk

#s=nops(orth)

initUk:=proc(orth,k,D,ds,st)


local i,j,s,U,c,T,l;




s:=nops(orth);
U:=linalg[matrix](st-ds,ds,0);


   for i to ds do
      c:=coeffs(expand((orth[s][i]-subs(d[k]=0,orth[s][i]))/d[k]),D,'T');
      c:=[c];
      T:=[T];
      for l to nops(T) do
         for j while T[l]<>orth[degree(T[l])+1][j] do od;
         U[sum('nops(orth[sumj])','sumj'=1..degree(T[l]))+j,i]:=c[l];
      od
   od;

evalm(U)
end:




#6##############################################################################

# construction des matrices Uk grace aux coefficients lamda resultant de la
# resolution du systeme GM.V=0
# oldds=ancien ds


constUk:=proc(Uk,orth,k,listL,s,st,ds,ns,oldds)

local newU,i,j,zero,col;



newU:=linalg[matrix](st-ds,1,0);
for j to ns-ds do
   newU[j+st-ns,1]:=listL[1][k,j];
od;

col:=linalg[matrix](st-ds,1,0);
for i from 2 to ds do
   for j to ns-ds do
      col[j+st-ns,1]:=listL[i][k,j];
   od;
   newU:=linalg[concat](newU,col);
od;
```

```
zero:=linalg[matrix](oldds,linalg[coldim](Uk),0);
if st-ds-linalg[rowdim](Uk)<>0 then
   Uk2:=linalg[stack](evalm(Uk),evalm(zero));
   Uk2:=linalg[concat](evalm(Uk2),evalm(newU));
else
   Uk2:=linalg[concat](evalm(Uk),evalm(newU));
fi;

evalm(Uk2);

end:




#7##########################################################################

initAk:=proc(B,ds,k,P,m)

local A,j,l;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j] &. P[l] od;
od;


evalm(A);


end:





#7bis#######################################################################

constAk:=proc(Ak,B,ds,k,P,m)

local A,j,l;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j+linalg[coldim](B)-ds] &. P[l] od;
od;


Ak2:=linalg[concat](evalm(Ak),evalm(A));

evalm(Ak2);


end:


#8##########################################################################



#procedure construisant la matrice correspondant au systemme a resoudre

constGM:=proc(listU,listA,st,ns,ds)
```

```
local i,j,l,LA,GM,zero,listaux;





zero:=linalg[matrix](st-ds,ns,0);
listaux:=[];
for i to n do
   for j to n-i do
      for l to j-1 do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),listU[n+1-i]];
      for l from j+1 to n-i do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),-listU[j]];
      for l from n+2-i to n do listaux:=[op(listaux),zero] od;
   od;
od;

GM:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));

LA:=evalm(linalg[blockmatrix](1,n,op(listA)));
GM:=evalm(linalg[stack](GM,LA));



GM;

end:



#9########################################################################


#resolution du systeme
# rend un vecteur parametre en v[i]

sol:=proc(GM,ns,dimsol)

local zero,res,V,i,rang;



res:=linalg[kernel](evalm(GM),'dimsol');





res;



end:



#10#######################################################################

# procedure virant les polynomes de degree inferieur a d

firstsort:=proc(b0,L0,newL0,d)
```

```
local i,nb0,nL0;

nb0:=[];
nL0:=[];

for i to nops(b0) do

   if degree(b0[i])=d then

      nL0:=[op(nL0),L0[i]];
      nb0:=[op(nb0),b0[i]];

   fi;

od;

newL0:=nL0;

nb0;

end:



#version 2

firstsort2:=proc(b0,L0,newL0,d)


local i,nb0,nL0;

nb0:=b0;
nL0:=L0;

i:=1;
while i<=nops(nb0) do
   if degree(nb0[i])<d then
      nb0:=subsop(i=NULL,nb0);
      nL0:=subsop(i=NULL,nL0);
   else
      i:=i+1;
   fi;
od;

newL0:=nL0;

nb0;


end:






#11##############################################################################


# procedure prenant en entree les candidats a etre de nouveau vecteurs de la base des solutions candid
# et rendant une base des solution vraiment nouvelles

# on passe aussi en entree les monomes qui interviennent dans la decomposition des polynomes
#de orthu et la matrice correspondant aux vecteurs de orthu.
```

```
makebase:=proc(M,newM,candid,orthu,mon,newmon,D,candidL,newL,L,d,nbwithoutL)

local i,j,vect,Laux,listv,nM,c,t,northu,nmon,nMaux,vectd;

Laux:=[];
nmon:=mon;
listv:=[];
northu:=orthu;
nM:=M;
bd:=[];
L2:=L;
mond:=[];

for i to nops(candid) do
   vect:=linalg[matrix](nops(nmon),1,0);

   c:=[coeffs(expand(candid[i]),D,'t')];
   t:=[t];

   oldnbmon:=nops(nmon);
   oldnbmond:=nops(mond);

   if nops(mond)<>0 then
      vectd:=linalg[matrix](nops(mond),1,0);
   fi;

   for j to nops(t) do

      if degree(t[j])=d then

         if nops(mond)=0 then
            mond:=[t[j]];
            vectd:=linalg[matrix](1,1,c[j]);
         else

            if member(t[j],mond,'place') then
               vectd[place,1]:=c[j];
            else
               mond:=[t[j],op(mond)];
               vectd:=linalg[stack](linalg[matrix](1,1,c[j]),vectd)
            fi;
         fi;

      else
         if member(t[j],nmon,'place') then
            vect[place,1]:=c[j];
         else
            nmon:=[t[j],op(nmon)];
            vect:=linalg[stack](linalg[matrix](1,1,c[j]),vect)
         fi;
      fi;
   od;

   nbmon:=nops(nmon);
   nbmond:=nops(mond);

   vect:=linalg[stack](vectd,vect);


      nMaux:=evalm(nM);




if -linalg[rowdim](M)+linalg[rowdim](nMaux)>0 then
```

```
               p1Maux:=linalg[delrows](nMaux,1..-linalg[rowdim](M)+linalg[rowdim](nMaux));


   if nbmon-oldnbmon>0 then

               p1Maux:=evalm(linalg[stack](linalg[matrix](nbmon-oldnbmon,linalg[coldim](nM),0),p1Maux));
           fi;
       p2Maux:=linalg[delrows](nMaux,-linalg[rowdim](M)+1+linalg[rowdim](nMaux)..linalg[rowdim](nMaux));



           if nbmond-oldnbmond>0 then
       p2Maux:=evalm(linalg[stack](linalg[matrix](nbmond-oldnbmond,linalg[coldim](nM),0),p2Maux));
           fi;
       nMaux:=evalm(linalg[stack](p2Maux,p1Maux));
  else
    nMaux:=evalm(linalg[stack](linalg[matrix](nbmon-oldnbmon+nbmond-oldnbmond,linalg[coldim](nM),0),nMaux));
fi;

         nMaux:=evalm(linalg[concat](vect,nMaux));

lorth:=[op(northu),candid[i]]; for r to nops(L2) do L2:=subsop(r=map(evalm,L2[r]),L2) od;


rg:=trigrang(lorth,'neworthu',evalm(nMaux),'newM2',L2,'newL2',nbwithoutL);
mark1;

nMaux:=evalm(newM2);






      if rg<>nops(northu) then
         northu:=neworthu;
         for r to nops(L2) do L2:=subsop(r=map(evalm,newL2[r]),L2) od;
         nM:=evalm(nMaux);



      fi;
od;

newL:=L2[nops(L2)];
newmon:=[op(mond),op(nmon)];
newM:=nM;

for r from nops(orthu)+1 to nops(northu) do bd:=[op(bd),northu[r]]; od;


bd;

end:




#12#######################################################################

#ENTREES : lorth    liste des candidats à etre des vecteurs de base
#          M        la precente matrice M calculee
#          L        liste des matrices L

# procedure triangularisant la matrice M et rendant les matrices L modifiées dans nnewL,
# la nouvelle matrice M dans nnewM et les vecteurs de la nouvelle base dans neworthu
```

```
trigrang:=proc(lorth,neworthu,M,nnewM,L,nnewL,nbwithoutL)

local northu,nL,nM,i,nbl,T,H,I,nMI;


northu:=lorth;

#on fais une transposition car maple triangularise en faisant des operations sur les lignes
#et on veut que ca corresponde a des operations sur les polynomes polynomes de la base.


nM:=linalg[transpose](M);
Lvide:=linalg[matrix](linalg[rowdim](L[1][1]),linalg[coldim](L[1][1]),0);


nL:=[evalm(Lvide)];
for i to nbwithoutL-1 do nL:=[op(nL),evalm(Lvide)] od;



for i to nops(L) do nL:=[op(nL),op(L[i])] od;
print(nL);
nbc:=linalg[coldim](nM);
nbl:=linalg[rowdim](nM);

I:=array(1..nbl,1..nbl,identity);
nMI:=linalg[concat](nM,evalm(I));

# on ne peut pas esperer deduire le rang de cette facon a cause de I -> le rang toujours le meme.

nMI:=linalg[gausselim](evalm(nMI));

#extraire la matrice utile

H:=linalg[delcols](evalm(nMI),1..nbc);
T:=linalg[delcols](evalm(nMI),nbc+1..nbc+nbl);

# calcul du rang de T


ligne:=0;
colonne:=1;
while  colonne<=linalg[coldim](T) and T[ligne+1,colonne]<>0  do ligne:=ligne+1; colonne:=colonne+1; od;
rang:=ligne;



# nLtronque ne contient que les matrices utiles de nL et elles sont
# rangees dans l'ordre inverse

nLtronquee:=[];  for i to nbl do nLtronquee:=[nL[i],op(nLtronquee)] od;
nLtronquee:=convert(nLtronquee,vector);





temp:=map(evalm,nLtronquee);
nLtronquee:=map(evalm,evalm(H &* temp));
nLtronquee:=map(evalm,convert(nLtronquee,list));


for i to nops(nLtronquee) do nL:=subsop(i=evalm(nLtronquee[nops(nLtronquee)+1-i]),nL); od;

nLL:=[];
k:=0;
for i to nops(L) do
```

```
   Laux:=[];
   for j to nops(L[i]) do

      k:=k+1;
      Laux:=[op(Laux),nL[k+nbwithoutL]]
   od;
   nLL:=[op(nLL),map(evalm,Laux)]
od;
print('rg',rang);



orthut:=[];

for i to nops(northu) do orthut:=[op(orthut),northu[nops(northu)+1-i]] od;

orthut:=convert(orthut,vector);

orthut:=evalm(H &* orthut);

orthut:=convert(orthut,list);

nnewM:=linalg[transpose](T);

northu:=[];

for i to nops(orthut) do northu:=[op(northu),orthut[nops(orthut)+1-i]] od;

neworthu:=northu;

nnewL:=nLL;

rang;


end:
```

11 Annexe 3

```
# Algorithme ne résolvant que partiellement la matrice GM pour ne donner
# que des nouvelles solutions.




#1###########################################################################




#DESCRIPTION#

#si I=(P1,...,Pn) n'a pas de composante m-pt primaire ou si la multiplicite est simple le programme ne fonctionne pas

#ENTREES#

# P est une liste  de polynomes [P1,...,Pm] (en u[i] I=1..n)
# pt est un point de K^n que l'on donne aussi sous forme de liste

#SORTIES#

# orthu est la trace du systeme inverse sur les polynomes
# Uks contient les matrices de multiplication dans le quotient





SI:=proc(Q,pt)

local i,j,p,m,orth,som,aux,o,D,P,bin,usedorth,B,L,lambda,dlambda,
U,s,st,ds,oldds,GM,Uks,Uk,Aks,Ak,k,V,Vb,Vaux,i1,b,b1,nbiter,b0,
L0,coefb,termb,orthu,ns,zero,Laux,Llaux,H,HG2,Uks1,Uks2,nU1,nU2,udone,
zero0,LA,mon,M,vect,c,Ak2,Uk2,DS,sumi,dimsol,t,nA,newH,
newG,chgcol,l2del,ndw,newM,newmon,newL,colop,rGM,T,G,skelet1,skelet2,
dimr,Lw,w2,w,Lw2,unesol,vt,olddimGM,nlT,R,dimw,Vtemp,Vt,LT,Lt;

global n;



#initialisation#

# orth est une liste qui contient les elements de L'orthogonal a I deja trouves
# D=[d[1],d[2],...,d[n]]

m:=nops(Q);
n:=nops(pt);
P:=[];

#on se ramene en zero par une translation

for p in Q do
   for i from 1 to n do
    p:=subs(u[i]=u[i]+pt[i],p);
   od;
   P:=[op(P),expand(p)]
```

```
od;




#ldegree(Pi) est l'ordre de Pi

o:=ldegree(P[1]);
for i from 2 to m do o:=min(ldegree(P[i]),o) od;
som:=0;
D:=[];
for  i from 1 to n do
   som:=som +d[i];
   D:=[op(D),d[i]]
od;
orth:=[[1]];


bin:=1;
for i to o-1 do
   bin:=expand(som*bin);
   coeffs(bin,D,'aux');
   orth:=[op(orth),[aux]];
od;



usedorth:=orth[o];

# on integre les elements de usedorth suivant [Mourrain] p17-18
# il faut savoir pour apres d'ou viennent les lambda qui apparaisent
# on les stoque donc dans la matrice B

B:=linalg[matrix](n,nops(usedorth));
for i from n by -1 to 1 do
   usedorth:=subs(d[i+1]=0,usedorth);
   for j from 1 to nops(usedorth)do
      B[i,j]:= d[i]*usedorth[j]
  od;
od;

# construction des matrices U pour le premier tour

s:=nops(orth);
ds:=nops(usedorth);
st:=sum('nops(orth[sumi])','sumi'=1..s);

udone:=true;

if s=1 then
   udone:=false;


     Aks:=[];
     for k to n do
        Ak:=initAk(B,ds,k,P,m);
        Aks:=[op(Aks),evalm(Ak)]
     od;

# resolution au premier tour si s=1

     zero0:=[op(Uks)];


     LA:=linalg[blockmatrix](1,n,op(Aks));
     GM:=evalm(LA);
```

```
# maintenant on triangularise quand meme

     GM:=bbtrig(evalm(GM),'chgcol','newH','rGM');
     H:=newH;

# colop contient les numeros des colonnes que l'on a deplace
# a la fin de la matrice (dans l'ordre inversement chronologique
# au déplacements) pour permettre de retrouver les polynomes correspondant
# a ces colonnnes

     colop:=chgcol;


     V:=sol(evalm(GM),ds,'dimsol');



     V:=map(deroul,V,colop);




     rGM:=evalm(rGM);


     T:=initT(evalm(GM),rGM,'newG');



     G:=newG;


# skelet1 est une liste qui contiendra un pointeur indiquant pour chaque etape la
# taille de la matrice T triangulaire inversible.
# skelet2 contient la dimension du palier correspondant .

     skelet1:=[rGM];
     skelet2:=[linalg[coldim](G)];




        Uks:=[];
     for i to n do
        Uk:=linalg[matrix](1,1,0);
        Uks:=[op(Uks),evalm(Uk)]
     od;





else





     Uks:=[];
     for i to n do
        Uk:=initUk(orth,i,D,ds,st);
        Uks:=[op(Uks),evalm(Uk)]
     od;



# construction des matrices A pour le premier tour
```

```
     Aks:=[];
     for k to n do
        Ak:=initAk(B,ds,k,P,m);
        Aks:=[op(Aks),evalm(Ak)]
     od;

# resolution au premier tour


# envisager un map elvalm sur Uks et Aks

     GM:=initGM(Uks,Aks,st,ds,ds,ds,'newH','chgcol');
     H:=newH;




rGM:=linalg[rank](GM);
dimr:=linalg[coldim](GM)-linalg[rank](GM);


T:=initT(evalm(GM),rGM,'newG');
invT:=linalg[inverse](T);



G:=newG;





  Lw:=[];
   for i to dimr do
      w:=linalg[vector](dimr,0);
      w[i]:=1;
      Lw:=[op(Lw),evalm(w)];
   od;

   Lw2:=[];
   for w in Lw do
      w2:=evalm(G &* w);
      Lw2:=[op(Lw2),evalm(-w2)];
   od;



   V:=[];
   for w2 in Lw2 do
   unesol:=resol(evalm(invT),w2);
   V:=[op(V),evalm(unesol)];
   od;


skelet1:=[rGM];
skelet2:=[dimr];
colop:=[chgcol];



   Vaux:=[];
   for i to dimr do
    vt:=goodvect(skelet1,skelet2,evalm(V[i]));
    vt:=colv(evalm(vt),evalm(Lw[i]));
    vt:=nderoul(evalm(vt),colop,[ds]);
   Vaux:=[op(Vaux),evalm(vt)];
   od;
```

```
V:=Vaux;

dimsol:=nops(V);



fi;



#on construit une base de l'espace des solutions:



L0:=[];
for i1 to dimsol do L0:=[op(L0),linalg[matrix](n,ds,V[i1])] od;

L:=[L0];


b0:=[];
for i1 to dimsol do
   b1:=0;
   for i from 1 to n do
      for j from 1 to ds do
         b1:= b1+B[i,j]*L[1][i1][i,j]
      od;
   od;
   b0:=[op(b0),expand(b1)];
od;

b:=[b0];




orth:=[op(orth),b[1]];


s:=nops(orth);



orthu:=[];
for i from o to s do
   orthu:=[op(orthu),op(orth[i])]
od;


ns:=nops(orthu);
oldds:=ds;
ds:=dimsol;
st:=st+ds;



#la premiere fois on trouve de vrais nouveaux polynomes qui forment une base

###########################
##### regime permanent #####
###########################




for nbiter to 10 while ds<>0 do
```

```
   if nbiter=1 then DS:=[ns-ds] fi;
   DS:=[op(DS),ds];
   print(`pDS`,DS,`pns`,ns);

   # construction des matrices U

   Uks1:=[];
   Uks2:=[];
   for i to n do
      Uk2:=constUk(evalm(Uks[i]),orth,i,L[nbiter],s,st,ds,ns,oldds,'nU1','nU2');
      Uks:=subsop(i=evalm(Uk2),Uks);
      Uks1:=[op(Uks1),evalm(nU1)];
      Uks2:=[op(Uks2),evalm(nU2)];
   od;



   # construction des A


   B:=linalg[matrix](n,nops(orthu));
   usedorth:=orthu;
   for i from n by -1 to 1 do
      usedorth:=subs(d[i+1]=0,usedorth);
      for j from 1 to nops(orthu) do
         B[i,j]:= expand(d[i]*usedorth[j])
      od;
   od;




   for k to n do
      Ak2:=constAk(evalm(Aks[k]),B,ds,k,P,m,'nA');


     Aks:=subsop(k=evalm(nA),Aks)
   od;


   # resolution


   olddimGM:=linalg[coldim](GM);

   nlT:=linalg[rowdim](T);



   print(`pDS`,DS);

   if udone=false then

     GM:=constGM(evalm(GM),Uks1,Uks2,Aks,st,ns,ds,oldds,evalm(H),nlT,DS,'newH','newR','newG','chgcol','l2del','ndw');

   else

     GM:=evalm(constGM2(evalm(GM),Uks1,Uks2,Aks,st,ns,ds,oldds,evalm(H),nlT,DS,'newH','newR','newG','chgcol','l2del','ndw'));

    fi;


if GM=false then RETURN(orth,map(evalm,Uks)) fi;
```

```
   H:=newH;
   G:=newG;
   R:=newR;
   dimw:=ndw;


   T:=constT(evalm(T),evalm(G),l2del,dimw);


invT:=linalg[inverse](T);



   Lw:=[];
   for i to dimw do
      w:=linalg[vector](linalg[coldim](R),0);
      w[linalg[coldim](R)-dimw+i]:=1;
      Lw:=[op(Lw),evalm(w)];
   od;

   Lw2:=[];
   for w in Lw do
      w2:=evalm(R &* w);
      Lw2:=[op(Lw2),evalm(-w2)];
   od;


   V:=[];
   for w2 in Lw2 do
   unesol:=resol(evalm(invT),w2);
   V:=[op(V),evalm(unesol)];
   od;


     Vaux:=[];


# stockage des operations faites

skelet1:=[op(skelet1),skelet1[nops(skelet1)]+linalg[coldim](G)-dimw];
skelet2:=[op(skelet2),dimw];
colop:=[map((x,a) -> eval(x+eval(a)),eval(chgcol),olddimGM),op(colop)];




   for i to dimw do
    vt:=goodvect(skelet1,skelet2,evalm(V[i]));
    vt:=colv(evalm(vt),evalm(Lw[i]));
    vt:=nderoul(evalm(vt),colop,DS);
    Vaux:=[op(Vaux),evalm(vt)];
   od;


   V:=Vaux;




   dimsol:=nops(V);
```

```
   # on construit une base de l'espace des solutions:
   # ATTENTION comme il y a des redondances avec avant il ne s'agit pas d'une base de l'espace que l'on va rajouter
# on reconstruit le vecteur qu'on aurait du trouver si on avait pas fait d'operations sur les colonnes

   L0:=[];
   for i1 to dimsol do


      Vtemp:=convert(V[i1],list);
      Vtemp:=linalg[matrix](n*ns,1,Vtemp);


      if DS[1]<ns then
         Vt:=linalg[delrows](Vtemp,n*DS[1]+1..n*ns);
         Vt:=linalg[col](Vt,1);


         LT:=linalg[matrix](n,DS[1],Vt);



         Vtemp:=linalg[delrows](Vtemp,1..n*DS[1]);


         for j from 2 to nops(DS)-1 do


            Vt:=linalg[delrows](Vtemp,n*DS[j]+1..linalg[rowdim](Vtemp));
            Vt:=linalg[col](Vt,1);

            Lt:=linalg[matrix](n,DS[j],Vt);
            Vtemp:=linalg[delrows](Vtemp,1..n*DS[j]);

            LT:=linalg[concat](LT,Lt);

         od;





        Lt:=linalg[matrix](n,DS[nops(DS)],linalg[col](Vtemp,1));
         LT:=linalg[concat](LT,Lt);


      else
         LT:=linalg[matrix](n,DS[1],linalg[col](Vtemp,1));
      fi;

   L0:=[op(L0),evalm(LT)];

   od;

   zero:=linalg[matrix](linalg[rowdim](L0[1]),linalg[coldim](L0[1])-linalg[coldim](L[1][1]),0);
   for i to nbiter do
      Llaux:=[];
      for j to nops(L[i]) do
         Laux:=linalg[concat](L[i][j],zero);
         Llaux:=[op(Llaux),evalm(Laux)];
      od;

      L:=subsop(i=Llaux,L);
   od;

   L:=[op(L),L0];
```

```
   b0:=[];
   for i1 to dimsol do
      b1:=0;
      for i from 1 to n do
         for j from 1 to ns do
            b1:= b1+B[i,j]*L[nbiter+1][i1][i,j]
         od;
      od;
      b0:=[op(b0),expand(b1)]

   od;



   b:=[op(b),b0];





   orth:=[op(orth),b[nbiter+1]];
   orthu:=[op(orthu),op(b[nbiter+1])];
   ns:=nops(orthu);
   oldds:=ds;
   ds:=nops(b[nbiter+1]);
   st:=st+ds;
   s:=nops(orth);

print(`porthu`,orthu);

od;




orth

end:



#2#############################################################################



# attention n est global

# procedure definissant l'action des derivations sur les polynomes

`&.` := proc(a,b)

local i,D;
global n;

D:=[];
for i from 1 to n do D:=[op(D),d[i]=u[i]^(-1)] od;
select(u_->evalb(degree(denom(u_))=0),expand(subs(D,a)*b));
for i from 1 to n do subs(u[i]=0,") od;
end:



#3#########################################################################################################




# procedure definissant l'action des polynomes sur les derivations
```

```
`&%`:= proc(a,b)

local i,U;
global n;

U:=[];
for i from 1 to n do U:=[op(U),u[i]=d[i]^(-1)] od;
select(d_->evalb(degree(denom(d_))=0),expand(subs(U,a)*b));

  end:



#4##############################################################################



# procedure derive
# donne la liste des derivees par rapport aux d[K] (dans l'ordre)


derive:=proc(L)

local i,res;

res:=[];
for i to n do
   res:=[op(res),expand((L-subs(d[i]=0,L))/d[i])];
od;

end:


#5##############################################################################

# n global utilise mais pas passe en argument!

# procedure d'initialisation des matrices Uk

#s=nops(orth)

initUk:=proc(orth,k,D,ds,st)


local i,j,s,U,c,T,l,sumj;


s:=nops(orth);

U:=linalg[matrix](st-ds,ds,0);


   for i to ds do
      c:=coeffs(expand((orth[s][i]-subs(d[k]=0,orth[s][i]))/d[k]),D,'T');
      c:=[c];
      T:=[T];
      for l to nops(T) do
         for j while T[l]<>orth[degree(T[l])+1][j] do od;

         U[sum('nops(orth[sumj])','sumj'=1..degree(T[l]))+j,i]:=c[l];
      od
   od;

evalm(U)
end:
```

```
#6##########################################################################

# Attention ne sert que dans le cas udone=false
# n global utilise mais pas passe en argument!

# procedure d'initialisation des matrices Uk

#s=nops(orth)

initUk2:=proc(orth,k,D,ds,st)


local i,j,s,U,c,T,l,sumj;


s:=nops(orth);
U:=linalg[matrix](st-ds,st,0);


   for i to ds do
      c:=coeffs(expand((orth[s][i]-subs(d[k]=0,orth[s][i]))/d[k]),D,'T');
      c:=[c];
      T:=[T];
      for l to nops(T) do
         for j while T[l]<>orth[degree(T[l])+1][j] do od;
         U[sum('nops(orth[sumj])','sumj'=1..degree(T[l]))+j,i+1]:=c[l];
      od
   od;

evalm(U)
end:


#7##########################################################################


# construction des matrices Uk grace aux coefficients lamda resultant de la
# resolution du systeme GM.V=0
# on retoutrne aussi dans nU1 et nU2 les matrices qui vont servir a construire les matrices
# G1 et G2 donc la nouvelle GM.
# oldds=ancien ds


constUk:=proc(Uk,orth,k,listL,s,st,ds,ns,oldds,nU1,nU2)

local newU,i,j,zero,col,Uk2;



newU:=linalg[matrix](st-ds,1,0);
for j to ns-ds do
   newU[j+st-ns,1]:=listL[1][k,j];
od;

col:=linalg[matrix](st-ds,1,0);
for i from 2 to ds do
   for j to ns-ds do
      col[j+st-ns,1]:=listL[i][k,j];
   od;
   newU:=linalg[concat](newU,col);
od;

if st-ds-oldds>0 then
   nU1:=linalg[delrows](newU,st-ds-oldds+1..st-ds);
   nU2:=linalg[delrows](newU,1..st-ds-oldds);


else
```

```
  nU1:=false;
  nU2:=newU;

fi;



zero:=linalg[matrix](oldds,linalg[coldim](Uk),0);
if st-ds-linalg[rowdim](Uk)<>0 then
   Uk2:=linalg[stack](evalm(Uk),evalm(zero));
   Uk2:=linalg[concat](evalm(Uk2),evalm(newU));
else
   Uk2:=linalg[concat](evalm(Uk),evalm(newU));
fi;

evalm(Uk2);

end:




#8###########################################################################

initAk:=proc(B,ds,k,P,m)

local A,j,l;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j] &. P[l] od;
od;


evalm(A);


end:





#9############################################################################


constAk:=proc(Ak,B,ds,k,P,m,nA)

local A,j,l,Ak2;

A:=linalg[matrix](m,ds);
for j to ds do
   for l to m do A[l,j]:=B[k,j+linalg[coldim](B)-ds] &. P[l] od;
od;

nA:=evalm(A);


Ak2:=linalg[concat](evalm(Ak),evalm(A));

evalm(Ak2);


end:


#10##########################################################################
```

```
#procedure construisant la matrice correspondant au systeme a resoudre
# pour le premier tour et qui la triangule



initGM:=proc(listU,listA,st,ns,ds,H,chgcol)


local i,j,l,LA,GM,zero,listaux;


zero:=linalg[matrix](st-ds,ns,0);
listaux:=[];
for i to n do
   for j to n-i do
      for l to j-1 do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),listU[n+1-i]];
      for l from j+1 to n-i do listaux:=[op(listaux),zero] od;
      listaux:=[op(listaux),-listU[j]];
      for l from n+2-i to n do listaux:=[op(listaux),zero] od;
   od;
od;

GM:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));

LA:=evalm(linalg[blockmatrix](1,n,op(listA)));


GM:=evalm(linalg[stack](LA,GM));

GM:=bbtrig(evalm(GM),'chgcol','H','rrrrr');


GM;

end:



#11########################################################################


#procedure construisant la matrice correspondant au systeme a resoudre
# si udone=false

# les 6 derniers arguments sont des noms.


constGM:=proc(GM,listU1,listU2,listA,st,ns,ds,oldds,H,nlT,DS,newH,newR,newG,chgcol,ull,ndimw)


local i,j,l,LA,nGM,zero,zero1,zero2,listaux,listaux1,listaux2,G1,G2,HG2,colop,
point,k,Gaux,t,nlGM,delta,ncG,Maux,rang2,ull1,chgcolin,nR,rG2;


if type(listU1[1],boolean) then

# il n'y a pas de matrice U1 on construit donc directement G2

zero2:=linalg[matrix](oldds,ds,0);


listaux2:=[];
for i to n do
```

```
   for j to n-i do
      for l to j-1 do  listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),listU2[n+1-i]];
      for l from j+1 to n-i do  listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),-listU2[j]];
      for l from n+2-i to n do  listaux2:=[op(listaux2),zero2] od;
   od;
od;



G2:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux2)));
LA:=evalm(linalg[blockmatrix](1,n,op(listA)));
G1:=evalm(LA);

else

     # les matrices U1 ont ete construites on construit G1 selon le squelet de la
     # matrice GM

listaux:=[];
zero:=linalg[matrix](DS[1],ds,0);
for i to n do
  for j to n-i do
      for l to j-1 do listaux:=[op(listaux),evalm(zero)]  od;
      listaux:=[op(listaux),evalm(linalg[submatrix](listU1[n+1-i],1..DS[1],1..ds))];

      for l from j+1 to n-i do listaux:=[op(listaux),evalm(zero)] od;
      listaux:=[op(listaux),evalm(linalg[submatrix](-listU1[j],1..DS[1],1..ds))];

      for l from n+2-i to n do listaux:=[op(listaux),evalm(zero)];  od;

  od;
od;

G1:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));

point:=DS[1];


for k from 2 to nops(DS)-2 do

   zero:=linalg[matrix](DS[k],ds,0);
   listaux:=[];
   for i to n do
   for j to n-i do

      for l to j-1 do listaux:=[op(listaux),evalm(zero)];  od;
      listaux:=[op(listaux),evalm(linalg[submatrix](listU1[n+1-i],point+1..point+DS[k],1..ds))];

      for l from j+1 to n-i do listaux:=[op(listaux),evalm(zero)]; od;
      listaux:=[op(listaux),evalm(linalg[submatrix](-listU1[j],point+1..point+DS[k],1..ds))];

      for l from n+2-i to n do listaux:=[op(listaux),evalm(zero)];  od;
   od;
   od;


Gaux:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));


G1:=evalm(linalg[stack](G1,Gaux));

point:=point+DS[k];
od;
```

```
# construction de G2

zero2:=linalg[matrix](oldds,ds,0);


listaux2:=[];
for i to n do
   for j to n-i do
      for l to j-1 do  listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),listU2[n+1-i]];
      for l from j+1 to n-i do listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),-listU2[j]];
      for l from n+2-i to n do listaux2:=[op(listaux2),zero2] od;
   od;
od;



G2:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux2)));


LA:=evalm(linalg[blockmatrix](1,n,op(listA)));



G1:=evalm(linalg[stack](LA,G1));




fi;


# on applique la matrice de triangularisation sur G1 pour rester
# coherent avec la matrice GM.

G1:=HG1(evalm(G1),evalm(H));



nlGM:=linalg[rowdim](GM);
delta:=nlGM-nlT;
ncG:=linalg[coldim](G1);

if delta>0 then

  Maux:=linalg[submatrix](G1,nlGM-delta+1..nlGM,1..ncG);



  G2:=linalg[stack](Maux,G2);
fi;

G2:=bbtrig(evalm(G2),'colop','HG2','rG2');



rang2:=eval(rG2);

ull1:=linalg[rowdim](G2)-rang2;

ull:=ull1;
dimw:=linalg[coldim](G2)-rang2;
ndimw:=dimw;
```

```
# si G2 est de rang maximal on sort de la procedure avec false comme valeur

if dimw=0 then RETURN(false) fi;

# stockage des changements de colonnes

chgcol:=colop;
chgcolin:=eval(colop);



G1:=linalg[submatrix](G1,1..nlGM-delta,1..ncG);

if (nops(chgcolin)<>0) then G1:=evalm(matderoule(evalm(G1),chgcolin)) fi;


G1:=linalg[stack](G1,G2);

newG:=evalm(G1);

nR:=linalg[submatrix](G1,1..linalg[rowdim](G1)-ull1,rang2+1..ncG);


newR:=evalm(nR);


# construction newH, la matrice de triangularisation


newH:=constnewH2(evalm(H),evalm(HG2),delta);

nGM:=evalm(linalg[stack](GM,linalg[matrix](linalg[rowdim](G2)-delta,linalg[coldim](GM),0)));


nGM:=evalm(linalg[concat](nGM,G1));


nGM;

end:





#12#####################################################################

#c'est la meme que constGM sauf qu'elle n'est utilisee que quand udone=true
#c'est parce que la matrice a dans ce cas un etage de plus (on a deja construit les U1 au premier tour)

constGM2:=proc(GM,listU1,listU2,listA,st,ns,ds,oldds,H,nlT,DS,newH,newR,newG,chgcol,ull,ndimw)


local i,j,l,LA,nGM,zero,zero1,zero2,listaux,listaux1,listaux2,G1,G2,HG2,colop,
point,k,Gaux,t,nlGM,delta,ncG,Maux,rang2,ull1,chgcolin,nR,rG2;



# construction de la matrice G1

listaux:=[];
```

```
zero:=linalg[matrix](st-ns,ds,0);
for i to n do
  for j to n-i do
      for l to j-1 do listaux:=[op(listaux),evalm(zero)]  od;
      listaux:=[op(listaux),evalm(linalg[submatrix](listU1[n+1-i],1..st-ns,1..ds))];

      for l from j+1 to n-i do listaux:=[op(listaux),evalm(zero)] od;
      listaux:=[op(listaux),evalm(linalg[submatrix](-listU1[j],1..st-ns,1..ds))];

      for l from n+2-i to n do listaux:=[op(listaux),evalm(zero)];  od;

  od;
od;

G1:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));

point:=st-ns;


for k from 1 to nops(DS)-2 do

   zero:=linalg[matrix](DS[k],ds,0);
   listaux:=[];
   for i to n do
   for j to n-i do

      for l to j-1 do listaux:=[op(listaux),evalm(zero)];  od;
      listaux:=[op(listaux),evalm(linalg[submatrix](listU1[n+1-i],point+1..point+DS[k],1..ds))];

      for l from j+1 to n-i do listaux:=[op(listaux),evalm(zero)]; od;
      listaux:=[op(listaux),evalm(linalg[submatrix](-listU1[j],point+1..point+DS[k],1..ds))];

      for l from n+2-i to n do listaux:=[op(listaux),evalm(zero)];  od;
   od;
   od;


Gaux:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux)));
G1:=evalm(linalg[stack](G1,Gaux));

point:=point+DS[k];
od;


# construction de la matrice G2


zero2:=linalg[matrix](oldds,ds,0);


listaux2:=[];
for i to n do
   for j to n-i do
      for l to j-1 do  listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),listU2[n+1-i]];
      for l from j+1 to n-i do listaux2:=[op(listaux2),zero2] od;

      listaux2:=[op(listaux2),-listU2[j]];
      for l from n+2-i to n do listaux2:=[op(listaux2),zero2] od;
   od;
od;


G2:=evalm(linalg[blockmatrix](n*(n-1)/2,n,op(listaux2)));

LA:=evalm(linalg[blockmatrix](1,n,op(listA)));
```

```
G1:=evalm(linalg[stack](LA,G1));


# on applique la matrice de triangularisation sur G1 pour rester
# coherent avec la matrice GM.

G1:=HG1(evalm(G1),evalm(H));


nlGM:=linalg[rowdim](GM);
delta:=nlGM-nlT;
ncG:=linalg[coldim](G1);

if delta>0 then

  Maux:=linalg[submatrix](G1,nlGM-delta+1..nlGM,1..ncG);



  G2:=linalg[stack](Maux,G2);

fi;

G2:=bbtrig(evalm(G2),'colop','HG2','rG2');



rang2:=eval(rG2);
ull1:=linalg[rowdim](G2)-rang2;
ull:=ull1;
dimw:=linalg[coldim](G2)-rang2;

ndimw:=dimw;

# si G2 est de rang maximal on sort de la procedure avec false comme valeur

if dimw=0 then RETURN(false) fi;

# stockage des changements de colonnes

chgcol:=colop;
chgcolin:=eval(colop);


G1:=linalg[submatrix](G1,1..nlGM-delta,1..ncG);


if (nops(chgcolin)<>0) then G1:=evalm(matderoule(evalm(G1),chgcolin)) fi;


G1:=linalg[stack](G1,G2);
newG:=evalm(G1);

nR:=linalg[submatrix](G1,1..linalg[rowdim](G1)-ull1,rang2+1..ncG);


newR:=evalm(nR);


# construction newH, la matrice de triangularisation

newH:=constnewH2(evalm(H),evalm(HG2),delta);

nGM:=evalm(linalg[stack](GM,linalg[matrix](linalg[rowdim](G2)-delta,linalg[coldim](GM),0)));


nGM:=evalm(linalg[concat](nGM,G1));
```

```
nGM;

end:

#13###########################################################################

#resolution du systeme
# rend une base des solutions

sol:=proc(GM,ns,dimsol)

local zero,res,V,i,rang;


res:=linalg[kernel](evalm(GM),'dimsol');




res;


end:

#14###########################################################################

#procedure appliquant  la matrice de triangularisation a G1

HG1:=proc(G1,H)

local nG1;


nG1:=evalm(H &* G1);


end:



#15###########################################################################

trigGM:=proc(GM,H)


local nGM,nbc,nbl,GMI,I;


nbl:=linalg[rowdim](GM);
nbc:=linalg[coldim](GM);

I:=array(1..nbl,1..nbl,identity);
GMI:=linalg[concat](GM,evalm(I));


nGM:=linalg[gausselim](GMI);


H:=linalg[delcols](evalm(nGM),1..nbc);
nGM:=linalg[delcols](evalm(nGM),nbc+1..nbc+nbl);
```

```
end:



#16########################################################################

initT:=proc(GM,rGM,G)

local T,nG;

if (linalg[coldim](GM) > rGM) then

  T:=linalg[delcols](GM,rGM+1..linalg[coldim](GM));
  G:=linalg[submatrix](GM,1..rGM,rGM+1..linalg[coldim](GM));
else
  G:=false;
  T:=GM;

fi;

if rGM<linalg[rowdim](GM) then

  T:=linalg[submatrix](T,1..rGM,1..rGM);

fi;



T;

end:


#17########################################################################

# cette procedure permet de coller deux vecteurs (identique a linalg[stack] pour les matrices)

colv:=proc(v1,v2)

local V,n1,n2,i;

n1:=linalg[vectdim](v1);
n2:=linalg[vectdim](v2);

V:=linalg[vector](n1+n2);

for i to n1 do
   V[i]:=v1[i]
od;
for i from n1+1 to n2+n1 do
   V[i]:=v2[i-n1]
od;

evalm(V);

end:


#18########################################################################

# construction de la nouvelle matrice de triangularisation

constnewH2:=proc(H,HG2,delta)


local nH,zero1,zero3,zero2,I1,I2,nHG2,nbl,nblHG2,zonet,nH1,nH2,oH1,oH2,oG1,oG2,res;
```

```
nbl:=linalg[rowdim](H);
nblHG2:=linalg[rowdim](HG2);


if delta=0 then

zero1:=linalg[matrix](nblHG2,nbl,0);
zero2:=linalg[matrix](nbl,nblHG2,0);

nH1:=evalm(linalg[stack](H,zero1));
nH2:=evalm(linalg[stack](zero2,HG2));

res:=linalg[concat](nH1,nH2);


else


oH1:=linalg[submatrix](H,1..nbl-delta,1..nbl);
oH2:=linalg[submatrix](H,nbl-delta+1..nbl,1..nbl);

oG1:=linalg[submatrix](HG2,1..nblHG2,1..delta);
oG2:=linalg[submatrix](HG2,1..nblHG2,delta+1..nblHG2);

zero2:=linalg[matrix](nbl-delta,nblHG2-delta,0);


zonet:=evalm(oG1 &* oH2);


nH:=linalg[stack](oH1,zonet);


nG:=linalg[stack](zero2,oG2);


res:=linalg[concat](nH,nG);

fi;


evalm(res);

end:


#19##########################################################################



# procedure realisant la triangularisation de la matrice en faisant en
# plus des changements de colonnes pour obtenir directement la matrice
# GM sous la forme voulue.
# on retourne aussi le rang rg de la matrice .
# la matrice de triangularisation est fH et les changements de colonnes
# sont stockes dans changecol.

bbtrig:=proc(A,changecol,fH,rg)


local B,colop0,nl,m1,i,H,j,piv,r,k,bool,I,m,lim,l,temp,c,M,vectemp,matemp;

  nl:= linalg[rowdim](A);
  m1:= linalg[coldim](A);
  colop0:=[];
```

```
r:=1;
k:=1;
bool:=false;
I:=array(1..nl,1..nl,identity);
B:=linalg[concat](A,evalm(I));


m:= linalg[coldim](B);



while (k<=m1) and (r <= nl) and (not bool) do

 for i from r to nl  while B[i,k]=0 do od;

    if i <= nl then
    lim := abs(B[i,k]);
        for j from i+1 to nl do
            l:= abs(B[j,k]);
            if l=0 then next fi;


            if l>lim then lim:= l; i:= j; fi;
        od;

         if r<>i then

         # Le pivot est dans la ligne i, on inverse donc les lignes i et r

            for j from k to m do
                  temp:= B[i,j]; B[i,j]:= B[r,j]; B[r,j]:= temp;
            od;

        fi;


        for i from r+1 to nl do
               if B[i,k] <> 0 then
                    c:=B[i,k]/B[r,k];

                   for j from k+1 to m do
                      B[i,j]:=eval(B[i,j]-c*B[r,j]);
                   od;
                   B[i,k]:= 0;
               fi;
        od;

        r:=r+1;
        k:=k+1;


    else
bool:=linalg[iszero](linalg[submatrix](B,r..nl,k..m1));
if not bool then
           M:=linalg[delcols](B,1..m1);
           B:=linalg[submatrix](B,1..nl,1..m1);

   vectemp:=linalg[col](B,k);
   B:= linalg[delcols](B,k..k);
           B:=linalg[concat](B,vectemp);

           B:=linalg[concat](B,M);
           colop0:=[k,op(colop0)];

fi;
    fi;
```

```
   od;

   changecol:=colop0;
   print(`chgcols`,changecol);
   M:=linalg[delcols](B,1..m1);
   B:=linalg[submatrix](B,1..nl,1..m1);
   fH:=evalm(M);
   rg:=k-1;

   RETURN(eval(B));
end:



#20##########################################################################

# retablit les vecteurs solutions dans leur forme normale (sans permutation de colonnes)
# pour le premier tour

deroul:=proc(vect,colop0)

local i,ml,ml1,j,temp;

ml:=convert(vect,list);


for i to nops(colop0) do
   temp:=ml[nops(ml)];
   ml1:=subsop(nops(ml)=NULL,ml);
   ml:=[seq(ml1[j],j=1..colop0[i]-1),temp,seq(ml1[j],j=colop0[i]..nops(ml1))];
od;


ml:=convert(ml,vector);

evalm(ml);

end:

#21#########################################################################

# retablit les vecteurs solutions dans leur forme normale (sans permutation de colonnes)

nderoul:=proc(vect,colop0,DS)

local i,j,ml,ml1,k,l,point,h,temp;

ml:=convert(vect,list);
k:=nops(ml);
l:=nops(DS);

point:=0;
for h to nops(colop0) do


   for i to nops(colop0[h]) do

      temp:=ml[k-point];
      ml1:=subsop(k-point=NULL,ml);
      ml:=[seq(ml1[j],j=1..colop0[h][i]-1),temp,seq(ml1[j],j=colop0[h][i]..nops(ml1))];

   od;

point:=point+n*DS[l-h+1];

od;

ml:=convert(ml,vector);
```

```
end:


#22###########################################################################

# les changement de colonnes effectues sur G2 doivent etre repercutes sur G1
# c'est ce que fait cette procedure

matderoule:=proc(M,invcol)

local i,nM,nlM,ncM,temp;

nlM:=linalg[rowdim](M);
ncM:=linalg[coldim](M);
nM:=M;


 for i by -1 from nops(invcol) to 1 do

      temp:=linalg[col](nM,invcol[i]);
      nM:=linalg[delcols](nM,invcol[i]..invcol[i]);
      nM:=linalg[concat](nM,evalm(temp));


 od;

evalm(nM);

end:




#23###########################################################################


# insere dans les vecteur solution les zeros correspondant au palliers de la matrice GM

goodvect:=proc(skelet1,skelet2,vect)

local i,vect1,j;


vect1:=linalg[vector](1,vect[1]);

for j from 2 to skelet1[1] do
      vect1:=colv(vect1,linalg[vector](1,vect[j]));
   od;
if nops(skelet1)>1 then
   vect1:=colv(vect1,linalg[vector](skelet2[1],0));
fi;

for i from 1 to nops(skelet1)-1 do

   for j from skelet1[i]+1 to skelet1[i+1] do
      vect1:=colv(vect1,linalg[vector](1,vect[j]));
   od;


   if  nops(skelet1)>i+1 then
   vect1:=colv(vect1,linalg[vector](skelet2[i+1],0));
   fi;
od;

evalm(vect1);
```

```
end:


#24#########################################################################


# procedure extrayant la matrice T


constT:=proc(T,G,l2cut,dw)
local pT,pG,nlT,ncT,nlG,ncG;



ncT:=linalg[coldim](T);
nlT:=linalg[rowdim](T);
nlG:=linalg[rowdim](G);
ncG:=linalg[coldim](G);


if l2cut>0 then
pG:=linalg[delrows](G,nlG-l2cut+1..nlG);
pG:=linalg[delcols](pG,ncG-dw+1..ncG);
else

pG:=linalg[delcols](G,ncG-dw+1..ncG);
fi;

pT:=linalg[stack](T,linalg[matrix](nlG-l2cut-nlT,ncT,0));

pT:=linalg[concat](pT,pG);

evalm(pT);

end:



#25#########################################################################

# vect = R . [00000|0000000|0000000000|1000]


resol:=proc(M,vect)


local t,res;

res:=evalm(M &* vect);

evalm(res);
end:
```