DEA de mathématiques

Mémoire

Résolution de systèmes d'équations polynomiales et applications.

Hélène PRIETO

Encadreur : Bernard MOURRAIN

Organisme: Projet SAGA, INRIA

21 juin 1999

Résumé

Dans ce mémoire nous nous intéressons aux méthodes de recherche des points isolés d'une variété algébrique définie par un système de n polynômes en n variables. Plus précisemment sont étudiées des méthodes basées sur une approche matricielle à partir du bézoutien du système. Nous commencons par rappeler comment la résolution d'un système d'équations polynomiales peut se ramener à un problème de valeurs propres. Nous présentons ensuite les propriétés du Bézoutien permettant ces approches. Différentes méthodes de résolution sont ensuite détaillées, certaines demandant l'ajout d'une forme linéaire au système initial, les autres considérant une variable comme faisant partie du corps des coefficients. Les résolutions étudiées portent sur des calculs de représentation univariée rationnelle des solutions, des calculs matriciels, et sur des perturbations du système initial. Deux problèmes pratiques de chimie et robotique, ainsi que la formulation algébrique qui en découle, sont brièvement exposés, ainsi que, pour le problème de chimie, les résultats obtenus en appliquant une des méthodes de résolution étudiée dans le cadre de ce mémoire. Un environnement informatique pour le calcul symbolique et numérique, la librairie **ALP**, Algèbre Linéaire et Polynômes, développée par le projet **SAGA** de l'INRIA/Sophia-Antipolis est ensuite présenté et illustré par des exemples d'utilisation d'algorithmes.

Mots-clé : équations polynomiales, matrices de multiplication, calcul matriciel, résultant, Bézoutien, matrice de Bézout, Forme de Chow, valeurs et vecteurs propres, perturbation, nombres infinitésimaux.

Table des matières

1 Introduction **5**

I Une méthode de résolution par le bézoutien **7**

1 Points isolés d'une variété algébrique **9**

1.1 Définitions - Propriétés . . . 9

1.2 Réduction de la résolution de systèmes polynomiaux à un problème de valeurs propres . . . 10

2 Résultants **13**

3 Bézoutien - Cas multivariable **15**

3.1 Définitions - Propriétés . . . 15

3.2 Bézoutien et résultant . . . 16

4 Représentation univariée rationnelle **17**

4.1 Forme de Chow . . . 17

4.2 Représentation rationnelle univariée . . . 17

4.3 Matrice de Bézout et Forme de Chow . . . 21

5 Résolution d'un système polynomial bien-contraint par une méthode basée sur le résultant **23**

5.1 Résolution par u-résultant et représentation univariée rationnelle . . . 23

5.2 Résolution par u-résultant et calcul matriciel . . . 23

5.3 Variable cachée et perturbation du système . . . 24

5.3.1 Perturbation du système . . . 24

5.3.2 Valeurs des racines en la variable cachée . . . 25

5.3.3 Calcul de toutes les coordonnées des racines . . . 26

5.3.4 Un algorithme . . . 26

6 Types de problème à résoudre **27**

6.1 Configurations de molécules . . . 27

6.1.1 Problème . . . 27

6.1.2 Formulation algébrique . . . 28

6.1.3 Résolution - Résultats . . . 28

6.2 Robot parallèle . . . 29

6.2.1 Problème . . . 29

6.2.2 Formulation algébrique . . . 30

II Environnement pour le calcul symbolique et numérique **33**

1 Introduction **35**

2 Exemples de méthodes combinant calcul symbolique et calcul numérique **37**
2.1 Les méthodes des puissances classique et inverse 37
2.2 Méthode de Sebastio et Silva 38
2.3 Méthode inverse implicite 39

3 Implémentation de l'environnement **41**
3.1 Choix du C++ 41
3.1.1 Objectifs 41
3.1.2 Intérêts du C++ 41
3.2 Structures de données 42
3.2.1 Approche 42
3.2.2 Architecture 43

4 Les ingrédients des algorithmes **45**
4.1 Exemple de code générique 45
4.2 Utilisation de librairie externe : GMP 46
4.3 Généricité et efficacité 47
4.4 Structure à deux étages : calcul de Bézoutien 48
4.5 Mélange de calculs formels et numériques : les équations de Kruppa 50
4.6 Connexion avec la librairie Lapack 51
4.7 Calcul de valeurs propres 52

Chapitre 1

Introduction

Deux approches sont généralement utilisées pour le calcul des racines d'un système polynomial de dimension zéro. La première est basée sur le calcul d'une base de Groebner de l'algèbre quotient définie par le système. Ce calcul permet d'obtenir, à partir d'un ensemble de polynômes, un autre ensemble "plus simple" de polynômes définissant le même idéal, mais à partir duquel la forme normale d'un polynôme donné est facilement calculable. Le problème de cette approche est qu'il s'agit d'un calcul exact pour lequel de grands nombres apparaissent généralement. Adapter ces méthodes à de l'arithmétique flottante, les coefficients étant alors donnés avec une certaine erreur, n'est pas facile. Un autre approche est basée sur les formulations de résultants qui peuvent être calculés en arithmétique flottante. Cependant, ceci ne s'applique que dans des situations génériques et n'est donc pas nécessairement adapté aux problèmes pratiques rencontrés comme en robotique, vision ou chimie par exemple. Les méthodes étudiées ici réexaminent l'approche basée sur le calcul de résultant mais d'un point de vue d'algèbre linéaire et donc de calculs matriciels. La plupart de ces approches étendent la réduction classique de résolution à un problème de valeurs et vecteurs propres. Les matrices considérées ici sont les matrices de **Bézout** (c.f. [3]) d'un système définissant une variété algébrique dont nous voulons calculer les points isolés. Ces matrices, comme les matrices de résultants de **Macaulay** (c.f. [18]) ou **Dixon** (c.f. [10]) permettent d'obtenir un multiple du résultant de $n+1$ polynômes en n variables. Le résultant d'un système donnant une conditon nécessaire et suffisante sur les coefficients des polynômes pour que ceux-ci aient des racines communes dans l'espace projectif associé. Nous verrons comment exploiter et adapter cette propriété pour un système, qui n'est pas forcément "générique", de n polynômes en n variables. En particulier, nous verrons comment ramener ce système "bien-contraint" (autant d'équations que de variables) à un système "sur-contraint" (ici, une équation de plus que le nombre de variables) afin de pouvoir construire une matrice de résultant. Cette transformation peut se faire par ajout d'une forme linéaire au système étudié ou en "cachant" une variable dans le corps des coefficients. Le résolution peut alors se faire, selon les cas, par calcul d'une représentation univariée rationnelle, ce qui permet de ramener le problème à la résolution d'une équation polynomiale en une variable, par recherche de valeurs et/ou vecteurs propres. Nous nous intéressons aussi à des perturbations du système, c'est-à-dire qu'après avoir donné une direction, le repère est "bougé", ce qui permet de retrouver les coordonnées des racines.
Un autre intérêt des matrices de résultants telles que les matrices de Bézout est que les calculs effectués ne se font que sur un nombre limité de monômes, nombre que l'on sait borner. Par exemple pour les matrices de Bézout et Macaulay, ce nombre ne peut pas dépasser le nombre de monômes de degré $\leq \sum_i d_i - n$, où d_i est le degré du $i^{ème}$ polynôme et n le nombre de variables. Alors que le calcul de bases de Groebner nécessite des polynômes intermédiaires de hauts degrés. Ainsi d'un point de vue technique, en appliquant un algorithme traduisant les méthodes présentées ici, la taille des matrices est réduite et l'espace mémoire occupé aussi. Ceci contrairement au calcul de bases de Groebner.

La seconde partie de ce mémoire est consacrées à la présentation d'un environnement pour le calcul symbolique et numérique, portant sur la librairie **ALP** (c.f. [19]) développée par le projet **SAGA** de l'INRIA/Sophia-Antipolis. Cet environnement permet d'appliquer et de mettre en place les méthodes de résolution basées, entre autres, sur le calcul matriciel. De plus il permet la combinaison de méthodes de calcul exact et de méthodes numériques.

Première partie

Une méthode de résolution par le bézoutien

Chapitre 1

Points isolés d'une variété algébrique

1.1 Définitions - Propriétés

Soit $R = \mathbb{K}[x_1, \dots, x_n]$ l'algèbre des polynômes en n variables, $\mathbf{x} = (x_1, \dots, x_n)$, sur un corps $\mathbb{K}$ que l'on supposera algébriquement clos. Soient $f_1, \dots, f_m$, m polynômes de R définissant le système polynomial $f_1(\mathbf{x}) = 0, \dots, f_m(\mathbf{x}) = 0$, et I l'idéal de R engendré par ces m polynômes, $I = (f_1, \dots, f_m)$.

On note $\mathcal{A} = \mathbb{K}[\mathbf{x}]/I$, l'algèbre quotient, et $\equiv$ l'égalité dans $\mathcal{A}$. Soit $I = Q_1 \cap \dots Q_d \cap Q'_{d+1} \cap \dots \cap Q'_k$ une décomposition primaire minimale de I, telle que les Q_i correspondent aux premiers isolés de dimension 0 et les Q'_j aux autres. Ce qui revient à dire, si $\mathcal{Z}(I) = \{\zeta \in \mathbb{K}^n ; f_1(\zeta) = 0, \dots, f_m(\zeta) = 0\}$, que les variétés $\mathcal{Z}(Q_i)$ sont les composantes de dimension zéro de $\mathcal{Z}(I)$ et les variétés $\mathcal{Z}(Q'_j)$ ses composantes immergées ou de dimension ≥ 1. Donc, si $\mathbf{m}_{\zeta_i}$ est l'idéal maximal définissant le point ζ_i de $\mathbb{K}^n$, l'idéal Q_i est $\mathbf{m}_{\zeta_i}$-primaire.
Soit $\mathcal{Z}^0(I) = \{\zeta_1, \dots, \zeta_d\}$ l'ensemble des racines isolées de $\mathcal{Z}(I)$, $I_0 = Q_1 \cap \dots \cap Q_d$ et $I^+ = Q'_{d+1} \cap \dots \cap Q'_k$. Si $I^+ = 0$ alors $\mathcal{Z}(I) = \mathcal{Z}^0(I)$ est de dimension 0.

Proposition 1.1.1 *Soit $\mathcal{A}_i = (0 : Q_i) = \{f \in R \mid fQ_i \subset 0\}$, $i = 1, \dots, d$ et $\mathcal{A}^+ = (0 : i$ des idéaux de $\mathcal{A}$. Alors on a la décomposition :*

$$\mathcal{A} = \mathcal{A}_1 \oplus \dots \oplus \mathcal{A}_d \oplus \mathcal{A}^+.$$

Preuve. Tout d'abord pour des idéaux Q, *et* Q' d'un anneau quelconque A, tels que $Q + Q' = A$, on a $(0 : Q) \cap (0 : Q') = (0 : A) = 0$ et $(0 : Q) + (0 : Q') = (0 : Q \cap Q')$ (voir [1]).

De $Q_i + Q_j = R$ si $i \neq j$ et $Q_i + I+ = R$ on peut déduire que :
$\mathcal{A}_1 + \dots + \mathcal{A}_d + \mathcal{A}^+ \equiv (0 : Q_1) + \dots + (0 : Q_d) + (0 : I^+) \equiv \mathcal{A}$.
De plus, cette somme est directe puisque $\mathcal{A}_i \cap (\mathcal{A}_{i+1} + \dots \mathcal{A}_d + \mathcal{A}^+) \equiv (0 : Q_i) \cap ((0 : Q_{i+1}) + \dots + (0 : Q_d) + (0 : I^+)) \equiv (0 : Q_i) + (Q_{i+1} \cap \dots \cap Q_d \cap I^+)) \equiv (0 : \mathcal{A}) \equiv 0$. On peut donc écrire :

$$\mathcal{A} = \mathcal{A}_1 \oplus \dots \oplus \mathcal{A}_d \oplus \mathcal{A}^+.$$

□

Ce dernier point entraine que pour $i \neq j$ on a $\mathcal{A}_i.\mathcal{A}_j \equiv 0$ et $\mathcal{A}_i.\mathcal{A}^+ \equiv 0$. Il existe donc un ensemble d'idempotents orthogonaux (c.f. [8]), $\mathbf{e}_1, \dots, \mathbf{e}_d$, $\mathbf{e}_+$ satisfaisant les propriétés suivantes :

$$\mathbf{e}_1 + \cdots + \mathbf{e}_d + \mathbf{e}_+ \equiv 1, \mathbf{e}_i \mathbf{e}_j \equiv \begin{cases} 0 \text{ si } i \neq j \\ \mathbf{e}_i \text{ si } i = j \end{cases}, \text{ et} \begin{cases} \mathbf{e}_i \mathbf{e}_+ = 0, \\ \mathbf{e}_+ \mathbf{e}_+ = \mathbf{e}_+. \end{cases}$$

et tels que $\mathcal{A}_i = \mathbf{e}_i\mathcal{A}$, $\mathcal{A}^+ = \mathbf{e}_+\mathcal{A}$. $\mathbf{e}_i\mathcal{A} \sim R/Q_i$ est une algèbre locale d'idéal maximal $\mathbf{m}_{\zeta_i}$. Pour tout $\zeta \in \mathcal{Z}^0(I)$, on note $\mathbf{e}_\zeta$ l'idempotent associé, $\mathcal{A}_\zeta = \mathbf{e}_\zeta\mathcal{A}$ la sous-algèbre associée et $\mathcal{A}^0 = \mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_d$ la sous-algèbre de $\mathcal{A}$ correspondant à $\mathcal{Z}^0(I)$.

On note $\widehat{R}$ l'espace dual de R, et $\widehat{\mathcal{A}}$, le dual de $\mathcal{A}$ défini par $\widehat{\mathcal{A}} = \{\Lambda \in \widehat{R},\ \Lambda(I) = 0\}$ (parfois noté $I^\perp$). Comme éléments simples de $\widehat{R}$, on peut citer l'évaluation de polynômes en des points de $\mathbb{K}^n$:

$$\begin{array}{rcl} \mathbf{1}_\zeta : R & \to & \mathbb{K} \\ p & \mapsto & p(\zeta) \end{array}$$

Nous pouvons remarquer que $\mathbf{1}_\zeta \in \widehat{\mathcal{A}}$ si et seulement si $\zeta \in \mathcal{Z}(I)$.

1.2 Réduction de la résolution de systèmes polynomiaux à un problème de valeurs propres

Pour tout $a \in \mathcal{A}$ on pose :

$$\begin{array}{rcl} M_a : \mathcal{A} & \to & \mathcal{A} \\ b & \mapsto & a\,b \end{array}$$

qui définit l'opérateur de multiplication par a dans $\mathcal{A}$, et

$$\begin{array}{rcl} M_a^{\mathtt{t}} : \widehat{\mathcal{A}} & \to & \widehat{\mathcal{A}} \\ \Lambda & \mapsto & a \cdot \Lambda \end{array}$$

l'application $\mathbb{K}$-linéaire induite par M_a.

La matrice de M_a^t dans la base duale de $\widehat{\mathcal{A}}$ est la transposée de la matrice de M_a. Elles ont donc les mêmes valeurs propres.

La résolution des systèmes polynomiaux par des recherches de valeurs et vecteurs propres est basée sur le théorème suivant (pour plus de détails et la preuve voir [2] p.24) :

Théorème 1.2.1 *On suppose $\mathcal{Z}(I)$ de dimension 0.*

1. *Les valeurs propres de l'opérateur lineaire M_p (resp. M_p^t) sont $\{p(\zeta_1), \ldots, p(\zeta_d)\}$.*
2. *Les vecteurs propres communs à tous les endomorphismes M_p^t, et donc en particulier aux $(M_{x_i}^t)$ sont (à un facteur près) $\mathbf{1}_{\zeta_1}, \ldots, \mathbf{1}_{\zeta_d}$.*
3. *Si $m = n$, les vecteurs propres communs aux (M_{x_i}) sont (à un facteur près) $J(\mathbf{x})\mathbf{e}_1, \ldots, J(\mathbf{x})\mathbf{e}_d$, où $J(\mathbf{x})$ est le jacobien de $p_1, \ldots, p_n$.*

Nous savons (1.1.1) qu'il existe une base de $\mathcal{A}$ dans laquelle, pour tout $a \in \mathcal{A}$, la matrice $\mathtt{M}_a$ de l'endomorphisme M_a, a la forme :

$$\mathtt{M}_a = \begin{pmatrix} \mathtt{M}_{a,1} & & 0 \\ & \ddots & \\ 0 & & \mathtt{M}_{a,d} \end{pmatrix}$$

où les $\mathtt{M}_{a,i}$ sont les matrices de multiplication par a dans les algèbres locales $\mathcal{A}_i$ et ont la forme suivante :

$$\mathtt{M}_{a,i} = \begin{pmatrix} a(\zeta_i) & & * \\ & \ddots & \\ 0 & & a(\zeta_i) \end{pmatrix}$$

Dans le cas d'une racine simple, on a $\mathtt{M}_{a,i} = (a(\zeta_i))$.

Pour tout couple $a,b \in \mathcal{A}$,

$$M_a^{\mathtt{t}}(\mathbf{1}_{\zeta_i})(b) = \mathbf{1}_{\zeta_i}(a\,b) = a(\zeta_i)b(\zeta_i) = a(\zeta_i)\mathbf{1}_{\zeta_i}(b),$$

et $\mathbf{1}_{\zeta_i}$ est un vecteur propre de $M_a^{\mathtt{t}}$, associé à la valeur propre $a(\zeta_i)$.

Pour une racine simple, ζ_i, le vecteur propre de $M_a^{\mathtt{t}}$ associé à la valeur propre $a(\zeta_i)$ est $\mathbf{1}_{\zeta_i}$ dont les coordonnées dans la base duale d'une base monomiale $(\mathbf{x}^{\alpha_1},\ldots,\mathbf{x}^{\alpha_D})$ de $\mathcal{A}$ sont $(\zeta_i^{\alpha_1},\ldots,\zeta_i^{\alpha_D})$, c'est à dire l'évaluation de cette base de $\mathcal{A}$ en la racine ζ_i.

Ceci nous conduit à l'algorithme suivant pour le calcul de racines simples d'un système polynomial.

1.2.1. **Algorithme de calcul des racines simples d'un système polynomial $f_1 = \cdots = f_m = 0$.**

1. Calculer la transposée de la matrice de multiplication $\mathtt{M}_{x_1}$, par exemple.
2. Calculer ses vecteurs propres $\mathbf{v}_i = (v_{i,1}, v_{i,x_1}, \ldots, v_{i,x_n}, \ldots)$ for $i = 1, \ldots, d$.
3. Pour $i = 1, \ldots, d$, calculer
$$\zeta_i = (\frac{v_{i,x_1}}{v_{i,1}}, \ldots, \frac{v_{i,x_n}}{v_{i,1}}).$$

On obtient ainsi les racines simples du système, ce qui en pratique est souvant suffisant.

La principale difficulté de cette méthode est de calculer la matrice de multiplication par une variable x_i dans l'algèbre $\mathcal{A}$. De plus, pour calculer les points isolés d'une variété de dimension ≥ 1, il est nécessaire de se ramener à un problème en dimension 0. Une autre difficulté étant lors de racines multiples, l'espace de vecteurs propres n'étant pas de dimension 1.

Chapitre 2

Résultants

Comme dans la section précédente, nous considérons $R = \mathbb{K}[x_1, \ldots, x_n]$ l'algèbre des polynômes en n variables $\mathbf{x} = (x_1, \ldots, x_n)$ sur un corps $\mathbb{K}$. Soient $f_1, \ldots, f_m$, m polynômes de R définissant le système polynomial $f_1(\mathbf{x}) = 0, \ldots, f_m(\mathbf{x}) = 0$ et $\mathcal{Z}(I) = \{\zeta \in \mathbb{K}^n; f_1(\zeta) = \ldots = f_m(\zeta) = 0\}$, l'ensemble des racines du système.

Voici quelques définitions (pour plus de détails, voir [14]) :

Définition 2.0.1 *Les variétés projectives définies par une seule équation homogène sont appelées hypersurfaces.*

Soient $n+1$ polynômes

$$\left\{ \begin{array}{lcl} f_0(\mathbf{x}) & = & \sum_{j=0}^{k_0} c_{0,j}\, \psi_{0,j}(\mathbf{x}) \\ & \vdots & \\ f_n(\mathbf{x}) & = & \sum_{j=0}^{k_n} c_{n,j}\, \psi_{n,j}(\mathbf{x}) \end{array} \right.$$

où

- $\mathbf{c} = (c_{i,j})$ sont des paramètres
- $\mathbf{x}$ est un point de la variété projective $X \subset \mathbb{P}^N$, de dimension n,
- Les fonctions $\mathcal{L}_i(\mathbf{x}) = (\psi_{i,j}(\mathbf{x}))_{j=1,\ldots,k_i}$ sont indépendantes des paramètres $\mathbf{c}$, et sont des fonctions régulières sur X.
- Pour des valeurs génériques de c, le système n'a pas de solution sur X.

Le problème de la théorie de l'élimination est, dans le cas décrit ci-dessus, de trouver des conditions nécessaires et suffisantes sur les paramètres $\mathbf{c} = (c_{i,j})$ pour que les équations $f_0 = 0, \ldots, f_n = 0$ aient une racine commune dans X.
Dans le cas classique, $\mathcal{L}_i(\mathbf{x})$ est le vecteur de tous les monômes de degré d_i et $X = \mathbb{P}^n$. La condition nécessaire et suffisante sur les $\mathbf{c} = (c_{i,j})$ pour que les polynômes homogènes $f_0, \ldots, f_n$ aient une racine commune dans $X = \mathbb{P}^n$ est $\mathrm{Res}_{\mathbb{P}^n}(f_0, \ldots, f_n) = 0$ où $\mathrm{Res}_{\mathbb{P}^n}$ est le *résultant projectif classique*.
D'un point de vue géométrique, nous regardons l'ensemble des paramètres $\mathbf{c} = (c_{i,j})$ tels qu'il existe $x \in X$ avec $\sum_j c_{i,j}\psi_{i,j}(\mathbf{x}) = 0$ pour $i = 0, \ldots, n$. C'est à dire que $\mathbf{c}$ est la projection du point $(\mathbf{c},\mathbf{x})$ de la *variété incidente*

$$W_X = \{(\mathbf{c},\mathbf{x}) \in \mathbb{P}^{k_0} \times \cdots \times \mathbb{P}^{k_n} \times X \; ; \; \sum_{j=0}^{k_i} c_{i,j}\psi_{i,j}(\mathbf{x}) = 0 \; ; \; i = 0, \ldots, n\}.$$

On note

$$\begin{aligned} \pi_1 : W_X &\rightarrow \mathbb{P}^{k_0} \times \cdots \times \mathbb{P}^{k_n}, \\ \pi_2 : W_X &\rightarrow X. \end{aligned}$$

les deux projections naturelles. L'image de W_X par π_1 est précisemment l'ensembles des paramètres pour lesquels le système a une racine. L'image par π_2 d'un point de W_X est une solution dans X du système associé. On a alors la définition suivante (voir [14]) :

Définition 2.0.2 *Soit $Z = \pi_1(W_X)$. Si Z est une hypersurface, alors son équation (unique à un facteur près) est appelée le* résultant *de $f_0, \ldots, f_n$ sur X. Il est noté* $\mathrm{Res}_X(f_0, \ldots, f_n)$.

Afin d'être dans cette situation, certaines conditions sont imposées (voir [13]) :

Conditions 2.0.2

1. *X est une variété projective irréductible.*
2. *Les fonctions $\mathcal{L}_i, i = 0, \ldots, n$, ne sont pas identiquement nulles sur X.*
3. *Pour des valeurs génériques de $\mathbf{c}$, le système $f_0, \ldots, f_n$ n'a pas de solution dans X, et n de ces équations ont un nombre fini de solutions communes.*

Le système $f_0 = \ldots = f_n = 0$ a donc une solution dans X si et seulement si $\mathrm{Res}_X(f_0, \ldots, f_n) = 0$, ceci généralise, sous les conditions 2.0.2, la définition du résultant classique sur $\mathbb{P}^n$ à d'autres variétés projectives.

Les méthodes de calcul effectif du résultant s'appuient sur la construction de matrices dont le déterminant va nous fournir le résultant ou un multiple non-trivial. Ces matrices de résultants peuvent se regrouper en deux grandes familles : les matrices de type *Sylvester* généralisant la construction de Sylvester (1835, voir [22]) au cas multivariable et les matrices de type *Bézout* généralisant la construction d'E. Bézout (1779, voir [3]) au cas multivariable. Cette dernière construction est présentée dans le chapitre suivant.

Chapitre 3

Bézoutien - Cas multivariable

On considère toujours les polynômes $f_0,\dots,f_n$ de $R=\mathbb{K}[\mathbf{x}]$, où $\mathbf{x}=(x_1,\dots,x_n)$.

3.1 Définitions - Propriétés

Voici quelques définitions et propriétés de la théorie des Bézoutiens (pour plus de détails voir [6, 13]) et qui généralise la construction de E. Bézout au cas univariable.

On ajoute à l'ensemble des variables $\mathbf{x}$ un nouvel ensemble de variables $\mathbf{y}=(y_1,\dots,y_n)$ et on note $\mathbf{x}^{(0)}=\mathbf{x}$, $\mathbf{x}^{(1)}=(y_1,x_2,\dots,x_n),\dots,\ \mathbf{x}^{(n)}=\mathbf{y}$.

Pour un polynôme $q\in R$, on définit $\theta_i(p)=\frac{p(\mathbf{x}^{(i)})-p(\mathbf{x}^{(i-1)})}{y_i-x_i}$.

Définition 3.1.1 *Pour $n+1$ polynômes $f=(f_0,f_1,\dots,f_n)\in R$, le* Bézoutien *de $f_0,f_1,\dots,f_n$ est un polynôme en $\mathbf{x}$ et $\mathbf{y}$ défini par:*

$$\Theta(f_0,f_1,\dots,f_n)=\Theta_f(f_0)=\det\begin{pmatrix} f_0(\mathbf{x}) & \theta_1(f_0) & \cdots & \theta_n(f_0)\\ \vdots & \vdots & & \vdots\\ f_n(\mathbf{x}) & \theta_1(f_n) & \cdots & \theta_n(f_n)\end{pmatrix}=\sum_{i,j}\theta^f_{i,j}\,\mathbf{x}^{u_i}\,\mathbf{y}^{v_j},\qquad (3.1)$$

C'est un polynôme de degré au plus $\sum_{i=0}^n deg(f_i)-n$.

Dans les bases monomiales $\mathbf{x}^\alpha$ et $\mathbf{y}^\beta$, Θ_f s'écrit :

$$\Theta_f(f_0)=\sum_{\alpha\in E,\beta\in F}t_{\alpha,\beta}\mathbf{x}^\alpha\mathbf{y}^\beta$$

.

Définition 3.1.2 *L'ensemble des monômes $\mathbf{x}^\alpha$ (resp. $\mathbf{y}^\beta$) est appelé le* $\mathbf{x}$-support *(resp.* $\mathbf{y}$-support*) de $\Theta_f(f_0)$.*

Soit maintenant l'application :

$$\begin{array}{rcl}\Phi_f(f_0):\widehat{R} & \to & R\\ \Lambda & \mapsto & \displaystyle\sum_{\alpha\in E,\beta\in F}t_{\alpha\,\beta}\mathbf{x}^\alpha\Lambda(\mathbf{y}^\beta).\end{array}$$

Définition 3.1.3 *On note* $\mathtt{B}_{f_0}(f)$*, la matrice de* $\Phi_f(f_0)$ *de la base duale de* $\widehat{R}$ *dans la base monomiale* $(\mathbf{x}^\alpha)$ *de* R*. Cette matrice est la* matrice de Bézout*, ou* Bézoutienne*, des polynômes* $f_0,f_1,\ldots,f_n$*.*

Voici deux propositions dont les détails peuvent se lire dans [13].

Proposition 3.1.4 *Le* $\mathbf{x}$*-support (resp.* $\mathbf{y}$*-support) de* $\Theta_f(1)$ *est un ensemble générateur de* $\mathcal{A}$*.*

Proposition 3.1.5 *Pour toute application polynomiale* $\mathbf{f} = (f_0,\ldots,f_n)$*,* $\Theta(f_0,\ldots,f_n) = f_0\Theta(1,f_1,\ldots,f_n)+ f_1\Theta(f_0,1,\ldots,f_n) + \ldots + f_n\Theta(f_0,f_1,\ldots,f_{n-1},1)$*.*

3.2 Bézoutien et résultant

On utilise les résultats suivants (voir [14]) :

Lemme 3.2.1 *Si* $\mathcal{A}_0$ *est l'algèbre quotient associée aux points isolés de la variété définie par l'idéal* $(f_1,\ldots,f_n)$ *et si* d *est sa dimension sur* $\mathbb{K}$ *alors il existe deux bases* $\mathbf{v} = (v_i)_{i\in\mathbb{N}}$ *et* $\mathbf{w} = (w_i)_{i\in\mathbb{N}}$ *de* R *telles que leurs images soient des bases de* $\mathcal{A}_0$*, et* $v_i,w_i \in I_0$ *pour* $i > D$*. Et, pour tout* $f_0 \in R$*, la matrice* $\mathtt{B}_{f_0}$ *est de la forme :*

$$\mathtt{B}_{f_0} = \left(\begin{array}{c|c} M_{f_0} & 0 \\ \hline 0 & L_{f_0} \end{array}\right) \tag{3.2}$$

où $\mathtt{M}_{f0}$ *est la matrice de multiplication par l'image de* f_0 *dans la base* $(v_1,\ldots,v_D)$ *de* $\mathcal{A}_0$*.*

Théorème 3.2.2 *On suppose les conditions 2.0.2 satisfaites. Alors tout mineur maximal non-nul de la matrice bezoutienne* $\mathtt{B}(f_0,\ldots,f_n)$ *est un multiple du résultant* $Res_X(f_0,\ldots,f_n)$*.*

Le lemme 3.2.1 nous permet d'extraire de la Bézoutienne du système étudié, la matrice de multiplication par un élément de R dans $\mathcal{A}$ dont les valeurs et vecteurs propres donnent, comme nous l'avons vu avec le theorème 1.2.1, les racines du système.

Chapitre 4

Représentation univariée rationnelle

Le but d'une représentation univariée rationnelle est de décrire les points isolés d'une variété algébrique comme étant les images par une application rationnelle des zéros d'un polynôme en une variable. Ceci peut se faire directement à partir des matrices de Bézout. On considère toujours un idéal $I = (f_1, \ldots, f_n)$, l'intersection, I_0, des composantes primaires isolées de I et $\mathcal{Z}_0(I)$ l'ensemble des points isolés de la variété définie par I.

4.1 Forme de Chow

Définition 4.1.1 *La forme de Chow de $\mathcal{Z}_0(f_1, \ldots, f_m) = \mathcal{Z}_0(I)$ est le polynôme des variables $\mathbf{u} = (u_1, \ldots, u_n)$:*

$$\mathcal{C}_{I_0}(\mathbf{u}) = \prod_{\zeta \in \mathcal{Z}_0(I)} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)^{\mu_\zeta}$$

où μ_ζ est la multiplicité de la racine $\zeta \in \mathcal{Z}_0(I)$.
La forme réduite de Chow, $\mathcal{C}^r_{I_0}(\mathbf{u})$ est la partie sans facteur carré (sans exposant μ_ζ) de $\mathcal{C}_{I_0}(\mathbf{u})$.

Proposition 4.1.2 *La forme de Chow de $\mathcal{Z}_0(I)$ est le déterminant de la multiplication par $u_0 + u_1x_1 + \ldots + u_nx_n$. C'est-à-dire le déterminant de $u_0\mathbb{I} + u_1\mathtt{M}_{x_1} + \ldots + u_n\mathtt{M}_{x_n}$, où les $\mathtt{M}_{x_i}$ sont les matrices de multiplication par les variables x_i dans l'algèbre quotient $\mathcal{A}_0 = R/I_0$.*

Preuve. Puisque les matrices de multiplication $\mathtt{M}_{x_i}$ commutent, elles peuvent se mettre sous forme triangulaire dans une même base. En particulier, elles se décomposent, diagonalement en blocs triangulaires $\mathtt{N}^\zeta_{x_i}$ pour $\zeta \in \mathcal{Z}_0(I)$. Le déterminant de $\mathtt{M}_{u_0+u_1x_1+\ldots+u_nx_n}$ est donc le produit des déterminants $det(u_0\mathbb{I} + u_1\mathtt{N}^\zeta_{x_1} + \ldots + u_n\mathtt{N}^\zeta_{x_n})$, pour $\zeta \in \mathcal{Z}_0(I)$. Les blocs $\mathtt{N}^\zeta_{x_i}$ étant triangulaires, de taille μ_ζ et les termes de la diagonale étant la $i^{ème}$ coordonnée de la racine ζ, on a :

$$det(\mathtt{M}_{u_0+u_1x_1+\ldots+u_nx_n}) = \prod_{\zeta \in \mathcal{Z}_0(I)} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)^{\mu_\zeta} \tag{4.1}$$

□

4.2 Représentation rationnelle univariée

Le théorème suivant permet de décrire les racines $\zeta \in \mathcal{Z}_0(I)$ comme les valeurs de fractions rationnelles en certaines racines d'un polynôme univariable :

Théorème 4.2.1 *Soit $\Delta(\mathbf{u})$ un multiple de la forme de Chow $\mathcal{C}_{I_0}(\mathbf{u})$. Alors, pour un choix générique de $t = (t_0, \ldots, t_n) \in \mathbb{K}^{n+1}$, nous avons*

$$\frac{\Delta}{pgcd(\Delta, \frac{\partial \Delta}{\partial u_0})}(t + \mathbf{u}) = d_0(u_0) + u_1 d_1(u_0) + \ldots + u_n d_n(u_0) + r(\mathbf{u}) \tag{4.2}$$

avec $r(\mathbf{u}) \in (u_1, \ldots, u_n)^2$, $pgcd(d_0, d_0') = 1$ et, pour tout $\zeta = (\zeta_1, \ldots, \zeta_n) \in Z_0(I)$, il existe ζ_0 racine de $d_0(u_0) = 0$ telle que

$$d_0'(\zeta_0)\zeta_i - d_i(\zeta_0) = 0, i = 1, \ldots, n. \tag{4.3}$$

Ce résultat n'implique pas que tout racine de $d_0(u_0)$ définisse un point dans $\mathcal{Z}_0(I)$ et donc cette représentation peut-être redondante.Par contre, comme les polynômes $d_0(u_0)$ et $d_0'(u_0)$ sont premiers entre eux, les fractions $\zeta_i(u_0) = \frac{d_i(u_0)}{d_0'(u_0)}(i = 1, \ldots, n)$ sont bien définies aux racines de $d_0(u_0)$. Nous verrons plus loin comment éviter les facteurs étrangers.

La preuve du théorème précédent demande le résultat suivant :

Lemme 4.2.2 *Soient $A(\mathbf{u})$ et $B(\mathbf{u})$ deux polynômes de $\mathbb{K}[\mathbf{u}] = \mathbb{K}[u_0, \ldots, u_n]$ premiers entre eux. Alors pour tout $t = (t_0, \ldots, t_n) \in \mathbb{K}^{n+1}$ générique, les éléments $A_0 = A(t_0 + u_0, t_1, \ldots, t_n)$ et $B_0 = B(t_0 + u_0, t_1, \ldots, t_n)$ de $\mathbb{K}[u_0]$ sont premiers entre eux.*

Preuve.
Les racines de A_0 et (resp.) de B_0 correspondent aux points d'intersections de la droite $L_t(u_0) = (t_0 + u_0, t_1, \ldots, t_n)$ avec $\mathcal{Z}(A(\mathbf{u}))$ et (resp.) $\mathcal{Z}(B(\mathbf{u}))$. Puisque $A(\mathbf{u})$ et $B(\mathbf{u})$ sont premiers entre eux, l'intersection est de codimension ≥ 2 dans $\mathbb{K}^{n+1}$. Donc pour tout t générique, L_t ne coupe pas la variété et les polynômes A_0 et B_0 n'ont pas racines communes dans $\overline{\mathbb{K}}$, la fermeture algébrique de $\mathbb{K}$. □

Preuve du théorème 4.2.1
Effectuons la décomposition suivante de $\Delta(\mathbf{u})$:

$$\Delta(\mathbf{u}) = (\prod_{\zeta \in \mathcal{Z}_0} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)^{\mu_\zeta})H(\mathbf{u}) \tag{4.4}$$

avec $\prod_{\zeta \in \mathcal{Z}_0}(u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)^{\mu_\zeta}$ et $H(\mathbf{u})$ premiers entre eux.

On note $d(\mathbf{u}) = \frac{\Delta(\mathbf{u})}{pgcd(\Delta(\mathbf{u}), \frac{\partial \Delta}{\partial u_0}(\mathbf{u}))}$, qui est un polynôme de la forme :

$$d(\mathbf{u}) = \prod_{\zeta \in \mathcal{Z}_0} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)h(\mathbf{u}) \tag{4.5}$$

où $\prod_{\zeta \in \mathcal{Z}_0}(u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)$ et $h(\mathbf{u})$ sont premiers entre eux.

Soit maintenant $t = (t_0, \ldots, t_n)$ générique, avec $t(\zeta) = t_0 + \zeta_1 t_1 + \ldots + \zeta_n t_n$, on a alors :

$$\begin{aligned}
d(t + \mathbf{u}) &= d_0(u_0) + u_1 d_1(u_0) + \ldots + u_n d_n(u_0) + r(\mathbf{u}) \\
&= \prod_{\zeta \in \mathcal{Z}_0(I)} (t(\zeta) + u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)h(t + \mathbf{u}) \\
h(t + \mathbf{u}) &= h_0(u_0) + u_1 h_1(u_0) + \ldots + u_n h_n(u_0) + s(\mathbf{u})
\end{aligned}$$

où $t(\zeta) = t_0 + t_1\zeta_1 + \ldots + t_n\zeta_n$ et $r(\mathbf{u})$, $s(\mathbf{u}) \in (u_1, \ldots, u_n)^2$. Par identification, cela donne :

$$\begin{aligned} d_0(u_0) &= \prod_{\zeta \in \mathcal{Z}_0(I)} (t(\zeta) + u_0) h_0(u_0), \\ d_i(u_0) &= (\sum_{\zeta \in \mathcal{Z}_0(I)} \zeta_i \prod_{\zeta \neq \zeta'} (t(\zeta') + u_0)) h_0(u_0) + \prod_{\zeta \in \mathcal{Z}_0(I)} (t(\zeta) + u_0) h_i(u_0). \end{aligned}$$

On a donc

$$d_0'(u_0) = (\sum_{\zeta \in \mathcal{Z}_0(I)} \prod_{\zeta' \neq \zeta} (t(\zeta') + u_0)) h_0(u_0) + \prod_{\zeta \in \mathcal{Z}_0(I)} (t(\zeta) + u_0) h'(u_0). \tag{4.6}$$

D'après le lemme 4.2.2, pour des valeurs génériques de t, les polynômes,

$$\prod_{\zeta \in \mathcal{Z}_0(I)} (t(\zeta) + u_0)$$

et $h_0(u_0)$ sont premiers entre eux puisque

$$\prod_{\zeta \in \mathcal{Z}_0(I)} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)$$

et $f(\mathbf{u})$ le sont. Ils n'ont pas de racines communes.

Si $-t(\zeta)$ est une racine de $d_0(u_0)$, alors

$$\begin{aligned} d_0' &= (\prod_{\zeta' \neq \zeta} (t(\zeta') - t(\zeta))) h_0(\zeta_0) \\ d_i(\zeta_0) &= \zeta_i (\prod_{\zeta' \neq \zeta} (t(\zeta') - t(\zeta))) h_0(\zeta_0) \end{aligned}$$

t étant générique, pour tout $\zeta' \neq \zeta$, $t(\zeta') \neq t(\zeta)$, et $d_0'(\zeta_0) \neq 0$, donc

$$\zeta_i = \frac{d_i(\zeta_0)}{d_0'(\zeta_0)} \ \textit{pour}\ i = 1, \ldots, n. \tag{4.7}$$

□

Les bonnes racines de $d_0(u_0) = 0$ sont celles pour lesquelles $g_i(u_0) = f_i(\zeta_1, \ldots, \zeta_n) = 0$, c'est-à-dire les racines des facteurs irréductibles de $d_0(u_0)$ qui divisent les numérateurs des $g_i(u_0)$, pour plus de détails on peut lire [14].
Mais dans le cas où $\Delta(\mathbf{u})$ est exactement la forme de Chow, la proposition suivante peut s'appliquer :
Proposition 4.2.3 *Avec les notations du théorème précédent, si $\Delta(\mathbf{u}) = \mathcal{C}_{I_0}(\mathbf{u})$ et t est séparant, alors il y a bijection entre les racines de $d_0(u_0) = 0$ et les éléments de $\mathcal{Z}_0(I)$.*

En pratique, plutôt que de développer entièrement le polynôme $d(t + u)$, on peut considérer les variables $u_1, \ldots, u_n$ comme étant des nombres infinitésimaux (i.e. $u_i^2 = 0 = u_iu_j$, pour $i,j = 1, \ldots, n$), ce qui permet d'obtenir uniquement les premiers termes $d_0(u_0) + u_1d_i(u_0) + \ldots + u_nd_n(u_0)$. Nous allons voir, après la remarque suivante, comment calculer une représentation univariée rationnelle sans avoir à développer le polynôme $d(t + \mathbf{u})$.

Remarque :

La condition de généricité sur t est satisfaite dès que $pgcd(d_0(u_0), d_0'(u_0)) = 1$. Ce qui est vérifiable lorsque $\Delta(\mathbf{u})$ est connu. Dans ce cas t est nécessairement un élément séparant de $\mathcal{Z}_0(I)$, et $h(u_0)$ et $\prod_{\zeta \in \mathcal{Z}_0} (t(\zeta) + u_0)$ n'ont pas de racines communes.

Calcul d'une représentation univariée rationnelle par dérivation.

L'idée est de regarder ce qui se passe lorsque,après avoir donné une direction t, on fait bouger le repère.

Soit I un idéal de dimension 0, $\mathcal{C}(\mathbf{u})$, ($\mathbf{u} = (u_0, \ldots, u_n)$) sa forme de Chow et t un vecteur générique de $\mathbb{K}^{n+1}$, où $\mathbb{K}$ est un corps que l'on suppose algébriquement clos.

Si $\zeta = (\zeta_1, \ldots, \zeta_n), t(\zeta) = t_0 + \zeta_1 t_1 + \ldots + \zeta_n t_n$, et si on pose $d_0(u_0) = \mathcal{C}(t + u_0)$, on peut écrire :

$$d_0(u_0) = \mathcal{C}(t + u_0) := \mathcal{C}(t_0 + u_0, t_1, \ldots, t_n) = \prod_{\zeta \in \mathcal{Z}(I)} (t(\zeta) + u_0)^{\mu_\zeta}. \tag{4.8}$$

puisque

$$\mathcal{C}(\mathbf{u}) = \prod_{\zeta \in \mathcal{Z}(I)} (u_0 + u_1\zeta_1 + \ldots + u_n\zeta_n)^{\mu_\zeta}$$

De même,

$$\mathcal{C}(t + \epsilon + u_0) = \mathcal{C}(t + u_0) + \epsilon \sum_{\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{Z}(I)} \mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi} + \mathcal{O}(\epsilon^2), \tag{4.9}$$

Or,

$$\begin{aligned}
d_0'(u_0) &= \frac{\partial d_0(u_0)}{\partial u_0} \\
&= \frac{\partial}{\partial u_0} \Big(\prod_{\zeta \in \mathcal{Z}(I)} (t(\zeta) + u_0)^{\mu_\zeta} \Big) \\
&= \mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi} + (t(\zeta) + u_0)^{\mu_\zeta} \Big(\prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi} \Big)' \\
&= \sum_{\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{Z}(I)} \mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}
\end{aligned}$$

Ce qui est le coefficient de ϵ dans le développement 4.9.

En effectuant un développement du même style mais avec cette fois le terme ϵx_i on obtient :

$$\mathcal{C}(t + \epsilon x_i + u_0) = \mathcal{C}(t + u_0) + \epsilon \sum_{\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{Z}(I)} \mu_\zeta \zeta_i (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi} + \mathcal{O}(\epsilon^2). \tag{4.10}$$

Si $d_i(u_0)$ est le coefficient de ϵ dans le développement (4.10) de $\mathcal{C}(t + \epsilon x_i + u_0)$, alors :

$$\lim_{u_0 \to -t(\zeta)} \frac{d_i(u_0)}{d_0'(u_0)} = \lim_{u_0 \to -t(\zeta)} \frac{\sum_{\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{Z}(I)} \mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}}{\sum_{\zeta = (\zeta_1, \ldots, \zeta_n) \in \mathcal{Z}(I)} \mu_\zeta \zeta_i (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}}$$

$$= \lim_{u_0 \to -t(\zeta)} \frac{\mu_\zeta \zeta_i (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}}{\sum_{\xi \neq \zeta} \mu_\xi \xi_i (t(\xi) + u_0)^{\mu_\xi - 1} \prod_{\gamma \neq \xi} (t(\gamma) + u_0)^{\mu_\gamma}} \mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi} + \sum_{\xi \neq \zeta} \mu_\xi (t(\xi) + u_0)^{\mu_\xi - 1} \prod_{\gamma \neq \xi} (t(\gamma) + u_0)^{\mu_\gamma}$$

$$\lim_{u_0 \to -t(\zeta)} \frac{d_i(u_0)}{d_0'(u_0)} = \lim_{u_0 \to -t(\zeta)} \frac{\mu_\zeta \zeta_i (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}}{\mu_\zeta (t(\zeta) + u_0)^{\mu_\zeta - 1} \prod_{\xi \neq \zeta} (t(\xi) + u_0)^{\mu_\xi}}$$

$$\lim_{u_0 \to -t(\zeta)} \frac{d_i(u_0)}{d_0'(u_0)} = \zeta_i$$

Ce qui nous donne bien une représentation univariée rationnelle des solutions et conduit à l'algorithme suivant, en ayant considéré que ϵ était un nombre infinitésimal ($\epsilon^2 = 0$) :

Algorithme 4.2.4 *Représentation rationnelle univariée des points isolés d'une variété*

1. *Calculer un multiple $\Delta(\mathbf{u})$ de la forme de Chow $\mathcal{C}(\mathbf{u})$ du système.*
2. *choisir un élément générique $t \in \mathbb{K}^{n+1}$, et calculer $\Delta(t + u_0)$, ce qui donne un multiple de $d_0(u_0)$,*
3. *Calculer les coefficients $d_0'(u_0)$ et $d_i(u_0)$, de ϵ dans $\Delta(t + \epsilon + u_0)$ et $\Delta(t + \epsilon x_i + u_0)$,*
4. *Calculer*
$$\lim_{u_0 \to -t(\zeta)} \frac{d_i(u_0)}{d_0'(u_0)} = \zeta_i,$$
ce qui donne une représentation rationnelle des solutions.

4.3 Matrice de Bézout et Forme de Chow

Nous allons montrer qu'un multiple $\Delta(\mathbf{u})$ de la forme de Chow $\mathcal{C}_{I_0}(\mathbf{u})$ peut-être obtenu à partir d'un mineur maximal non-nul de la matrix de Bézout, et ceci même dans le cas non-générique.

Proposition 4.3.1 *Tout mineur maximal non nul, $\Delta(\mathbf{u})$, de la matrice de Bézout $\mathtt{B}_u$ des polynômes $u_0 + u_1x_1 + \ldots + u_nx_n, f_1, \ldots, f_n \in \mathbb{K}(\mathbf{u})[\mathbf{x}]$ est divisible par la forme de Chow $\mathcal{C}_{I_0}(\mathbf{u})$.*

Preuve.
D'après le lemme 3.2.1 il existe deux bases de $\mathbb{K}(\mathbf{u})[\mathbf{x}]$ telles que la matrice de Bézout $\mathtt{B}_u$ se décompose, dans ces bases, sous la forme :

$$\mathtt{B}_{\mathbf{u}} = \left(\begin{array}{c|c} \mathtt{M}_{\mathbf{u}} & 0 \\ \hline 0 & \mathtt{L}_{\mathbf{u}} \end{array}\right) \tag{4.11}$$

où $\mathtt{M}_u$ est la matrice de multiplication par $u_0 + u_1x_1 + \ldots + u_nx_n$ dans le $\mathbb{K}(\mathbf{u})$-espace vectoriel $\mathbb{K}(\mathbf{u})[\mathbf{x}]/I_0$. Donc tout mineur maximal non nul de la matrice $\mathtt{B}_{\mathbf{u}} = u_0\mathtt{B}_1 + u_1\mathtt{B}_{x_1} + \ldots + u_n\mathtt{B}_{x_n}$ est divisible par $\det(u_0\mathbb{I} + u_1\mathtt{M}_{x_1} + \ldots + u_n\mathtt{M}_{x_n}) = \mathcal{C}_{I_0}(\mathbf{u})$. □

Chapitre 5

Résolution d'un système polynomial bien-contraint par une méthode basée sur le résultant

Soit un système bien-contraint :

$$f_1, \ldots ,f_n \in \mathbb{K}[x_1, \ldots ,x_n], \tag{5.1}$$

Soit l'idéal $I = (f_1, \ldots ,f_n)$, et $\mathcal{Z}^0(I)$ l'ensemble des points isolés de la variété définie par I. Pour utiliser une méthode de résultant pour calculer les racines du système il est nécessaire de définir un système sur-contraint. Pour cela, et pour la résolution du nouveau système défini, plusieurs solutions sont envisageables.

5.1 Résolution par u-résultant et représentation univariée rationnelle

L'idée est de construire un nouveau système en ajoutant un polynôme linéaire $f_0 = u_0 + u_1x_1 + \ldots + u_nx_n$ pour $\mathbf{u} = (u_0, \ldots ,u_n)$ suffisamment générique. En pratique seulement u_0 est une indéterminée, les autres coefficients étant spécialisés par des valeurs aléatoires. On peut alors calculer la matrice de Bézout de ce système dont les coefficients sont des polynômes en $\mathbf{u} = (u_0, \ldots ,u_n)$. D'après la proposition 4.3.1 tout mineur maximal de cette matrice sera un multiple de la forme de Chow $\mathcal{C}_{I_0}(\mathbf{u})$ de $\mathcal{Z}^0(I)$. Ce qui permet d'appliquer l'algorithme 4.2.4 afin de calculer une représentation rationnelle des solutions, la direction donnée et sur laquelle interviendra le déplacement de repère est alors la forme linéaire f_0.

5.2 Résolution par u-résultant et calcul matriciel

Comme pour la méthode précédente, la première étape consiste à construire un nouveau système en ajoutant un polynôme linéaire $f_0 = u_0 + u_1x_1 + \ldots + u_nx_n$ pour $\mathbf{u} = (u_0, \ldots ,u_n)$ suffisamment générique. La matrice de Bézout, $\mathtt{B}$, du système obtenu pourra alors se décomposer sous la forme :

$$\mathtt{B} = u_0\mathtt{B}_0 + u_1\mathtt{B}_1 + \ldots + u_n\mathtt{B}_n$$

De même si $\mathtt{D}$ est la sous-matrice de $\mathtt{B}$ correspondant à un mineur maximal non nul de $\mathtt{B}$, elle pourra se décomposer ainsi :

$$\mathtt{D} = u_0\mathtt{D}_0 + u_1\mathtt{D}_1 + \ldots + u_n\mathtt{D}_n$$

Ce qui conduit à un problème de vecteurs propres généralisé :

Si v est un vecteur propre de $\mathtt{D}$ associé à la valeur propre $\zeta = (\zeta_1, \ldots, \zeta_n)$, alors

$$(\mathtt{D}_i - \zeta_i\mathtt{D}_0)v = 0,\ i = 1, \ldots, n.$$

Donc pour un vecteur propre v tel que $(\mathtt{D}_1 - \zeta_1\mathtt{D}_0)v = 0$, on a $\mathtt{D}_i v = \zeta_i\mathtt{D}_0 v$, $i = 2, \ldots, n$, ce qui permet à partir d'un seul calcul de vecteur propre de retrouver toutes les coordonnées des racines par des multiplications matrice-vecteur puis division de certains coefficients des vecteurs obtenus. Nous pouvons donc appliquer l'algorithme suivant :

Algorithme 5.2.1 *u-résultant et calcul matriciel*
Entrée : $f_1, \ldots, f_n$ polynômes en n variables.

1. *Construction d'un polynôme linéaire $f_0 = u_0 + u_1x_1 + \ldots + u_nx_n$*
2. *Construction de la matrice de Bézout* $\mathtt{B}$
3. *Extraction d'un mineur maximal de* $\mathtt{B}$ *et construction de la matrice associée* $\mathtt{D}$ *(calcul d'un rang approché par SVD)*
4. *Décomposition de* $\mathtt{D}$ *en matrices $\mathtt{D}_0, \ldots, \mathtt{D}_n$, chacune correspondant aux coefficients des u_i dans* $\mathtt{D}$
5. *Résolution du problème de vecteurs propres généralisé :* $(\mathtt{D}_1 - \lambda\mathtt{D}_0)v = 0$.
6. *Calcul de $w_0 = \mathtt{D}_0 v$ et $w_i = \mathtt{D}_i v$, $i = 1, \ldots, n$*
7. *Calcul des ζ_i tels que $w_0\zeta_i = w_i$*

5.3 Variable cachée et perturbation du système

Une autre alternative pour définir un système sur-contraint est de "cacher" une variable x_i dans le corps des coefficients, on considère alors :

$$f_1, \ldots, f_n \in (\mathbb{K}(x_i))[x_1, \ldots, x_n].$$

La variable cachée, x_i, doit être choisie, dans la mesure du possible, telle que toutes les racines soient séparées par projection sur x_i afin d'éviter le cas de racines multiples. Cette approche présente l'avantage de produire un système sur-contraint sans accroître la dimension du problème.

5.3.1 Perturbation du système

Le calcul des coordonnées des racines peut alors se faire par perturbation du système.

Soit $\mathtt{B}_{\mathbf{f}}$ la matrice de Bézout du système défini par $\mathbf{f} = (f_1, \ldots, f_n)$.

Si on pose $\tilde{f}_i(\mathbf{x}) = f_i(x_n + \epsilon x_i)$, un mineur maximal, $\Delta_\epsilon(x_n)$, de la matrice de Bézout, $\mathtt{B}_{\tilde{\mathbf{f}}}$, de ce nouveau système $\tilde{\mathbf{f}} = (\tilde{f}_1, \ldots, \tilde{f}_n)$ sera de la forme :

$$\Delta_\epsilon(x_n) = \Delta(\mathtt{B}_{\mathbf{f}} + \epsilon_1 \mathtt{B}_1(x_n) + \ldots + \epsilon_{n-1}\mathtt{B}_{n-1}(x_n) + \mathcal{O}(\epsilon_i\epsilon_j)).$$

avec

$$\Delta_\epsilon(x_n) = \prod_{\zeta\in\pi_n(\mathcal{V}(I_0))} (x_n - (\zeta_n + \epsilon_1\zeta_1 + \ldots + \epsilon_{n-1}\zeta_{n-1}))^{\mu_\zeta} H_\epsilon(x_n)$$

$$\Delta_\epsilon(x_n) = \prod_{\zeta\in\pi_n(\mathcal{V}(I_0))} (x_n - \zeta_n)^{\mu_\zeta} H_0(x_n) + \sum_{j=1,\ldots,n-1} \epsilon_j(\sum_{\zeta\in\pi_n(\mathcal{V}(I_0))} \zeta_i \prod_{\xi\neq\zeta}(x_n - \xi_n)^{\mu_\xi}) H_1(x_n) + \ldots$$

$$\Delta_\epsilon(x_n) = \Delta(x_n) + \epsilon_1\Delta_1(x_n) + \ldots + \epsilon_{n-1}\Delta_{n-1}(x_n) + \ldots$$

où $\pi_n(\mathcal{V}(I_0))$ est la projection sur x_n de la variété définie par I_0.

D'après le théorème 3.2.2, tout mineur maximal $\Delta(x_n)$, de $\mathtt{B}_{\mathbf{f}}$ est un multiple du résultant du système, donc doit s'annuler pour que ce dernier admette des solutions. On obtient ainsi une condition sur les valeurs des racines en x_n.

La première étape est donc de calculer les valeurs des $n^{ièmes}$ coordonnées des racines.

5.3.2 Valeurs des racines en la variable cachée

Pour trouver la valeur de x_i en les racines plusieurs méthodes sont envisageables :

- On a vu que pour que le système (5.1) ait des solutions le résultant doit être nul, donc tout multiple du résultant doit s'annuler. Alors, si $\mathtt{M}(x_i)$ est la matrice de résultant du système, résoudre $det(\mathtt{M}(x_i)) = 0$ permet de calculer les valeurs de x_i en les racines.

- Une autre facon de faire est, en remarquant que si $f = u_0 + u_1x_1 + \ldots + u_nx_n$, u_0 étant une indéterminée, alors sa matrice de multiplication est de la forme : $\mathtt{M}_f = \mathtt{M}_{\sigma+u_0} = \mathtt{M}_\sigma + u_0\mathbb{I}$, avec $\sigma = u_1x_1 + \ldots + u_nx_n$. Alors, si $f(\zeta) = 0$, $-u_0$ est une valeur propre de $\mathtt{M}_\sigma$. Et on aura bien $det(\mathtt{M}_f) = det(\mathtt{M}_\sigma + u_0\mathbb{I}) = 0$. En considérant $u_0 = x_i$, chercher les valeurs de x_i en les racines, revient donc à un problème de recherche de valeurs propres.

Dans la deuxième facon de procéder, il est possible que la matrice $\mathtt{M}_f$ ne soit pas linéaire en la variable cachée x_i et qu'elle soit une matrice polynomiale $\mathtt{M}_f = \mathtt{A}_d x_i^d + \ldots + \mathtt{A}_1 x_i + \mathtt{A}_0$, pour $d \geq 1$. Si $\mathtt{A}_d$ est numériquement inversible, on ramène l'équation $det(\mathtt{M}_f) = det(\mathtt{A}_d)det(\mathbb{I}x_i^d + \ldots + \mathtt{A}_d^{-1}\mathtt{A}_1 x_i + \mathtt{A}_d^{-1}\mathtt{A}_0) = 0$ au problème de valeurs et vecteurs propres généralisés ([16]) :

$$\left(\begin{bmatrix} 0 & \mathbb{I} & 0 & \cdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \mathbb{I} \\ -\mathtt{A}_0 & -\mathtt{A}_1 & \cdots & -\mathtt{A}_{d-1} \end{bmatrix} - \beta \begin{bmatrix} \mathbb{I} & 0 & \cdots & 0 \\ 0 & \ddots & 0 & \cdots \\ \vdots & 0 & \mathbb{I} & 0 \\ 0 & \cdots & 0 & \mathtt{A}_d \end{bmatrix} \right) \begin{bmatrix} v_\alpha \\ \beta v_\alpha \\ \vdots \\ \beta d - 1 v_\alpha \end{bmatrix} = 0$$

Si $\mathtt{A}_d$ n'est pas inversible, on procède à un changement de variable : $x_i = \frac{(t_1y+t_2)}{(t_3y+t_4)}$.

Dans notre cas, il s'agit de retrouver les valeurs des racines en x_n à partir de la matrice de Bézout, ce qui revient dans le cas linéaire à résoudre le problème de valeurs propres généralisé :

$$(\mathtt{B}_1 - x_n\mathtt{B}_0)v = 0$$

La construction vu pour le cas non-linéaire s'applique aussi pour les matrices de Bézout.

5.3.3 Calcul de toutes les coordonnées des racines

Pour retrouver toutes les coordonnées des racines, les deux types de calcul des méthodes précédentes sont applicables. C'est à dire :

- On dispose de $\Delta(x_n)$ qui, en revenant aux représentations rationnelles univariées des solutions, est un multiple de $d_0(x_n)$, des $\Delta_i(x_n)$, multiples des $d_i(x_n)$, on peut donc calculer :

$$\zeta_i = \lim_{x_n \to \zeta_n} \frac{\Delta_i(x_n)}{\Delta'(x_n)}$$

.

- Ou, en remarquant que pour un vecteur propre v_ϵ de $\mathtt{B}_\epsilon(x_n)$ alors $(\mathtt{B}_\mathtt{f} + \epsilon_1\mathtt{B}_1(x_n) + \ldots + \epsilon_{n-1}\mathtt{B}_{n-1}(x_n))v_\epsilon = \lambda\mathtt{B}_0 v_\epsilon$, il suffit de calculer les vecteurs propres du problème généralisé : $(\mathtt{B}_{epsilon} - \lambda\mathtt{B}_0)v = 0$ et d'effectuer les calculs $w_i = \mathtt{B}_i v_\epsilon$ et $w_0 = \mathtt{B}_0 v_\epsilon$ pour $i = 1, \ldots, n-1$ ce qui permettra de retrouver les ζ_i tels que $w_0\zeta_i = \epsilon_i w_i$.

5.3.4 Un algorithme

Nous présentons ici un algorithme induit par l'approche décrite précédemment calculant les racines par recherche de valeurs propres.

Algorithme 5.3.1 *Variable cachée et perturbation du système - Résolution par recherche de valeurs propres Entrée : $f_1, \ldots, f_n$ à coefficients polynomiaux en x_n*

1. *Construction de la matrice de Bézout* $\mathtt{B}$ *du système*
2. *Extraction d'un mineur maximal et d'une sous-matrice* $\mathtt{SB}$
3. *Calculer les valeurs des racines ζ_n en x_n*
4. *Pertuber le système en remplacant x_n par $x_n + \epsilon_1 x_1 + \ldots + \epsilon_{n-1}x_{n-1}$, ϵ_i infinitésimal.*
5. *Extraction d'une nouvelle sous-matrice $\mathtt{SB}_\epsilon$ et décomposition : $\mathtt{SB}_\epsilon = \mathtt{SB}_0 + \epsilon_1\mathtt{SB}_1 + \ldots + \epsilon_{n-1}\mathtt{SB}_{n-1}$.*
6. *Résoudre le problème de valeurs propres généralisé :$(\mathtt{SB}_i - \zeta_i\mathtt{SB}_0)v_\epsilon = 0$.*
7. *Calcul de $w_0 = \mathtt{SB}_0 v_\epsilon$ et $w_i = \mathtt{SB}_i v_\epsilon$*
8. *Calcul des ζ_i tels que $w_0\zeta_i = \epsilon_i w_i$.*

Chapitre 6

Types de problème à résoudre

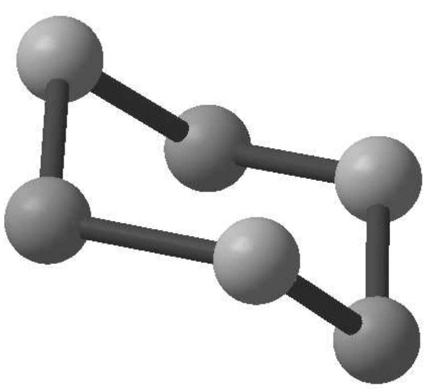
De nombreux problèmes en robotique, imagerie, traitement du signal,..., demande une résolution de systèmes d'équations polynomiales. Nous présentons ici deux problèmes industriels et la formulation algébrique qui en découle.

6.1 Configurations de molécules

Une branche récente de la biologie et de la chimie algorithmique cherche à utiliser des algorithmes venant de la géométrie ou de la robotique en les appliquant à la recherche de configuration de molécules. Ici nous présentons le cas d'une molécule à 6 atomes, pour plus de détails on peut lire [14].

6.1.1 Problème

On considère des molécules cycliques à 6 atomes :



Ceci est une configuration possible d'une telle molécule. Le parallèle avec la robotique est évident si on considère les atomes comme des articulations et les liens entre eux comme des joints rigides. Les longueurs des liens et les angles entre deux liaisons consécutives sont considérés fixés. Seuls sont autorisés à varier les angles dihédraux, c'est-à-dire les angles solides formés par deux plans consécutifs, chacun étant défini par un atome et ses deux liens. Le seul mouvement permis est la rotation autour des axes des liaisons. En termes géométriques, la molécule est équivalente à un mécanisme série dans lequel tout couple d'axes consécutifs s'intersectent à un atome. Ceci nous permet de réduire le problème 6-dimensionnel à un système de 3 équations à trois inconnues.

6.1.2 Formulation algébrique

La molécule a une structure cyclique de 6 atomes, typiquement de carbone, ce qui détermine la structure primaire. Ce qui est en dehors de cette structure est ignorée. Les longueurs des liens et les angles qu'ils forment donnent les contraintes, les angles dihédraux pouvant varier.

Deux formulations peuvent être envisagées, l'une utilisant les angles, l'autre les distances. Seule la première est présentée ici.

La première donne le système polynomial suivant :

$$\alpha_{11} + \alpha_{12}cos\theta_2 + \alpha_{13}cos\theta_3 + \alpha_{14}cos\theta_2 cos\theta_3 + \alpha_{15}sin\theta_2 sin\theta_3 = 0 \tag{6.1}$$
$$\alpha_{21} + \alpha_{22}cos\theta_3 + \alpha_{23}cos\theta_1 + \alpha_{24}cos\theta_3 cos\theta_1 + \alpha_{25}sin\theta_3 sin\theta_1 = 0 \tag{6.2}$$
$$\alpha_{31} + \alpha_{32}cos\theta_1 + \alpha_{13}cos\theta_2 + \alpha_{14}cos\theta_1 cos\theta_2 + \alpha_{35}sin\theta_1 sin\theta_2 = 0 \tag{6.3}$$
$$cos^2\theta_1 + sin^2\theta_1 = 1 \tag{6.4}$$
$$cos^2\theta_2 + sin^2\theta_2 = 1 \tag{6.5}$$
$$cos^2\theta_3 + sin^2\theta_3 = 1 \tag{6.6}$$

où les α_{ij} sont les coefficients d'entrée.

Il peut être intéressant de travailler sur une formulation équivalente mais comportant moins d'équations. Ceci peut se faire en appliquant la transformation standard aux demi-angles, qui donne des équations rationnelles en de nouvelles inconnues t_i :

$$t_i = tan\frac{\theta_i}{2} \ : \ cos\theta_i = \frac{1-t_i^2}{1+t_i^2},\ sin\theta_i = \frac{2t_i}{1+t_i^2},\ i = 1{,}2{,}3.$$

Ceci prend automatiquement en compte les trois dernières équations, et en multipliant les deux côtés de la i$^{\text{ème}}$ équation par $(t + t_j^2)(1 + t_k^2)$, où $(i{,}j{,}k)$ est une permutation de 1,2,3, on obtient le système suivant :

$$f_1 = \beta_{11} + \beta_{12}t_2^2 + \beta_{13}t_3^2 + \beta_{14}t_2t_3 + \beta_{15}t_2^2t_3^2 = 0 \tag{6.7}$$
$$f_2 = \beta_{21} + \beta_{22}t_3^2 + \beta_{23}t_1^2 + \beta_{24}t_3t_1 + \beta_{25}t_3^2t_1^2 = 0 \tag{6.8}$$
$$f_3 = \beta_{31} + \beta_{32}t_1^2 + \beta_{33}t_2^2 + \beta_{34}t_1t_2 + \beta_{35}t_1^2t_2^2 = 0 \tag{6.9}$$

où les β_{ij} sont des coefficients d'entrée.

6.1.3 Résolution - Résultats

Les résultats présentés ici ont été calculé à partir de la méthode (paragraphe 5.2 de la première partie) basée sur l'ajout d'une forme linéaire au système et de recherche de valeurs propres. Le système à résoudre a le nombre maximal de racines, 16, et, de plus elles sont toutes réelles.

$$\begin{array}{c} -x_2^2x_3^2 - x_3^2 + 24x_2x_3 - x_2^2 - 13 \\ -x_1^2x_3^2 - x_3^2 + 24x_1x_3 - x_1^2 - 13 \\ -x_1^2x_2^2 - x_2^2 + 24x_1x_2 - x_1^2 - 13 \end{array}$$

Le rang de la Bézoutienne du système, après ajout de la forme linéaire $f_0 = u_0 + u_1 x_1 + \ldots + u_n x_n$ est de 28.

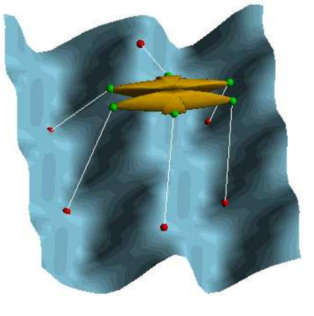
x_1	x_2	x_3
10.850487224	0.779548045	0.779548045
0.779548045	0.779548045	10.850487224
0.779548045	10.853913563	0.779548045
0.779548045	0.779524472	0.779548045
0.332599000	4.625181601	4.625181601
-0.779548045	-0.779548344	-0.779548045
-0.779548045	-0.779548045	-10.859502097
-0.779548045	-10.859502097	-0.779548045
-10.853211545	-0.779548045	-0.779548045
-4.625181602	-4.625181601	-4.625181601
-4.625181601	-4.625181601	-0.331957622
-4.625181601	-0.331957622	-4.625181601
-0.332989621	-4.625181601	-4.625181601
4.625181602	4.625181601	4.625181601
4.625181601	0.332599000	4.625181601
4.625181601	4.625181602	0.332599000

6.2 Robot parallèle

Voici une brève présentation du problème direct d'un manipulateur parallèle (appelé aussi plate-forme de Stewart ou main-gauche) et des équations algébriques associées.

6.2.1 Problème

On considère, sur un solide fixe S_X, 6 points fixes $(X_i)_{1\leq i\leq 6}$, et 6 points Z_i d'un solide mobile S_Z. Les articulations entre les deux solides S_X et S_Z sont des barres extensibles $(X_i, Z_i)_{1\leq i\leq 6}$ attachées aux solides par des joints sphériques.



La plateforme subit des petits déplacements en modifiant la longueur des barres. Dans la pratique, il est facile de mesurer les longueurs des articulations, mais beaucoup moins de déterminer la position du solide S_Z en fonction de ces longueurs. Des études portant sur les contraintes algébriques et plus particulièrement sur l'application qui associe à un déplacement du solide S_Z, le carré de la longueur $\|X_i, Z_i\|$, plus précisément, ont permis de mieux comprendre la géométrie du problème. Il a ainsi été montré qu'il n'y a pas plus de 40 positions possibles lorsqu'on connait les distances $\|X_i,Z_i\|$. Des travaux (c.f. [9]) ont montrés que ces 40 positions peuvent être réelles. Pour une présentation plus profonde de ces études, on peut lire [17], [21], [20].

6.2.2 Formulation algébrique

Cette formulation est décrite en détail dans [20].
Le problème est donc d'obtenir le nombre de positions possibles du robot lorsque l'on connait les longueurs des bras.
On note $\mathbb{K}$ ($\mathbb{C}$ en théorie et plutôt $\mathbb{R}$ en pratique) un corps, et $\mathbb{A}^n = \mathbb{K}^n$ l'espace affine de dimension n sur $\mathbb{K}$. On considère $\mathbb{A}^n$ comme un ensemble ouvert de l'espace projectif[1] $\mathbb{P}(\mathbb{K}^{n+1})$. On note $\mathcal{D}$ l'ensemble des déplacements de $\mathbb{A}^3$. C'est un ensemble algébrique de dimension 6 et on a $\mathcal{D} = SO_3 \times \mathbb{K}^3$, $\mathbb{K}^3$ où $\mathbb{K}^3$ est l'ensemble des translations et SO_3 est la variété[2] des rotations, laissant fixe l'origine.

Si (Y_i) est la position initiale du solide S_Y, on s'intéresse à l'application :

$$\psi \ : \ \mathcal{D} \to \ \mathbb{K}^6$$

$$D \mapsto (\|D.Y_i - X_i\|^2)_{1 \leq i \leq 6}$$

L'espace D est aussi appelé *espace des configurations* et $\mathbb{K}^6$ l'*espace des observations.*

La variété SO_3 peut-être paramétrisée (à l'aide des quaternions) par :

$$\omega \ : \ \mathbb{P}^3 - V(a^2+b^2+c^2+d^2=0) \to SO_3$$

$$(a,b,c,d) \mapsto R = \frac{1}{a^2+b^2+c^2+d^2} \begin{vmatrix} a^2+b^2-c^2-d^2 & 2(bc-da) & 2(bd-ca) \\ 2(bc+da) & a^2-b^2+c^2-d^2 & 2(cd-ba) \\ 2(bd+ca) & 2(cd+ba) & a^2-b^2-c^2+d^2 \end{vmatrix}$$

où $V(a^2+b^2+c^2+d^2=0)$ est la variété définie par l'équation $a^2+b^2+c^2+d^2$. De plus, ω est bijective.
On note $T \in \mathbb{K}^3$, le vecteur de translation, $T = [T_1,T_2,T_3]$.

Nous allons considérer la fibre[3] "générique" de l'application ψ. Elle possède le même nombre de points (comptés avec leurs multiplicités) que la fibre de $\psi(\omega \times Id)$. Ce nombre est appelé le *degré* de l'application $\psi(\omega \times Id)$ et correspond aussi à la dimension suivante ([20]) :

Définition 6.2.1 *Le degré d de l'application $\psi(\omega \times Id)$.est la dimension du corps $\mathbb{K}(\|\omega(\frac{a}{d},\frac{b}{d},\frac{c}{d}.Y_i + T - X_i\|^2_{a \leq i \leq 6})$ comme K-espace vectoriel, où $K = \mathbb{K}(\frac{a}{d},\frac{b}{d},\frac{c}{d},T_1,T_2,T3)$.*
Pour calculer explicitement ce degré, la proposition suivante est utilisée :

Proposition 1 *La fibre d'une valeur non critique est un ensemble algébrique de d points distincts sur lequel le jacobien de l'application considérée est de rang maximal.*

Les valeurs non-critiques sont dans le complémentaire de l'image du seuil critique, qui est un ouvert dense, sous-ensemble de l'espace des observations. Donc en prenant une valeur au hasard dans ce dernier espace, il est assuré qu'elle ne sera pas une valeur critique. La multiplicité des

1. On appelle *espace projectif* de dimension n sur le corps $\mathbb{K}$, noté $\mathbb{P}^n(k)$, l'ensemble des classes d'équivalence de $\equiv$ sur $\mathbb{K}^{n+1} - \{0\}$, avec $(x'_0,\ldots,x'_n) \equiv (x_0,\ldots,x_n)$ si il existe $\alpha \in \mathbb{K}$ tel que $(x'_0,\ldots,x'_n) \equiv \alpha(x_0,\ldots,x_n)$.
2. Si $\mathbb{K}$ est un corps et si $f = (f_0,\ldots,f_s)$ sont des polynômes, on définit $V(f_1,\ldots,f_s) = \{(a_1,\ldots,a_n) \in \mathbb{A}^n(k) \ : f_i(a_0,\ldots,a_n) = 0$ pour tout $1 \leq i \leq s$ comme la *variété algébrique* définie par $f = (f_0,\ldots,f_s)$.
3. Si $\phi \ : \ Y \mapsto X$ est une application régulière, X et Y étant des ensembles algébriques, une *fibre* de ϕ est l'image réciproque d'un point de X, i.e. $\phi^{-1}(p)$, $p \in X$.

points distincts des fibres aux valeurs non critique est donc de 1. Comme le nombre de points, compt'es avec multiplicité, est le degré d de l'application, il y a exactement d points distincts dans un fibre générique de ψ. Ceci permet de calculer le degré de ψ comme le degré de l'idéal d'efinissant une fibre de ψ.

Sont présentées ici les équations établies directement dans un espace de dimension 6, en utilisant une paramétrisation des rotations par des quaternions.
En se donnant 6 carrés des longueurs δ_i et les points (X_i), (Y_i), il faut déterminer combien de solutions $D = (R,T)$ satisfont le système :

$$\begin{aligned} & [R.Y_i + T - X_i, R.Y_i + T - X_i] = \delta_i \\ = \quad & [Y_i,Y_i] + [T,T] + [X_i,X_i] \\ & +2\,[R.Y_i,T] - 2\,[X_i,T] - 2\,[R.Y_i,X_i] \end{aligned}$$

pour $1 \leq i \leq 6$. Comme les points (Y_i) correspondent aux positions initiales du solide S_Z, nous pouvons supposer sans perte de généralité que $Y_6 = X_6 = 0$ De plus, on note $X_6 = X_0 = [0,0,0]$, $Y_6 = Y_0 = [0,0,0]$, $Z_0 = Z_6$ ... On a donc $[T,T] = \delta_6 = \alpha_0$, ce qui permet d'obtenir le système équivalent :

$$\left\{ \begin{array}{l} [T,T] = \alpha_0 \\ [R.Y_1,T] - [X_1,T] - [R.Y_1,X_1] = \alpha_1 \\ [R.Y_2,T] - [X_2,T] - [R.Y_2,X_2] = \alpha_2 \\ [R.Y_3,T] - [X_3,T] - [R.Y_3,X_3] = \alpha_3 \\ [R.Y_4,T] - [X_4,T] - [R.Y_4,X_4] = \alpha_4 \\ [R.Y_5,T] - [X_5,T] - [R.Y_5,X_5] = \alpha_5 \end{array} \right. \tag{6.10}$$

Les valeurs α_i sont des combinaisons linéaires des carrés des longueurs initiales $(\delta_i)_{1 \leq i \leq 6}$, par exemple $\alpha_1 = \delta_1 - \delta_6$.

Deuxième partie

Environnement pour le calcul symbolique et numérique

Chapitre 1

Introduction

Certaines méthodes comme les méthodes de minimisation, les itérations de Newton sont souvent utilisées, mais le résultat n'est pas toujours garanti. Une autre méthode connue, la méthode des bases de **Groebner**, appliquée à l'arithmétique flottante ne donne pas de résultats corrects et a un coût de calcul élevé.

Les méthodes présentées dans la première partie ont de multiples avantages. Tout d'abord elles sont, si les problèmes sont raisonnables, rapides et stables numériquement, puisqu'on a une présentation matricielle du problème. Elles ont ainsi l'avantage d'être applicables en arithmétique flottante. De plus, les méthodes basées sur le Bézoutien permettent de calculer les points isolés même si le problème a des composantes de dimension non nulle.

Ce dernier point peut-être illustré en reprenant le système présenté pour le problème de la molécule à 6 atomes que l'on définit par la matrice des β_{ij} :

$$\begin{pmatrix} -\frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{1}{2} & 2 & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{1}{2} & 2 & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{1}{2} & 2 & \frac{\sqrt{3}}{2} \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{1}{2} & 2 & \frac{\sqrt{3}}{2} \end{pmatrix}$$

Ce n'est donc pas un système zéro-dimensionnel. En effet, c'est l'union d'une courbe de degré 4 dont la projection sur le plan des coordonnées sont définies respectivement par l'équation $p_1 = 0$, $p_2 = 0$, $p_3 = 0$, et de 16 points. Parmi ces 16 points, 4 points sont sur la droite $t_1 = t_2 = t_3$ et sont isolés, tandis que les 12 autres points sont immergés dans la courbe. Une méthode basée sur le Bézoutien permet de retrouver les points qui sont immergés dans la courbe.

Chapitre 2

Exemples de méthodes combinant calcul symbolique et calcul numérique

Le but de ce chapitre est de présenter deux méthodes commbinant le calcul symbolique et numérique. Tout d'abord est effectué un rappel sur la méthode classique des puissances et puissances inverses.

2.1 Les méthodes des puissances classique et inverse

Nous avons vu dans la première partie comment un problème de résolution d'équations polynomiales pouvait se réduire à un problème de recherche de vecteurs et valeurs propres. Nous rappelons ici la méthode classique des puissances. Celle ci peut s'adapter au cas univariable comme au cas multivariable. L'anneau R considéré ici est donc indifféremment un anneau de polynômes, $R=\mathbb{K}[x]$, en une variable sur un corps $\mathbb{K}$ ou un anneau de polynômes, $R=\mathbb{K}[x_1,\ldots,x_n]$ en n variables. L'idéal $I \subset R$ est un idéal zéro-dimensionnel, engendré par un polynôme f dans le cas univariable, ou par un ensemble de m polynômes $f_1,\ldots,f_m$ dans le cas multivariable.
Soit $\mathtt{M}_a$ la matrice de multiplication pas a dans une base $\mathcal{B}$ de l'algèbre quotient $\mathcal{A}=R/I$, de dimension d. $\mathtt{M}_a$ est supposée telle que $a(\zeta_i)\neq 0$, $i=1,\ldots,d$. Par définition $\mathtt{M}_a$ est inversible et a est lui-aussi inversible dans $\mathcal{A}$.
Soit $\mathbf{v}_0=\mathbf{w}_0$ les vecteurs des coordonnées d'un élément de $\widehat{\mathcal{A}}$ dans la base duale de $\mathcal{B}$. Les méthodes classiques des puissances et (resp.) puissances inverses consistent à effectuer les itérations suivantes :

$$\mathbf{w}_k=\frac{1}{\|\mathbf{w}_{k-1}\|}\mathtt{M}_a^t\mathbf{w}_{k-1} \text{ et } (resp.)\ \mathbf{v}_k=\frac{1}{\|\mathbf{v}_{k-1}\|}(\mathtt{M}_a^t)^{-1}\mathbf{v}_{k-1},\ k=1,2,\ldots$$

Si ζ est une racine simple maximisant ou (resp.) minimisant $\|a\|$ et si $\mathbf{w}=\mathbf{v}=(\zeta^\alpha)$ est la base monomiale évaluée en ζ, nous avons :

$$\lim_{k\to\infty}\mathbf{w}_k=\mathbf{w}\ et\ (resp.)\ \lim_{k\to\infty}\mathbf{v}_k=\mathbf{v}$$

.

Ce qui nous donne un moyen de sélectionner un vecteur propre de $\mathtt{M}_a^t$ correspondant à la racine maximisant ou (resp.) minimisant le module de a sur $\mathcal{Z}(I)$.

Remarques 2.1.1 – *Si il y a k racines minimisant le module de a sur $\mathcal{Z}(I)$, il est encore possible de retrouver les vecteurs propres correspondant à ces racines à partir des vecteurs successifs $\mathbf{v}_n,\ldots,\mathbf{v}_{n+k+1}$.*

- *Dans le cas où ζ est une racine multiple, le processus converge encore vers un vecteur propre de $\mathtt{M}_a^t$ mais plus lentement.*

Les deux prochains paragraphes sont consacrés à la présentation de deux méthodes dérivées de celles que nous venons de voir, l'une pour le cas univariable (méthode de **Sebastio e Silva**), l'autre pour le cas multivariable (méthode inverse implicite).

2.2 Méthode de Sebastio et Silva

Soit $p \in \mathbb{K}[x]$, un polynôme de degré d, en une variable, à coefficients dans un corps $\mathbb{K}$, et $\mathcal{A} = \mathbb{K}[x]/(p)$. Une base traditionnelle de l'algèbre quotient $\mathcal{A}$ est la base monomiale :

$$M = \{1,x^1,\ldots,x^{d-1}\}$$

Un autre base intéressante est la base de Horner, invoquant les polynômes qui apparaissent dans le schéma de Horner associé à p :

$$H = \{h_{d-i} \;=\; a_0x^{d-i} + a_1x^{d-i-1} + ... + a_{d-i},\, 1 \leq i \leq d-1\}$$

Cette dernière base présente deux avantages principaux par rapport à la base monomiale. Tout d'abord, calculer un carré f^2 dans la base de Horner de $\mathcal{A}$ nécessite seulement deux produits de convolution, et pour le produit fg, quatre convolutions sont nécessaires (voir [5]). Un autre avantage de la base de Horner est que, si α est une racine simple alors, les idempotents $\mathbf{e}_\alpha = \frac{1}{p'(\alpha)}\frac{p(x)}{x-\alpha}$ se décomposent dans H comme $\mathbf{e}_\alpha \;=\; \frac{1}{p'(\alpha)}(1,\alpha,\alpha^2,\ldots,\alpha^{d-1})$. Ce qui entraine que si une méthode converge vers un $\mathbf{e}_\alpha$ alors nous n'obtenons pas seulementune racine approchée α mais aussi toutes ses puissances jusqu'à α^{d-1}. Ce qui peut être utilisé pour évaluer toute expression algébrique telle que $g(\alpha)$, même un polynôme de degré supérieur à $d-1$, puisque dans $\mathcal{A}$ tout polynôme est réduit modulo p à un polynôme de degré au plus $d-1$.
La méthode de **Sebastion e Silva** (voir [5]) est un processus itératif pour approximer numériquement une racine d'un polynôme p. C'est une méthode très proche de celle décrite dans le paragraphe précédent mais elle utilise un processus accéléré. En effet, plutôt que de calculer les puissances successives f^n, on calcule les carrés successifs $(f^2)^n$:

Algorithme 2.2.1 *Méthode de Sebastio e Silva*

- *Choisir une valeur initiale f_0 dans $\mathcal{A}$*
- *Itérer $f_{n+1} \;=\; \frac{(f_n)^2}{\|(f_n)^2)\|}$ dans $\mathcal{A}$.*

Alors, si il existe une racine simple α maximisant $\|f\|$, nous obtenons :

$$\lim_{n\to\infty} f_{n+1} = \mathbf{e}_\alpha$$

.

De plus, si le processus est effectué dans la base de Horner, il est facile de sélectionner la racine la plus proche d'un point donné. En effet, en supposant que pour un point donné $t \in \mathbb{K}$, il existe une seule racine α proche de t, le processus est démarré avec la valeur initiale $J_t \;=\; (1,t,t^2,\ldots,t^{d-1}) \;=\; \frac{p(x)\;-\;p(t)}{(x-t)p'(t)}$, donc $J_t(\alpha) \;=\; \frac{-p(t)}{\alpha-t}$. Alors, l'évaluation de J_t en les racines de p est maximale pour $\|\alpha\|$ maximal. Par exemple, pour calculer la plus petite racine de p, le processus est lancé avec J_0, pour la plus grande p est inversé et la plus petite racine est recherchée.
Il est possible d'obtenir d'autres racines de p après en avoir calculé une. Pour cela il est nécessaire de procéder à une déflation du polynôme p. Cette déflation peut être explicite (calcul exact de $\frac{p(x)}{x-\alpha}$, ou implicite. Dans ce dernier cas, après avoir obtenu un premier idempotent $\mathbf{e}_\alpha$ à partir

d'une valeur initiale f, nous choisissons un élément $g \in \mathcal{A}$, pouvant être encore f, et relancons le processus avec $(g - g\mathbf{e}_\alpha)$ comme valeur initiale. Comme $(g - g\mathbf{e}_\alpha) = 0$, α n'est plus dominante et le processus ne peut plus converger vers $\mathbf{e}_\alpha$. A la sortie, nous obtenons un nouvel idempotent, $\mathbf{e}_\beta$, où β est la racine dominante associée à $\mathbf{e}_\beta$, et α est ignorée. Au prochain pas, un nouvel élément h est choisi (pouvant encore être f ou g) et le processus est redémarré avec $h - h(\mathbf{e}_\alpha + \mathbf{e}_\beta)$. En continuant de cette facon, nous pouvons récupérer autant de racines de p que nous le souhaitons.

2.3 Méthode inverse implicite

Il s'agit ici d'appliquer la méthode des puissances inverses au cas multivariable. Donc $\mathcal{A} = \mathbb{K}[x_1, \ldots, x_n]/I$ où $I = (f_1, \ldots, f_m)$. Cette méthode nécessite le calcul de la matrice de multiplication par $f_0 \in \mathcal{A}$, $\mathtt{M}_{f_0}$ qui peut être retrouvée à partir d'une matrice de résultant, comme par exemple, la matrice de Sylvester (voir [8]).
Tout d'abord, rappelons que cette méthode permet de calculer une racine minimisant le module de f_0. Elle présente l'avantage, par rapport, à la méthode classique, de permettre l'utilisation de décalages des variables, qui est la transformation de la matrice $\mathtt{M}_{f_0}$ en la matrice $(\mathtt{M}_{f_0} - \sigma\mathbb{I})^{-1}$, où σ est proche de la valeur propre sélectionnée, et ceci pour accélérer la convergence (voir [16]).

Soit une matrice $\mathtt{S}$, inversible, de la forme :

$$\mathtt{S} = \begin{pmatrix} \mathtt{A} & \mathtt{B} \\ \mathtt{C} & \mathtt{D} \end{pmatrix} \tag{2.1}$$

et qui, sous certaines hypothèses, donne la matrice $\mathtt{M}_{f_0}$ comme le complément de schur de $\mathtt{D}$:

$$\mathtt{M}_{f_0} = \mathtt{A} - \mathtt{B}\,\mathtt{D}^{-1}\,\mathtt{C}.$$

Et,

$$\mathtt{S}^{-1} = \begin{pmatrix} \mathtt{U} & \mathtt{V} \\ \mathtt{Z} & \mathtt{W} \end{pmatrix}$$

On a alors $\mathtt{M}_{f_0}$ qui est inversible et $\mathtt{U} = \mathtt{M}_{f_0}^{-1}$. Voici maintenant l'algorithme itératif :

Algorithme 2.3.1 *Méthode des puissances inverses implicites*

1. *Choisir un vecteur aléatoire* $\mathbf{u}_0$.
2. *Résoudre le système* $\mathtt{S}^t \begin{bmatrix} \mathbf{v}_{n+1} \\ \mathbf{v}'_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_n \\ 0 \end{bmatrix}$.
3. *Si* $v_{n+1,1}$ *est la première coordonnées de* v_{n+1} *et n'est pas nulle, alors,* $u_{n+1} = \frac{1}{v_{n+1,1}} v_{n+1}$, *sinon arrêt.*

A la sortie, si $\zeta \in \mathcal{Z}(I)$ est une racine simple minimisant $\|f_0\|$ et $\sigma = (\zeta^\alpha)_{\alpha \in E_0}$,

$$\lim_{n \to \infty} u_n = \sigma$$

.

Cet algorithme peut être accéléré en calculant une décomposition LU de la matrice $\mathtt{S}^t$ et en tenant compte de son caractère creux.

Chapitre 3

Implémentation de l'environnement

La librairie **ALP** est implémentée en **C++**. Elle est dédiée à la manipulation d'objets en algèbre linéaire et polynomiale. Dans ce chapitre, nous commencons par expliquer les raisons du choix d'un langage tel que le **C++** illustré par quelques exemples. Puis nous donnons un apercu des structures de données.

3.1 Choix du C++

Le langage **C++** commence à être utilisé pour l'écriture de librairies mathématiques, non pas pour son caractère orienté objet mais pour la généricité qu'il offre. Nous expliquons ici pourquoi, en fonction de nos objectifs, ce langage est le plus approprié actuellement. Cet exposé est illustré par quelques exemples significatifs.

3.1.1 Objectifs

La librairie **ALP** a pour objectif d'être le plus générique et efficace possible. Pour cela elle doit pouvoir utiliser des structures de données et des implémentations paramétrées, qui peuvent être spécialisées et optimisées, et ceci sans modification de la librairie. Nous souhaitons pouvoir, par exemple, changer facilement le type des coefficients des polynômes ou simplement effectuer une spécialisation d'un algorithme pour une structure de données particulière.
De plus cette librairie doit pouvoir avoir des connexions avec des librairies externes telles que **Lapack** pour l'algèbre linéaire numérique, **gmp** pour l'arithmétique en grande précision, ou **umfpack** pour l'accès à des solveurs de systèmes linéaires creux.

3.1.2 Intérêts du C++

La facilité de surcharge des opérateurs prédéfinis permet de traduire un algorithme en langage **C++** en utilisant des notations assez intuitives, comme nous le montrons dans l'exemple d'inversion d'un polynôme dans la section suivante. Ce qui a le grand avantage de rendre le code clair et de repousser la frontière entre les types et fonctions définis par l'utilisateur et ceux prédéfinis du langage.
De plus le **C++** en tant que langage orienté objet permet le *polymorphisme*, mais il a été choisi, pour des raisons d'efficacité de ne pas utiliser de classes abstraites ni de fonctions virtuelles qui définissent un polymorphisme dynamique. En effet, l'appel d'une fonction virtuelle est une décision prise pendant l'exécution, et même si un bon compilateur minimisera autant que possible le coût d'une telle fonction, celui-ci devient non-négligeable lors d'appels intensifs. De plus une fonction virtuelle ne peut pas être déclarée en ligne, puisque le type de l'objet n'est pas connu à la compilation. C'est pour cette raison que la librairie **ALP** utilisent de nouvelles techniques,

inspirées de la **STL**, bibliothèque appartenant maintenant à la distribution standard **C++**, à base de *template*, ce qui permet un polymorphisme "statique". Elles sont utilisées pour guider la compilation afin d'obtenir un code efficace. Ces nouvelles techniques, appelées *template expressions*, permettent de construire des programmes **C++** meilleurs en performance que leurs analogue **FORTRAN** (c.f. [23]). Pour une analyse poussée de ces techniques de programmation en calcul scientifique on peut lire [11].

De plus le **C++** offre une grande facilité de connexion avec d'autres librairies, ce qui permet de les intégrer parfaitement dans les structures de données et les algorithmes. Un exemple d'utilisation de la librairie **Lapack** est donnée dans 4.6, un exemple avec la librairie **gmp** dans 4.2

3.2 Structures de données

3.2.1 Approche

La particularité de notre approche est que le choix de la bonne implémentation est effectué à la compilation, ceci étant du à l'utilisation de *templates*. Un point intéressant des nouvelles techniques exposées ci-dessus, les *template expressions*, se situe dans la compilation de certaines opérations telles que multiplication (ou addition,...) d'une matrice par une matrice, ou d'une matrice par un vecteur. En effet, plutôt que de calculer un objet contenant le résultat de l'évaluation ces opérations retournent un moyen de le calculer. Le calcul proprement dit se faisant uniquement lorsqu'il est nécessaire, comme lors d'un assignement ou d'une impression.
Voici un exemple de l'utilisation des expressions template pour l'addition des vecteurs :

```
v = v1 + v2 + v3;
```

où `v, v1, v3` sont des vecteurs (de type `Vector`). Les implémentations habituelles définiraient l'opérateur

```
Vector & operator+(const Vector & v1, Vector & v2);
```

de telle facon que l'instruction précédente produirait un vecteur temporaire, `vtmp1`, pour l'addition de `v1` et `v2`, un autre, `vtmp2` pour l'addition de `vtmp1` et `v3` puis l'operateur

```
Vector & operator=(const Vector & v);
```

serait utilisé pour assigner à `v` la valeur de `vtmp2`, en utilisant une boucle

```
for(index_type i=0; i<v.size(); i++) v[i]=vtmp2[i];
```

Dans ce type d'implémentation, deux vecteurs temporaires sont alloués et détruits, ce qui est une consommation de temps et d'espace mémoire, spécialement dans le cas de gros vecteurs.

Dans **ALP**, en utilisant les expressions template, nous avons l'opérateur

```
VectHdl<Plus<Vector,Vector> > & operator+(const Vector & v1, Vector & v2);
```

où VectHdl<Plus<Vector,Vector» est le type d'une structure de données contenant des références sur les deux vecteurs `v1`, `v2` mais qui ne calcule pas la somme de ces vecteurs. L'instruction `v1+v2+v3` considérée produit alors une donnée du type

```
VectHdl<Plus<Vector,VectHdl<Plus<Vector,Vector> > > >;
```

Ainsi, l'assignement

```
v = v1 + v2 + v3;
```

invoquera l'opérateur

```
Vector & operator=(const Vector & v);
```

qui est implémenté avec

```
for(index_type i=0; i<v.size(); i++) (*this)[i]=_elem(v,i);
```

Le point important est que le compilateur expandera l'appel à la fonction en ligne `_elem(v,i)` dans `v1[i]+v2[i]+v3[i]`, puisque le ième coefficient d'un élément `v` de type `VectHdl<Plus<...,...»` est la somme des ième coefficients des ses deux composants.

Une grande attention a aussi été donnée à la gestion de l'espace mémoire. Et les copies inutiles de gros objets ont été limitées en ayant des fonctions retournant des pointeurs, qui peuvent être facilement détruits. Pour ce dernier point des classes spéciales ont été implémentées, contenant un pointeur sur le résultat. Elles sont utilisées uniquement comme type de retour des fonctions. Ainsi, la copie d'un tel objet correspond juste à une opération sur un pointeur.

3.2.2 Architecture

La librairie comprend trois niveaux de structures :

- Les **conteneurs**, des petites classes correspondant à la représentation interne des objets.
- Les **interfaces** qui sont des classes généralement paramétrées par le type de conteneur et qui spécifie comment voir celui-ci.
- Les **catégories** qui sont des familles d'interfaces ou de classes partageant des propriétés communes. Elles ne nécessitent pas forcément une implémentation propre, mais correspondent à des propriétés de classes.

A titre d'exemple `UPoly<toeplitz<double> >` est un polynôme univariable dense, représenté de manière interne par le conteneur `toeplitz<double> >`. `MatStr<toeplitz<double>` est une matrice structurée de type Toeplitz avec les implémentations spécialisées correspondantes (comme, par exemple, une multiplication matrice-vecteur par FFT) sur le même conteneur. Alors que la première classe appartient à la catégorie `UPOLYNOMIAL`, la deuxième fait partie de la catégorie `MATRIX`.

Chapitre 4

Les ingrédients des algorithmes

Nous présentons dans cette partie plusieurs exemples utilisant des fonctions qui sont utilisées par les algorithmes explicités precédemment. Nous en profitons pour souligner certaines particularités de l'implémentation.

4.1 Exemple de code générique

Nous proposons, ici, un exemple d'algorithme mettant en valeur la clarté et l'écriture intuitive de code.

Réciproque d'un polynôme

Cet algorithme est tiré de [4]. Il s'agit d'un algorithme de division, de complexité $nlog(n)$. Il permet, en particulier, de réduire la complexité de la résolution $T.X = V$, où T est une matrice de Toeplitz, triangulaire inférieure, d'un coùt de $nlog^2(n)$ à $nlog(n)$. En effet, une fois les coefficients mis dans un vecteur t, l'inversion de T n'est plus que le calcul de l'inverse de t.

Nous supposons que $p[0] \neq 0$. Pour ce calcul, nous considérons l'équation $f(w) = P - \frac{1}{w} = 0$, où $P = p(z)$, $w = w(z) = \frac{1}{p(z)}$ sur l'anneau des séries formelles en z, et applique des itérations de Newton à cette équation. Nous obtenons ainsi : $w_0 = \frac{1}{p[0]}$, and $w_{j+1} = w_j(2-Pw_j)$ où $j = 0,1,\dots$ En réduisant les différents polynômes modulo $z^{2^{j+1}}$ à chaque pas, nous effectuons deux convolutions de deux couples de vecteurs de dimension au plus 2^{j+1} par pas.

Nous présentons ici une fonction générique, où le type générique R est vu comme un polynôme grâce au dernier argument passé.
Un des avantages de ce style de programmation réside dans le fait que tout type de données comportant les opérations élémentaires et des champs `index_type` et `coef_type` peut être passé, en particulier tout vecteur de la librairie **ALP**. L'utilisation de cette fonction se fait en voyant la structure de donnée comme un polynôme (`UPOLYNOMIAL`).

```
template <class R>
inline void reciprocal(R & w,const R & p, const UPOLYNOMIAL)
{
  typedef typename R::index_type index_type;
  typedef typename R::coef_type C;
```

```
  const index_type deg = p.degree()+1;
  R w0(deg),v(deg),xp(p);
  index_type K = (index_type) (log(p.degree()+1)/log(2)),j=1;
  C p0 = xp[deg-1];

  xp /= p0;
  w0 =xp/xp[0];
  v = w0;
  v *= C(-1.0);
  v[0]+=C(2);
  w=v/xp[0];
  w0=w;
  while (j <= K) {
    reduce(w0,pow(2,j+1));
    w=xp*w0;
    v =w;
    v*=C(-1.0);
    v[0]+=C(2);
    reduce(v,pow(2,j+1));
    w=w0*v;
    w0=w;
    j++;
  }
   reduce(w,deg);
   w/=p0;
}
```

Voici le programme principal appelant cette fonction, ainsi que la sortie :

```
#include "_UPoly.H"

int main(int argc, char** argv) {
  double u[]={-1,3,4,6,7,8};
  POL q(6,u), rec(9);

  cout << " q = " << q <<endl;
  reciprocal(rec,q);
  cout << " L'inverse de q : " << rec << endl;
}
```

La sortie donne :

```
q = (-1)+(3)*x+(4)*x^2+(6)*x^3+(7)*x^4+(8)*x^5
L'inverse de q : (-1)+(-3)*x+(-13)*x^2+(-57)*x^3+(-248)*x^4+(-1079)*x^5
```

4.2 Utilisation de librairie externe : GMP

Cet exemple se propose de résoudre un problème de calibration de robots parallèle en utilisant une arithmétique de grande précision. De plus, cette résolution se fait par un appel à l'implémentation générique de la fonction `Svd`, fonction qui sera utilisée dans l'implémentation de la méthode de résolution par le Bézoutien.

```
#include "_Matrix.H"
#include "_BFloat.H"
#include "linear/Svd.H"

#define Coef Scl<MP_FLOAT>
#define VECT VectStd<array1d<Coef> >
#define MATT  MatStd<array2d<Coef> >

int main (int argc, char **argv)
{
  MATT A("TestAlpSSerr");

int n= A.nbrow(),m=A.nbcol();
int mm=min(n,m);

MATT U(n,n), V(m,m);
VECT S(mm),Sol(m);

Svd(S,A,U,V);

Sol=V.col(m-1);
Coef c(Sol[0]);
Sol/=c;
cout << Sol << endl;
}
```

La solution obtenue est :

```
0.1e1, 0.9908069458630016923e0, 0.1010549694614195751e1, 0.1111111111109947332e1,...
```

La solution théorique est :

```
 1.0e+00 9.9080694586312567651 0e-01 1.0105496946141032665 9e+00 1.11111111111111116045e+00 ...
```

4.3 Généricité et efficacité

Nous montrons ici un exemple de résolution d'une équation polynomiale en une variable. Il s'agit d'un polynôme de degré10^6, ce qui permet de souligner une certaine puissance dans la méthode utilisée. Celle-ci est la méthode de résolution de **Sebastio e Sylva** dérivée de la méthode des puissances. L'implémentation est une fonction générique qui sera utilisée dans le cadre de la méthode de résolution avec le Bézoutien pour le cas multivariable. inverses. Les différents calculs effectués par cette fonction utilisent une FFT pour les multiplications.

```
#include "util/basic.H"
#include "arithm/Arith.H"
#include "_UPoly.H"
```

```
#include "polynomial/UPolySpl.H"
#include "polynomial/Upar.H"
#include "algo/SolveUPol.H"
#include "CPUClock.h"

#define POL UPolySpl<upar<complex<double> > >
#define N 1000000


int main(int argc, char** argv) {
  complex<double> t[N];

  t[N-1]=complex<double>(1.0,0.0);
  for (unsigned int i = 0; i < N-1; i++) t[i] = complex<double> (rand()%100 *0.1,0.0);

  POL pol(N,t);
  complex<double> set, x0;

  x0 = complex<double> (0.4,0.6);
  CPUClock T;
  solve(set,pol,complex<double>(0.4,0.6),Sebastiao());
  T.stop();
  cout << "une racine : " << set << endl;
  cout << "Le temps mis est : " << T.time() << endl << endl;;
}
```

La sortie donne :

```
  une racine : (0.501551,0.772527)
  Le temps mis est : 2520
```

4.4 Structure à deux étages : calcul de Bézoutien

A partir d'un système d'équations polynomiales, nous construisons un ematrice dont le déterminant est le Bézoutien du système. Les coefficient de cette matrice sont des polynômes en les variables $\mathbf{x} = (x_1, \ldots, x_n)$ et $\mathbf{y} = (y_1, \ldots, y_n)$. Le déterminant est calculé à partir de la méthode de **Berkowitz** dont voici l'implémentation :

```
template<class R,class S>
typename R::coef_type deter(const R & M, const S & aux=S())
{
  typedef typename R::coef_type coef_type;
  typedef typename R::index_type index_type;
  coef_type d;
  R SM(M);

  d=Trace(SM,MATRIX());
  for (index_type i=2; i<=M.nbcol(); i++)
    {
      SM*=M;
```

```
      SM-=d*M;
      d=Trace(SM,MATRIX());
      d*=(S(typename S::coef_type(1.0/i)));
    }
  return d;
}
```

Voici maintenant l'exemple :

```
#include "_Matrix.H"
#include "_MPoly.H"
#include "algo/MBezout.H"

typedef Monom<double, dynamicexp<'x'> > Mon;
typedef MPoly<list<Mon>, Plex<Mon> >    Pol;
#define MAT MatStd<array2d<double> >

int main (int argc, char **argv) {
  list<Pol> l;
  int j=1,nb;
  MAT B;

  cin >> nb;

  while (j<=nb) {
    Pol Temp;
    cin >> Temp;
    l.push_back(Temp);
    j++;
    cout << Temp << endl;
  }
  B= MBezoutian(l,array2d<double>());
  cout << " La bezoutienne" << B << endl;
  return 0;
}
```

dont la sortie est :

```
(1)*x1*x3+(1)*x3+(1)*x2+(1)*x1+(1)
(1)*x3+(1)*x2+(1)*x1^2+(-1)
(1)*x3+(1)*x2^2+(1)*x1+(-1)
(1)*x3^2+(1)*x2+(1)*x1+(-1)

 Le bezoutien est :
((-1)*x3+(-1)*x2+(-1)*x1^2+(1))*y2*y3^2+((-1)*x2*x3+(1)*x3+(-1)*x1^2*x2
+(1)*x2+(1)*x1+(-1))*y3^2+((1)*x1*x3+(1)*x3+(1)*x2+(1)*x1+(1))*y1*y2*y3
+((-1)*x2*x3+(1)*x1*x3+(1)*x1*x2+(2)*x1)*y2*y3+((1)*x1*x2*x3+(1)*x2*x3
+(-1)*x3+(1)*x1*x2+(1)*x2+(-1)*x1+(1))*y1*y3+((1)*x1*x2*x3+(-1)*x1*x3
+(2)*x1*x2+(1)*x1^2+(-1)*x1+(-2))*y3+((-1)*x3^2+(-1)*x2+(-1)*x1
+(1))*y1^2*y2+((1)*x2*x3+(1)*x1*x3+(2)*x3+(-1)*x1*x2)*y1*y2
+((1)*x1^2*x3^2+(1)*x1*x2*x3+(-1)*x2*x3+(1)*x3+(-1)*x1*x2
+(1)*x2+(1)*x1+(-3))*y2+((-1)*x2*x3^2+(1)*x3+(-1)*x1*x2+(1)*x2
+(1)*x1+(-1))*y1^2+((1)*x3^2+(1)*x1*x2*x3+(2)*x2*x3+(-1)*x1*x3
+(-1)*x3+(-2))*y1+((1)*x1^2*x2*x3^2+(-1)*x1*x3^2+(1)*x2*x3
```

```
+(-1)*x1^2*x3+(3)*x1*x3+(-3)*x3+(1)*x1*x2+(-3)*x2+(-3)*x1+(5))

 La bezoutienne
        [ 0 0 0 0 0 0 0 0 0 -1 0 0 -1 -1 0 1 ]
        [ 0 0 0 0 0 0 -1 0 0 1 -1 0 1 0 1 -1 ]
        [ 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 ]
        [ 0 0 0 0 0 0 -1 0 1 0 0 1 0 0 2 0 ]
        [ 0 0 0 0 0 1 1 0 0 -1 0 1 1 0 -1 1 ]
        [ 0 0 0 0 0 1 0 0 -1 0 0 2 0 1 -1 -2 ]
        [ 0 0 0 0 -1 0 0 0 0 0 0 0 -1 0 -1 1 ]
        [ 0 0 0 0 0 0 1 0 1 2 0 -1 0 0 0 0 ]
        [ 0 0 1 0 0 1 -1 0 0 1 0 -1 1 0 1 -3 ]
        [ 0 -1 0 0 0 0 0 0 0 1 0 -1 1 0 1 -1 ]
        [ 0 0 0 0 1 1 2 0 -1 -1 0 0 0 0 0 -2 ]
        [ 1 0 0 -1 0 0 1 -1 3 -3 0 1 -3 0 -3 5 ]
```

4.5 Mélange de calculs formels et numériques : les équations de Kruppa

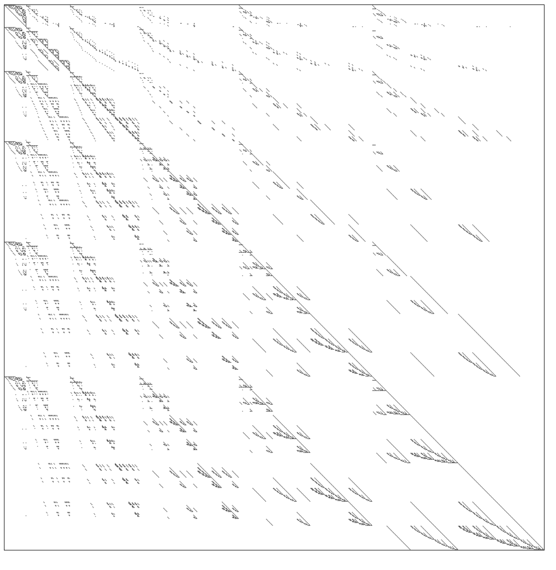
Nous nous intéressons maintenant à un système surdéterminé qui apparait dans le problème de reconstruction en vision, où tout couple de points, en correspondance avec les images, donnent une équation polynomiale. Ce qui fait que nous pouvons trouver autant d'équations que nous voulons, il n'existe qu'une solution. L'exemple donné ici, est un système de 6 quadriques dans un espace de dimension 5, venant du problème d'autocalibration de **Kruppa**. Pour plus de détails sur ces problèmes on peut lire [15].

```
#define RepMMon "RepDTab.H"
#define RepMPol "RepList.H"
#include "polynomial/ALP_include.H"
#include "linear/array1d.H"
#include "linear/Vector.H"
#include "polynomial/MPSolve.H"
#include "CPUClock.h"
#define DsVector(C) VectStd<array1d<C> > >
#define Mon MMon<double,DynamicArray>
#define Pol MPoly<list<Mon >, Dlex<Mon > >
#define YYSTYPE Pol

#include <string.h>
#include <math.h>

list<Pol> l;

#include "polynomial/parser.tab.h"
#include "polynomial/lex.yy.c"
#include "polynomial/parser.tab.c"
int yywrap(void);

int main (int argc, char **argv)
{
```

```
        yyparse();
        int nb =0;
        for (list<Pol>::const_iterator i = l.begin(); i != l.end(); ++i){
          nb++;
          cout << *i << endl; }

        Pol lp[nb];
        int j =0;
        for (list<Pol>::const_iterator i = l.begin(); i != l.end(); ++i)
          {if (j < nb) lp[j]=*i; j++;}
        lp[2]+=lp[1];
        DsVector(Pol) L(3,lp);
        CPUClock T;
        cout << MPSolve(L,compcol_mat_double()) << endl ;
        cout << "Le temps mis est : " << T.time() << endl;
        return 0;
}
```

La sortie est :

```
    Dimension of S : 792
    Number of non-zero entries : 16600
    Dimension of A : 1

    [-47.5396, -18.0428, 60.7231, 26.3451, 47.8788]

    Le temps mis est : 1.4
```



System of Kruppa equations

4.6 Connexion avec la librairie Lapack

Voici un exemple d'utilisation de la librairie **Lapack**. De plus, ce programme utilise une méthode `rank` de la librairie **ALP**, dont l'implémentation fait appel à une fonction de décomposition QR de la **Lapack**. Cette fonction `rank` sera aussi utilisée pour l'algorithme de résolution par le Bézoutien.

```
#include "_Lapack.H"

typedef VectStd<array1d<double> > VCT;
typedef MatSpl<lapack<double> >           MAT;

int main(int argc, char** argv) {
  double v[]={1,2,3,4,1,4,3,3,4,11,5,5,6,8,12,0,1,9,7,2};
  unsigned int r =0;

  MAT A(5,4,v);

  cout <<A<< endl;

  Lib<lapack<double> >::rank(A.rep,r);

  cout << endl << "Le rang de A est : " << r << endl;
};
```

En sortie :

```
        [ 1 4 5 0 ]
        [ 2 3 5 1 ]
        [ 3 3 6 9 ]
        [ 4 4 8 7 ]
        [ 1 11 12 2 ]

        Le rang de A est : 3
```

4.7 Calcul de valeurs propres

Ceci est un exemple de calcul de valeurs propres, qui peut être utilisé pour la méthode de résolution par la recherche de valeurs et vecteurs propres.

```
#define _Matrix_
#define _lapack_
#define _Maple_
#define _Latex_

#include "require.H"
#include "linear/Eigen.H"

#define VCT VectStd<array1d<double> >
#define MAT MatSpl<lapack<double>,Lib<lapack<double> > >
#define MATC MatSpl<lapack<complex<double> >,Lib<lapack<complex<double> > >

int main(int argc, char** argv) {

 lapack<double> r("mat33.in");
 MAT A(r);
```

```
  double v[]={1,2,0,2,1,0,0,0,1};
  MAT B(3,3,v);

  cout << A << endl<< B << endl;
  cout << Eigenval(A) <<endl <<endl;
  cout << Eigenval(B) <<endl <<endl;
};
```

En sortie :

```
 [ 58 9 2 ]
 [ 186 383 96 ]
 [ -912 -1551 -388 ]

 [ 1 2 0 ]
 [ 2 1 0 ]
 [ 0 0 1 ]

 [(50,0), (1,0), (2,0)]

 [(3,0), (-1,0), (1,0)]
```

Bibliographie

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

[2] E. Bézout. *Théorie Générale des Équations Algébriques.* Paris, 1779.

[3] D. Bini and V.Y. Pan. *Polynomial and Matrix Computations*, volume 2: Selected Topics. Birkhäuser, Boston, 1998. To appear.

[4] Pan V.Y. Bondyfalat D., Mourrain B. Controlled iterative methods for solving polynomial systems. *Proceedings for Issac'98, ACM Press*, 1998.

[5] J.P. Cardinal. On two iterative methods for approximating the roots of a polynomial. In J. Renegar, M. Shub, and S. Smale, editors, *Proc. AMS-SIAM Summer Seminar on Math. of Numerical Analysis, (Park City, Utah, 1995)*, volume 32 of *Lectures in Applied Math.*, pages 165–188. Am. Math. Soc. Press, 1996.

[6] J.P. Cardinal and B. Mourrain. Algebraic approach of residues and applications. In J. Renegar, M. Shub, and S. Smale, editors, *Proc. AMS-SIAM Summer Seminar on Math. of Numerical Analysis, (Park City, Utah, 1995)*, volume 32 of *Lectures in Applied Math.*, pages 189–210. Am. Math. Soc. Press, 1996.

[7] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1992.

[8] P. Dietmaier. The Stewart-Gough platform of general geometry can have 40 real postures. In *ARK*, pages 7–16, Strobl, 294JuinJuillet1998.

[9] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proc. of Lond. Math. Society*, 6:49–69, 473–492, 1908.

[10] G. Dos Reis. Vers une nouvelle approche du calcul scientifique en c++. Rapport de Recherche 3362, INRIA, 1998.

[11] D. Eisenbud. *Commutative Algebra with a view toward Algebraic Geometry*, volume 150 of *Graduate Texts in Math.* Springer-Verlag, 1994.

[12] M. Elkadi and B. Mourrain. Approche effective des résidus algébriques. Rapport de Recherche 2884, INRIA, 1996.

[13] M. Elkadi and B. Mourrain. Some applications of bezoutians in effective algebraic geometry. Rapport de Recherche 3572, INRIA, 1998.

[14] O. Faugeras. *Three-Dimensional Computer Vision: a Geometric Viewpoint.* mit-press, 1993.

[15] G.H. Golub and C.F. Van Loan. *Matrix computations.* John Hapkins, Univ. Press, Baltimore, Maryland, 3^{rd} edition, 1996.

[16] D. Lazard. Generalized Stewart Platform: How to compute with rigid motions? In *IMACS Symp. on Symbolic Computation*, pages 85–88, Lille, 14-17Juin1993.

[17] F.S. Macaulay. Some formulae in elimination. *Proc. London Math. Soc.*, 1(33):3–27, 1902.

[18] B. Mourrain. Alp, algèbre linéaire pour les polynômes, 1996-1999. Bibliothèque de classes C++, pour les manipulations de matrices issues de problèmes polynomiaux et leur résolution `http://www.inria.fr/saga/mourrain/ALP/`.

[19] B. Mourrain. The 40 generic positions of a parallel robot. In M. Bronstein, editor, *ISSAC'93*, ACM press, pages 173–182, Kiev (Ukraine), Juillet1993.

[20] Pan V.Y. Mourrain B. Multivariate polynomials, duality and structured matrices. 1997.

[21] F. Ronga and T. Vust. Stewart platforms without computer?, 1992. Preprint.

[22] J.J. Sylvester. On a theory of syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's functions, and that of the greatest algebraic common measure. *Philosophical Trans.*, 143:407–548, 1853.

[23] T. Veldhuizen. Blitz++, 1999. `http://monet.uwaterloo.ca/blitz/`.