Bases tensorielles de Bernstein et solveurs

Tahari A.K.[1] — Fünfzig C.[2] — Michelucci D.[2] — Foufou S.[2,3] — Ait-Aoudia S.[4]

1. Université Amar Thelidji U.A.T. Laghouat Algérie
k.tahari@mail.lagh-univ.dz

2. Laboratoire LE2I, UMR CNRS 5158, Université de Bourgogne
<christoph.fuenfzig; dmichel; sfoufou>@u-bourgogne.fr

3. CSE, CENG, Qatar University, P.O. Box 2713, Doha, Qatar
sfoufou@qu.edu.qa

4. Institut National d'Informatique, Alger, Algérie
s_ait_aoudia@ini.dz

RÉSUMÉ. *Les bases tensorielles de Bernstein permettent de calculer de bons encadrements des valeurs des polynômes sur des pavés, et de résoudre les systèmes polynomiaux rencontrés en synthèse d'images, en modélisation géométrique, et en résolution de contraintes géométriques. Cet article présente deux types de solveurs. Le premier est classique, et limité aux petits systèmes de 6 ou 7 inconnues. Le second est nouveau et utilisable sur des systèmes de taille arbitraire. Il nécessite la définition d'un nouvel objet mathématique, le polytope de Bernstein.*

ABSTRACT. *Tensorial Bernstein bases can be used to compute sharp ranges of the values of polynomials over a box, and to solve systems of polynomial equations in Computer Graphics, Geometric Modelling, Geometric Constraints Solving. Two kinds of solvers are presented. The first is classical, and applies to small systems, up to 6 or 7 unknowns. The second is new and applies to systems of arbitrary size. It requires a new mathematical object, the Bernstein polytope.*

MOTS-CLÉS : *intervalle, encadrement, bases de Bernstein, contraintes géométriques, systèmes d'équations, solveur, bissection, polynôme univarié, polynôme multivarié.*

KEYWORDS: *interval, range, Bernstein bases, geometric constraints, systems of equations, solver, Bézier curves and surfaces, bisection, univariate polynomial, multivariate polynomial.*

1. Introduction

Aujourd'hui plusieurs applications de l'informatique graphique ou géométrique nécessitent la résolution de systèmes non linéaires. Le calcul des images de synthèse par lancer de rayons nécessite le calcul des points d'intersection entre des rayons (des demi-droites) et des surfaces définies par une ou plusieurs équations algébriques. La modélisation géométrique, et la Conception et Fabrication Assistées par Ordinateur (CFAO) ont besoin de calculer les points d'intersection entre de telles surfaces. Il en résulte des systèmes d'équations algébriques de petites tailles. Enfin, tous les modeleurs géométriques utilisés en CFAO fournissent aujourd'hui la possibilité de modéliser des objets géométriques, ou de dimensionner des pièces, par un ensemble de contraintes géométriques, telles que des relations d'incidences, de tangences, des spécifications d'angles ou de distances entre des éléments géométriques : points, droites, plans, cercles, sphères, etc (Jermann *et al.*, 2007). La résolution de ces contraintes géométriques nécessite la résolution de systèmes d'équations algébriques non linéaires. Les sous systèmes irréductibles peuvent être de grande taille (plus d'une dizaine d'inconnues et d'équations) et sont résolus par des méthodes numériques : citons l'itération de Newton-Raphson, l'homotopie (ou continuation), les méthodes de Newton par intervalles (Michelucci, 1996; Durand, 1998; Sommese *et al.*, 2005; Kearfott, 1996), et les solveurs utilisant les bases tensorielles de Bernstein ou d'autres bases géométriques (Garloff *et al.*, 2001; Mourrain *et al.*, August 2005; Sherbrooke *et al.*, 1993; Elber *et al.*, 2001; Reuter *et al.*, 2008; Michelucci *et al.*, July 2008).

Les bases tensorielles de Bernstein sont utilisées en informatique graphique pour calculer de bons encadrements des valeurs d'un polynôme multivarié $f(x)$, $x = (x_1, \ldots x_n)$, sur l'hypercube $x \in [0,1]^n$ ou sur un pavé $u_i \leq x \leq v_i$ de $\mathbb{R}^n$: le polynôme $f(x)$ est exprimé dans la base tensorielle de Bernstein, et $f([0,1]^n)$ est contenu dans l'intervalle défini par le plus petit et le plus grand de ces coefficients. Contrairement aux coefficients dans la base habituelle : $(1, x_1, x_1^2, \ldots) \times (1, x_2, x_2^2, \ldots) \times (1, x_3, x_3^2, \ldots) \times \ldots$, dite base canonique, les coefficients dans la base tensorielle de Bernstein dépendent du pavé contenant l'argument x. Un pavé est décrit par un couple $[u, v]$ où $u \in \mathbb{R}^n$ et $v \in \mathbb{R}^n$, et est l'ensemble des points $x = (x_i)$ de $\mathbb{R}^n$ tels que $u_i \leq x_i \leq v_i$. La méthode de Paul de Casteljau permet de calculer ces coefficients pour un sous pavé, sans repasser par la base canonique. Tout cela est aujourd'hui classique en informatique graphique ; par exemple les bases de Bernstein sont utilisées de façon routinière pour calculer des couvertures de courbes ou de surfaces algébriques implicites (Martin *et al.*, 2002). Récemment, les propriétés des bases tensorielles de Bernstein, ou d'autres bases géométriques (splines rationnelles) ont été utilisées, au delà du 2D et du 3D, pour des solveurs de systèmes d'équations algébriques non linéaires (Garloff *et al.*, 2001; Mourrain *et al.*, August 2005; Sherbrooke *et al.*, 1993; Elber *et al.*, 2001; Reuter *et al.*, 2008; Michelucci *et al.*, July 2008).

Toutefois, les polynômes qui sont habituellement creux dans la base canonique deviennent denses dans la base tensorielle de Bernstein. Par exemple, le monôme 1 s'écrit $(\mathcal{B}_0^{(d_1)}(x_1) + \ldots \mathcal{B}_{d_1}^{(d_1)}(x_1)) \times \ldots \times (\mathcal{B}_0^{(d_n)}(x_n) + \ldots \mathcal{B}_{d_n}^{(d_n)}(x_n))$ dans la base

tensorielle de Bernstein, où $\mathcal{B}_i^{(d_k)}(x_k) = \binom{d_k}{i} x_k^i (1-x_k)^{d_k-i}$ est le $i$ème polynôme dans la base des polynômes de Bernstein de degré d_k. De même, un polynôme linéaire $p(x_1, \ldots x_n)$ nécessite un nombre exponentiel 2^n de coefficients dans la base tensorielle de Bernstein, alors qu'il suffit de $n+1$ coefficients dans la base canonique. Un polynôme quadratique nécessite 3^n coefficients dans la base tensorielle de Bernstein, et $O(n^2)$ coefficients dans la base canonique, pour les monômes : $x_i^2, x_i x_j, x_i$ et 1.

Ceci rend les solveurs classiques de Bernstein inutilisables pour les systèmes de plus de 6 ou 7 équations. Or les contraintes géométriques, surtout en 3D, fournissent des systèmes irréductibles bien plus grands. Par exemple un icosaèdre régulier, ou non régulier (20 triangles, 12 sommets, 30 arêtes), peut être spécifié par la longueur de ses arêtes, ce qui donne un système non réductible de 30 équations ; de même leur dual, le dodécaèdre régulier ou non (12 faces pentagonales, 20 sommets, 30 arêtes) peut être spécifié par les longueurs de ses 30 arêtes et les coplanarités de ses 12 faces pentagonales. Ces systèmes sont quadratiques. En utilisant les notations habituelles, les équations sont : $a_k^2 + b_k^2 + c_k^2 = 1, a_k x_i + b_k y_i + c_k z_i + d_k = 0, (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 = D_{ij}^2$, où (x_i, y_i, z_i) sont les coordonnées du sommet i, où $a_k x + b_k y + c_k z + d_k = 0$ est l'équation du plan de la face numéro k, et D_{ij} est la longueur de l'arête ij. Pour obtenir un système bien contraint, 3 sommets peuvent être fixés, par exemple un sommet est fixé à l'origine, un premier voisin est fixé sur l'axe des x, et un second est fixé dans le plan xy. Ces systèmes sont gros, et quasiment irréductibles. Ils ne peuvent être résolus par les solveurs de Bernstein classiques, à cause de la taille exponentielle de la représentation des polynômes dans la base tensorielle de Bernstein.

Cet article propose un nouvel algorithme polynomial pour résoudre la difficulté relative à la taille de la représentation des polynômes dans la base tensorielle de Bernstein. Jusqu'à maintenant, la seule solution consistait à ne pas utiliser la base tensorielle de Bernstein, mais la base simplicielle (dite aussi homogène) de Bernstein (Michelucci *et al.*, 2006b; Reuter *et al.*, 2008).

La première partie de l'article en §2 décrit les solveurs classiques fondés sur les bases de Bernstein ; ces solveurs ne peuvent traiter que des petits systèmes, avec 6 à 7 inconnues au plus, comme il en apparaît en synthèse d'images, par exemple pour le lancer de rayons sur des surfaces paramétriques. La seconde partie de l'article, en §3 présente un nouveau type de solveur : il évite cette limitation en définissant le polytope de Bernstein et en recourant à la programmation linéaire (Chvatal, 1983). Ce deuxième type de solveurs est utilisable sur des systèmes de taille arbitraire : la réduction d'un pavé, en préservant la ou les racines contenues, est effectuée en temps polynomial, si bien que chaque racine réelle simple est isolée en temps polynomial. Bien sûr, ce solveur nécessite toujours un temps exponentiel quand le pavé étudié contient une quantité exponentielle de racines.

2. Solveur et bases tensorielles de Bernstein

2.1. *Le principe*

Cette partie présente les solveurs classiques utilisant les bases de Bernstein.

Soit $f(x) = 0$, avec $x = (x_1, \dots x_n)$ et $f = (f_1, \dots f_n)$ le système à résoudre. Les solveurs actuels utilisant les bases de Bernstein trouvent toutes les racines réelles contenues dans un pavé donné $[u, v]$ de $\mathbb{R}^n$. Leur principe est le même que celui des solveurs de type Newton par intervalles : le pavé initial est empilé ; tant que la pile de pavés est non vide, le pavé en sommet de pile est dépilé et étudié.

Pour étudier un pavé $p = [u, v] = \{x | u_i \leq x_i \leq v_i\}$, le solveur commence par calculer un pavé $p' = [u', v'] \subset [u, v]$ inclus dans p et qui contient les mêmes racines que p. Le paragraphe §2.4.4 présente la méthode utilisée pour la réduction de p en p'. Après la réduction, trois cas sont possibles :

- soit p' est vide, et alors p ne contient aucune racine ;

- soit p' est trop petit pour être encore subdivisé ; alors, p' contient une racine simple (le jacobien est de rang n dans p'), ou plusieurs racines simples proches, ou une racine multiple : des tests permettant de prouver l'existence et l'unicité peuvent éventuellement être utilisés ; ils sont donnés en §2.5 ; p', après un éventuel étiquetage (contient-il une racine simple ? le jacobien s'annule-t-il dans p' ? etc), est inséré dans une liste des solutions du système ; certains solveurs permettent de préciser la racine simple contenu dans un pavé en appliquant quelques itérations de Newton à partir du centre du pavé ;

- soit le pavé est encore gros, mais n'est plus significativement réductible ; il contient probablement plusieurs racines ; il est coupé en deux, le long de son côté le plus long, ou bien le long du côté qui a été le moins réduit, et les deux sous pavés résultant de la subdivision sont empilés. La subdivision est le seul moyen de séparer plusieurs racines distinctes ; mais elle a un coût exponentiel ; l'idéal est de ne l'utiliser que pour séparer les racines.

Les solveurs utilisant les propriétés des bases tensorielles de Bernstein (ou les solveurs de Bernstein par concision) ressemblent beaucoup aux solveurs de type Newton par intervalles. Cependant, les solveurs de type Newton par intervalles, comme celui de ALIAS (COPRIN, 2004), utilisent une autre réduction que celle, spécifique aux solveurs de Bernstein qui est présentée en §2.4.4. Pour réduire le pavé p, les solveurs de type Newton par intervalles itèrent $p \leftarrow p \cap n(p)$ où $n(p)$ est un encadrement de $n(x)p$: pour $x \in p$. $n(x) = x - Mf(x)$ est l'itération de Newton, et M est une inverse approximative du jacobien du polynôme f au centre du pavé p.

Les solveurs de type Newton par intervalles utilisent typiquement une arithmétique d'intervalles centrée pour calculer l'encadrement de $n(x \in p)$: pour calculer l'encadrement d'un polynôme $f(x \in p)$ sur un pavé p, ils utilisent : $f(x \in p) \subset f(c) + (p - c)f'(p)$, où c est le centre du pavé p, et $f'(p)$ est un encadrement de $f'(x \in p)$ calculé avec une arithmétique d'intervalles naïve. Les bases tensorielles de

Bernstein fournissent de meilleurs encadrements que l'arithmétique d'intervalles, et de meilleures réductions. Cela est illustré par les figures 5, et 14.

Le reste de cette partie est structuré ainsi : Le paragraphe §2.2 rappelle les définitions et les propriétés essentielles des bases tensorielles de Bernstein, présente la conversion de la base canonique à la base de Bernstein, et montre l'utilisation de ces bases pour la conversion et l'encadrement des polynômes univariés et multivariés. Le paragraphe §2.3 donne les formulations matricielles pour la méthode de de Casteljau et pour les conversions entre bases. Le paragraphe §2.4 présente quelques algorithmes utilisant les bases tensorielles de Bernstein pour le calcul de couvertures de courbes implicites, pour l'isolation des racines réelles de polynômes univariés, pour le calcul de l'enveloppe convexe en 2D, ainsi que pour le préconditionnement et la réduction de pavés. Le paragraphe §2.5 présente quelques tests utiles pour prouver l'existence ou l'absence de racines, et l'unicité de racine dans un pavé. Les principales limitations des solveurs fondés sur les bases tensorielles de Bernstein sont rappelées dans le paragraphe §2.6.

2.2. *Définitions et propriétés essentielles*

Soit $\mathcal{C}(k,d) = \binom{d}{k} = \frac{d!}{(d-k)!\ k!}$ le nombre de sous-ensembles de k éléments dans un ensemble de d éléments, nous utilisons la convention : $\mathcal{C}(k,d) = 0$ quand k ou d est négatif.

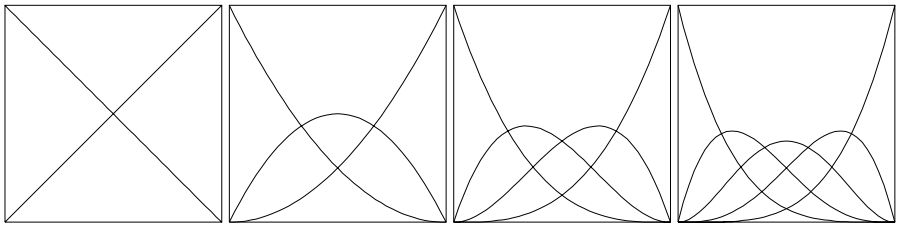

Figure 1. *Les polynômes de Bernstein pour les degrés 1, 2, 3, 4, de gauche à droite. Chaque carré est le carré unité. Les valeurs maximum et minimum de $\mathcal{B}_i^{(d)}(x)$ se produisent en $x = i/d$.*

Les $d+1$ polynômes de Bernstein $\mathcal{B}_i^{(d)}(x), i = 0, \ldots d, x \in \mathbb{R}$, de degré d (aussi notés $\mathcal{B}_i(x)$ quand le degré est sous-entendu) constituent une base des polynômes de degré d. Par définition :

$$\mathcal{B}_i^{(d)}(x) = \binom{d}{i} x^i (1-x)^{d-i}$$

Les graphes $(x, \mathcal{B}_i^{(d)}(x))$ sont représentés en Figure 1 pour $d = 1, 2, 3, 4$. Ces polynômes apparaissent dans l'expansion de Newton de $(x + (1 - x))^d$:

$$(x + (1 - x))^d = 1 = \sum_{i=0}^{d} \binom{d}{i} x^i (1 - x)^{d-i} = \sum_{i=0}^{d} \mathcal{B}_i^{(d)}(x)$$

La conversion entre la base canonique : $(1, x, x^2, \ldots x^d)$ et la base de Bernstein : $(\mathcal{B}_0^{(d)}(x), \ldots \mathcal{B}_d^{(d)}(x))$ est une transformation linéaire, représentable par une matrice carrée d'ordre $d + 1$. Une formule classique est (Farin, 1988) :

$$x^k = (1/\mathcal{C}(k, d)) \sum_{i=k}^{d} \mathcal{C}(k, i)\, \mathcal{B}_i^{(d)}(x)$$

On en déduit, pour $k = 1$:

$$x = (1/d) \times \sum_{i=0}^{d} i\, \mathcal{B}_i^{(d)}(x)$$

et pour $k = 0$, on retrouve la formule :

$$x^0 = 1 = \sum_{i=0}^{d} \mathcal{B}_i^{(d)}(x)$$

Les propriétés essentielles de la base de Bernstein résultent de leur définition ; la somme $\sum_{i=0}^{d} \mathcal{B}_i^{(d)}(x)$ égale 1 ; pour $x \in [0, 1]$, ils sont tous positifs ou nuls. Il en résulte que pour $x \in [0, 1]$, $p(x) = \sum_i p_i \mathcal{B}_i^{(d)}(x)$ est une combinaison linéaire convexe des coefficients $p_i, i = 0, \ldots n$.

Pour un polynôme $p(x) \in \mathbb{R}[x]$, chaque coefficient p_i est un nombre réel, et $p(x \in [0, 1])$ se trouve dans l'enveloppe convexe des p_i, qui est l'intervalle $[\min_i p_i, \max_i p_i]$. Cet encadrement est précis, et sa borne inférieure, ou supérieure, est atteinte quand elle se produit en $i = 0$ ou en $i = d$. En effet, $\mathcal{B}_0^{(d)}(0) = 1$ et $\mathcal{B}_{i>0}^{(d)}(0) = 0 \Rightarrow p(0) = p_0$; d'autre part, $\mathcal{B}_d^{(d)}(1) = 1$ et $\mathcal{B}_{i<d}^{(d)}(1) = 0 \Rightarrow p(1) = p_d$.

Si les p_i sont des points de $\mathbb{R}^2$, ou de $\mathbb{R}^3$, alors l'ensemble $p(x \in [0, 1])$ décrit un arc de courbe, appelée courbe de Bézier ; cet arc est inclus dans l'enveloppe convexe des coefficients p_i, qui sont appelés points de contrôle.

Exemple : comme $x = 0\,\mathcal{B}_0(x) + 1/d\,\mathcal{B}_1(x) + 2/d\,\mathcal{B}_2(x) + \ldots d/d\,\mathcal{B}_d(x)$, la courbe polynomiale $(x, y = p(x))$, avec $x \in [0, 1]$, est incluse dans l'enveloppe convexe de ses points de contrôle $(i/d, p_i)$. On en déduit que la courbe $(x, y = p(x))$ passe par le premier et par le dernier point de contrôle, $(0, p_0)$ et $(1, p_d)$.

Contrairement aux coefficients dans la base canonique usuelle : $(1, x, x^2, \ldots x^d)$, les points de contrôle (*i.e.,* les coefficients dans la base de Bernstein, ou coefficients

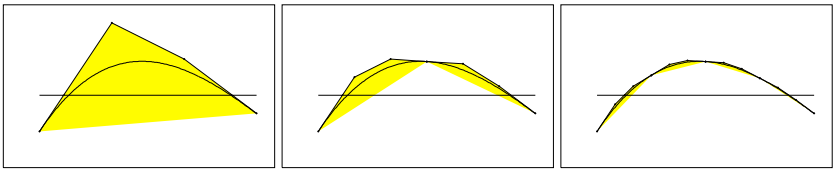

Figure 2. *Deux itérations de la méthode de de Casteljau sur une courbe de Bézier* $(x, y = f(x))$*, où* f *est un polynôme de degré 3. Les enveloppes convexes des points de contrôle sont affichées.*

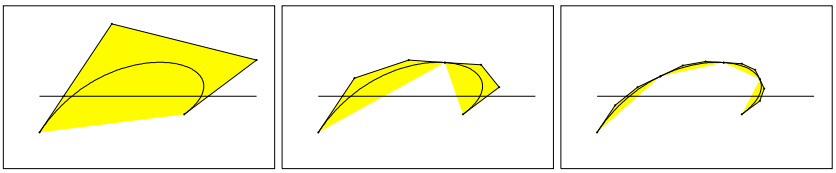

Figure 3. *Deux itérations de la méthode de de Casteljau sur une courbe de Bézier* $(x = f_1(u), y = f_2(u))$*,* $u \in [0, 1]$*, de degré 3. Il y a donc 4 points de contrôle* $p_i = (x_i, y_i)$*. Les enveloppes convexes des points de contrôle sont affichées.*

de Bernstein) de $p(x)$ dépendent de l'intervalle de l'argument x. Un algorithme classique dû à Paul de Casteljau (Farin, 1988) calcule les points de contrôle pour le sous intervalle $x \in [0, t]$ et le sous intervalle $x \in [t, 1]$, sans repasser par la base canonique. La valeur $t = 1/2$ est la plus souvent utilisée.

2.2.1. *Méthode de de Casteljau*

À partir des coefficients de Bernstein d'un polynôme $f(x), x \in [0, 1]$, la méthode de de Casteljau calcule les coefficients de Bernstein de sa moitié gauche : $x \in [0, 1/2]$, et de sa moitié droite $x \in [1/2, 1]$. Les Figures 2 et 3 montrent deux itérations de l'algorithme sur un polynôme $y = f(x)$ et sur une courbe de Bézier. La Figure 4 montre l'arbre des calculs de la méthode.

Soit $b^{(d)} = b = (b_0, b_1, \ldots b_d)$ les points de contrôle d'un polynôme univarié $y = f(x), x \in [0, 1]$. L'algorithme calcule le vecteur des milieux : $b^{d-1} = ((b_0 + b_1)/2, \ldots (b_i + b_{i+1}))/2 \ldots)$. Le vecteur $b^{(d-1)}$ a un élément de moins que $b^{(d)}$. Puis l'algorithme calcule de même $b^{(d-2)}$ à partir de $b^{(d-1)}$, etc, jusqu'à obtenir $b^{(0)}$, qui contient un seul élément. En formule : $b_i^{(k)} = (b_i^{(k+1)} + b_{i+1}^{(k+1)})/2$, pour $k = d-1, \ldots 0$ et $i = 0, \ldots k$.

Les premiers éléments de $b^{(d)}, b^{(d-1)}, \ldots b^{(0)}$ donnent le vecteur $l = (l_i)$ des coefficients de Bernstein de la moitié gauche $x \in [0, 1/2]$ de $f(x)$, et les derniers éléments

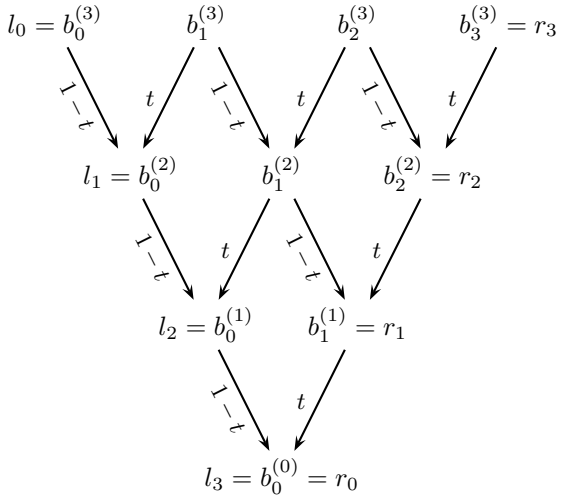

Figure 4. $b^{(d)} = (b_i^{(d)})$ *est le vecteur des coefficients de Bernstein d'un arc de degré* $d = 3$. *La Figure montre les interpolations linéaires calculées par la méthode de de Casteljau, pour trouver les vecteurs* $l = (l_i)$ *et* $r = (r_i)$ *des coefficients de Bernstein de la moitié gauche et de la moitié droite de l'arc*.

de $b^{(d)}, b^{(d-1)}, \ldots b^{(0)}$ donnent le vecteur $r = (r_i)$ des coefficients de Bernstein de la moitié droite $x \in [1/2, 1]$ de $f(x)$. En formule, la moitié gauche de $f(x)$ a pour coefficients de Bernstein : $l_i = b_0^{(d-i)}$ et la moitié droite a pour coefficients de Bernstein : $r_i = b_i^{(i)}$, pour $i = 0, \ldots d$.

Plutôt que des coefficients 1/2, 1/2, il est possible d'utiliser des coefficients $1-t, t$, ce qui fournit les points de contrôle pour le sous intervalle gauche $x \in [0, t]$, et pour le sous intervalle droit $x \in [t, 1]$. Il est également possible d'utiliser une valeur de t hors de $[0, 1]$, mais la stabilité numérique de l'algorithme est perdue ; quand $|t|$ croît, l'instabilité numérique augmente.

Ce calcul $b_i^{(k)} = (b_i^{(k+1)} + b_{i+1}^{(k+1)})/2$ s'applique en toute dimension, quand les $b_i^{(k+1)}, b_{i+1}^{(k+1)}$ sont des points et non plus des valeurs scalaires. Aussi la méthode de de Casteljau s'applique-t-elle en toute dimension.

2.2.2. *Conversion entre bases dans le cas univarié*

La conversion entre bases peut être effectuée matriciellement, comme en §2.3.2, mais pour le programmeur, la méthode qui suit est plus commode ; par exemple, elle s'étend au cas multivarié (§2.2.4). Il faut d'abord définir une fonction qui calcule la contribution du monôme x^k au i ième polynôme de Bernstein pour le degré d ; c'est

$$\mathcal{W}([0, 1], x^k, \mathcal{B}_i^{(d)}) = \mathcal{C}(k, i)/\mathcal{C}(k, d)$$

Jusqu'à maintenant, nous avons considéré que x appartient à l'intervalle $[0,1]$. Pour encadrer les valeurs du polynôme $f(x)$ pour $x \in [u,v]$, il suffit d'utiliser la transformation affine qui applique $[u,v]$ sur $[0,1]$: $x = u + (v-u)x'$. Quand x décrit $[u,v]$, x' décrit $[0,1]$. Donc, en posant $w = v - u$, la contribution de x^k, avec $x \in [u,v]$, à $\mathcal{B}_i^{(d)}(x)$, est : $\mathcal{W}([u,v], x^k, \mathcal{B}_i^{(d)}(x)) = \mathcal{W}([0,1], (u+(v-u)x)^k, \mathcal{B}_i^{(d)}(x))$:

$$\begin{aligned}
\mathcal{W}([u,v], x^k, \mathcal{B}_i^{(d)}(x)) &= \mathcal{W}([0,1], (u+(v-u)x)^k, \mathcal{B}_i^{(d)}(x)) \\
&= \mathcal{W}([0,1], \sum_{j=0}^{k} \mathcal{C}(j,k) \times w^j \times x^j \times u^{k-j}, \mathcal{B}_i^{(d)}(x)) \\
&= \sum_{j=0}^{k} \mathcal{C}(j,k) \times w^j \times u^{k-j} \times \mathcal{W}([0,1], x^j, \mathcal{B}_i^{(d)}(x)) \\
&= \sum_{j=0}^{\min(k,i)} w^j \times u^{k-j} \times \mathcal{C}(j,k) \times \mathcal{C}(j,i)/\mathcal{C}(j,d)
\end{aligned}$$

La formule est valable pour tout w, positif, négatif, ou nul. Ensuite, pour convertir de la base canonique vers la base de Bernstein, un polynôme $f(x)$ de degré d dans un pavé $x \in [u,v]$, il faut d'abord initialiser tous les coefficients de Bernstein à 0, puis pour chaque monôme canonique $c_k \times x^k$ du polynôme $f(x)$, et pour tous les i entiers dans $0, 1, \ldots d$, il faut incrémenter de $c_k \times \mathcal{W}([u,v], x^k, \mathcal{B}_i^{(d)})$ le coefficient du i ième polynôme de Bernstein $\mathcal{B}_i^{(d)}$. Les deux boucles sur x^k et le polynôme de Bernstein $\mathcal{B}_i^{(d)}$ peuvent être imbriquées dans n'importe quel ordre ; de même les valeurs pour les indices i et k peuvent être considérées dans n'importe quel ordre. Si les monômes $c_k x^k$ de $f(x)$ sont stockés dans une liste, alors la même puissance peut apparaître plusieurs fois : il n'est pas utile de réduire le polynôme.

2.2.3. *Base tensorielle de Bernstein pour les polynômes multivariés*

Pour illustrer le cas des polynômes multivariés, et par simplicité, nous considérons des polynômes bivariés en x et y. La base canonique est le produit cartésien de la base canonique en x : $(x^0, x^1, \ldots x^d)$ et de la base canonique en y : $(y^0, y^1, \ldots y^e)$, où d est le degré maximum en x et e le degré maximum en y. d et e peuvent être différents. Pour $d = 3, e = 2$, la base canonique est : $(x^0y^0, x^0y^1, x^0y^2, x^1y^0, x^1y^1, x^1y^2, x^2y^0, x^2y^1, x^2y^2, x^3y^0, x^3y^1, x^3y^2)$.

De même la base tensorielle de Bernstein est le produit de la base de Bernstein en x : $(\mathcal{B}_0^{(d)}(x), \mathcal{B}_1^{(d)}(x), \ldots \mathcal{B}_d^{(d)}(x))$ et de la base de Bernstein en y : $(\mathcal{B}_0^{(e)}(y), \mathcal{B}_1^{(e)}(y), \ldots \mathcal{B}_e^{(e)}(y))$. Les bases tensorielles canoniques et de Bernstein ont donc même cardinalité $(d+1) \times (e+1)$ (d'ailleurs, toutes les bases d'un espace vectoriel ont même cardinalité, en dimension finie). La notation $\mathcal{B}_{i,j}^{(d,e)}(x,y) = \mathcal{B}_i^{(d)}(x) \times \mathcal{B}_j^{(e)}(y)$ est utilisée par commodité.

La généralisation à un nombre quelconque de variables est évidente.

La propriété dite de l'enveloppe convexe s'étend au cas multivarié : pour les mêmes raisons que dans le cas univarié, les valeurs d'un polynôme multivarié dans un pavé sont comprises entre le plus petit et le plus grand des coefficients de Bernstein. La borne de cet encadrement est atteinte quand elle se produit en un sommet du pavé, *i.e.,* chacun des indices i_k est soit nul soit maximum, autrement dit égal au degré d_k en la k ième variable.

2.2.4. *Conversion tensorielle pour un polynôme multivarié*

Usuellement, un polynôme bivarié est exprimé comme un ensemble de termes $c_{k,l}x^k y^l$ dans la base canonique. La contribution de chaque terme $c_{k,l}x^k y^l$ au polynôme de Bernstein $\mathcal{B}_{i,j}^{(d,e)}(x,y) = \mathcal{B}_i^{(d)}(x) \times \mathcal{B}_j^{(e)}(y)$ est $c_{k,l}$ fois le produit de la contribution de x^k à $\mathcal{B}_i^{(d)}(x)$ et de la contribution de y^l à $\mathcal{B}_j^{(d)}(y)$. En formule :

$$\mathcal{W}([0,1], c_{k,l}x^k y^l, \mathcal{B}_{i,j}^{(d,e)}(x,y)) = c_{k,l} \times \mathcal{W}([0,1], x^k, \mathcal{B}_i^{(d)}(x)) \times \mathcal{W}([0,1], y^l, \mathcal{B}_j^{(e)}(y))$$

Généralisons au cas multivarié. Pour convertir un polynôme en n variables de la base canonique à la base tensorielle de Bernstein, il faut initialiser à zéro tous les coefficients dans la base tensorielle de Bernstein. Puis, pour chaque terme $c_k x^k = c_{k_1,k_2,\ldots k_n} x_1^{k_1} x_2^{k_2} \ldots x_n^{k_n}$ du polynôme, et pour chaque multi-indice $i = i_1 i_2 \ldots i_n$, avec $0 \leq i_j \leq d_j$, et pour $j = 1, \ldots n$, il faut ajouter la contribution du terme $c_k x^k$ au polynôme de Bernstein $\mathcal{B}_i^{(d)} = \mathcal{B}_{i_1}^{(d_1)}(x_1) \times \mathcal{B}_{i_2}^{(d_2)}(x_2) \ldots \times \mathcal{B}_{i_n}^{(d_n)}(x_n)$.

Généralisons maintenant l'hypercube $[0,1]^n$ au pavé $[u,v]$, $u = (u_1, u_2, \ldots u_n)$ et $v = (v_1, v_2, \ldots v_n)$. Soit $w = (w_i = v_i - u_i)$. La contribution de chaque terme $c_k x^k$ au polynôme de Bernstein $\mathcal{B}_i$ est alors (la formule est correcte pour tout w_i, négatif, positif ou nul) :

$$\mathcal{W}([u,v], c_k x^k, \mathcal{B}_i(x)) =$$

$$c_{k_1,k_2,\ldots k_n} \times \mathcal{W}([u_1,v_1], x_1^{k_1}, \mathcal{B}_{i_1}^{(d_1)}(x_1)) \times \ldots \times \mathcal{W}([u_n,v_n], x_n^{k_n}, \mathcal{B}_{i_n}^{(d_n)}(x_n))$$

2.2.5. *Encadrement d'un polynôme multivarié sur un pavé*

Pour calculer un intervalle englobant les valeurs d'un polynôme multivarié sur un pavé $[u,v]$, où $u = (u_1, u_2, \ldots u_n)$ et $v = (v_1, v_2, \ldots v_n)$, les coefficients du polynôme dans la base tensorielle de Bernstein sont calculés, soit en convertissant de la base canonique à la base tensorielle de Bernstein, soit en utilisant l'algorithme de Paul de Casteljau, quand le pavé est issu de la subdivision d'un pavé plus grand. L'encadrement du polynôme est alors donné par l'intervalle du plus petit coefficient de Bernstein, et du plus grand coefficient de Bernstein du polynôme. Une borne de cet encadrement est atteinte quand elle se produit en un sommet du pavé. Les encadrements calculés ainsi sont bien meilleurs que ceux calculés par l'arithmétique d'intervalles, comme en témoigne la Figure 5.

2.3. *Formulation matricielle*

Ce paragraphe donne une formulation matricielle pour la conversion entre les bases, et pour la méthode de subdivision de de Casteljau.

2.3.1. *Formulation matricielle de la méthode de de Casteljau*

La matrice$\mathcal{P}^{(d)}$, ou $\mathcal{P}$ quand le degré d est sous-entendu, parfois appelée matrice de Pascal, est une matrice carrée d'ordre $d+1$, telle que $\mathcal{P}^{(d)}_{l,c} = \mathcal{C}(l,c)$, en utilisant la convention que les indices de ligne et de colonne commencent par 0. C'est une matrice triangulaire supérieure, qui contient les $d+1$ premières lignes du triangle de Pascal. L'inverse de $\mathcal{P}^{(d)}$ est vraiment remarquable. Par exemple, pour $d=4$, $\mathcal{P}$ et $\mathcal{P}^{-1}$ sont :

$$\mathcal{P}^{(4)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 & 6 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \qquad (\mathcal{P}^{(4)})^{-1} = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & -2 & 3 & -4 \\ 0 & 0 & 1 & -3 & 6 \\ 0 & 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\mathcal{G}^{(d)} = \mathcal{P}^{(d)} \times \text{diag}(1, 1/2, 1/4, \ldots 1/2^d)$ où $\mathcal{G}$ pourrait être appelée la matrice de de Casteljau à gauche. Les colonnes de $\mathcal{G}^{(d)}$ sont les colonnes de $\mathcal{P}^{(d)}$, divisées par des puissances croissantes de 2. $\mathcal{G}^{(d)}$ apparaît dans la méthode de de Casteljau : si $b = (b_0, b_1, \ldots b_d)$ est le vecteur des coefficients de Bernstein d'un polynôme univarié, alors $b\,\mathcal{G}$ est le vecteur des coefficients de Bernstein de la moitié gauche du polynôme. De même, $\mathcal{D}^{(d)}$ ou $\mathcal{D}$ pourrait être appelée la matrice de de Casteljau à droite, et $b\,\mathcal{D}$ est le vecteur des coefficients de Bernstein de la moitié droite du polynôme. $\mathcal{D}^{(d)}$ a les mêmes colonnes que $\mathcal{G}^{(d)}$ mais en ordre inverse. Par exemple, pour le degré $d=4$:

$$\mathcal{G}^{(4)} = \begin{pmatrix} 1 & 1/2 & 1/4 & 1/8 & 1/16 \\ 0 & 1/2 & 2/4 & 3/8 & 4/16 \\ 0 & 0 & 1/4 & 3/8 & 6/16 \\ 0 & 0 & 0 & 1/8 & 4/16 \\ 0 & 0 & 0 & 0 & 1/16 \end{pmatrix}$$

Si, dans la méthode de de Casteljau, les coefficients sont $1-t, t$ au lieu de $1/2, 1/2$ (autrement dit si le domaine $[0,1]$ est divisé en $[0,t]$ et $[t,1-t]$), alors la matrice de de Casteljau à gauche devient : $\mathcal{G}^{(d)} = \text{diag}(1, \beta, \beta^2, \ldots \beta^d)\mathcal{P}^{(d)}\text{diag}(1, \alpha, \alpha^2, \ldots \alpha^d)$ où $\alpha = 1-t, \beta = t/\alpha$. De nouveau, la matrice droite $\mathcal{D}^{(d)}$ a les mêmes colonnes que $\mathcal{G}^{(d)}$ mais en ordre inverse.

2.3.2. *Conversion matricielle dans le cas univarié*

Ce paragraphe donne une méthode matricielle pour le changement de base. La conversion de la base canonique $X = (x^0, x^1, \ldots x^d)$ à la base de Bernstein $\mathcal{B} =$

$(\mathcal{B}_0(x), \mathcal{B}_1(x), \ldots \mathcal{B}_d(x))$ est une transformation linéaire représentable par une matrice M carrée d'ordre $d+1$. Par commodité, nous supposons que les indices de ligne et de colonne de M et M^{-1} commencent en 0. Alors :

$$X = \mathcal{B}M, \quad M_{i,k} = \mathcal{W}([0,1], x^k, \mathcal{B}_i^{(d)}) = \mathcal{C}(k,i)/\mathcal{C}(k,d)$$

Inversement :

$$XM^{-1} = \mathcal{B}, \quad (M^{-1})_{l,c} = \mathcal{C}(c,d) \times \mathcal{C}(l-c, d-c) \times (-1)^{l+c}$$

Le polynôme $f(x) = \sum_{k=0}^{d} c_k x^k$ est représenté dans la base canonique par le vecteur $c = (c_0, c_1, \ldots c_d)$, ou encore : $f(x) = Xc^t$. Sa représentation dans la base de Bernstein est $f(x) = \sum_{i=0}^{d} b_i \mathcal{B}_i^{(d)}(x)$, et est stockée dans le vecteur $b = (b_0, b_1, \ldots b_d)$. De plus $X = \mathcal{B}M$ où M est la matrice définie dans le paragraphe précédent. De $f(x) = Xc^t = \mathcal{B}b^t$ et $X = \mathcal{B}M$, on déduit $b^t = Mc^t$.

Par exemple, pour le degré $d = 4$, les matrices M et M^{-1} sont :

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1/2 & 0 & 0 & 0 \\ 1 & 1/2 & 1/6 & 0 & 0 \\ 1 & 3/4 & 1/2 & 1/4 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \qquad M^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 12 & -12 & 4 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix}$$

Le lecteur intrigué par les ressemblances entre M et $\mathcal{P}$ pourra vérifier que $M^{-1} = \Theta \times \mathrm{diag}_{k=0}^{d}(\mathcal{C}(k,d)) \times \mathcal{P}^t \times \Theta$ où Θ est la matrice diagonale d'ordre $d+1$: $\Theta = \mathrm{diag}(-1, 1, -1, 1, \ldots)$.

La formulation matricielle met bien en évidence certaines propriétés combinatoires. Aussi, elle est parfois commode, par exemple pour étudier le conditionnement des matrices de conversion entre les deux bases. Classiquement, il résulte d'une telle étude que les flottants en simple précision (sur 32 bits) deviennent insuffisamment précis pour des degrés plus grands ou égaux à 10, et les flottants en double précision (sur 64 bits) pour des degrés plus grands ou égaux à 20.

2.3.3. *Conversion matricielle dans le cas bivarié*

Posons $X = (x^0, x^1, \ldots x^d)$, $Y = (y^0, y^1, \ldots y^e)$. Le polynôme dans la base canonique est $f(x,y) = XCY^t$, avec C une matrice ayant $d+1$ lignes et $e+1$ colonnes. $C_{r,c}$ est le coefficient de $x^r y^c$. En utilisant $X = \mathcal{B}_x M_x$ et $Y = \mathcal{B}_y M_y$ avec les conventions naturelles :

$$f(x,y) = XCY^t = \mathcal{B}_x M_x C (\mathcal{B}_y M_y)^t = \mathcal{B}_x (M_x C M_y^t) \mathcal{B}_y^t$$

donc les coefficients de $f(x,y)$ dans la base tensorielle de Bernstein sont donnés par la matrice $T = M_x C M_y^t$, donc $f(x,y) = \mathcal{B}_x T \mathcal{B}_y^t$.

La formulation matricielle précédente est souvent utilisée en CFAO et en informatique graphique pour les surfaces de Bézier : elles sont décrites par $(x = X(u,v), y =$

$Y(u,v), z = Z(u,v))$, où $(u,v) \in [0,1]^2$; les polynômes X, Y, Z sont donnés dans la base de Bernstein, par un tableau à deux entrées p_{ij} de points 3D : (x_{ij}, y_{ij}, z_{ij}). Malheureusement, ce formalisme matriciel s'étend mal au cas trivarié (les fonctions paramétriques volumiques, parfois nommées "volumes de Bézier") et au delà : les matrices sont des tableaux à 2 dimensions. Il vaut donc mieux utiliser la méthode de conversion tensorielle en §2.2.2.

2.4. *Algorithmes utilisant les bases tensorielles de Bernstein*

2.4.1. *Calcul de couvertures de courbes implicites*

Les bases tensorielles de Bernstein sont souvent utilisées pour calculer des couvertures des courbes ou surfaces algébriques implicites données par leur équation : $f(x,y) = 0$, ou $f(x,y,z) = 0$ (Martin *et al.*, 2002). Une couverture d'une courbe ou surface est un ensemble discret de pixels, de voxels, etc, qui contient la courbe ou surface en question. Une bonne couverture doit être la moins épaisse possible : idéalement, tous ses pixels, voxels, etc ont une intersection non vide avec la courbe ou la surface couverte.

La Figure 5 montre des couvertures de courbes implicites polynomiales $f(x,y) = 0$ calculées avec une méthode de subdivision récursive du carré initial : un intervalle encadrant $f(x,y)$ pour $(x,y) \in P$ est calculé ; si l'intervalle ne contient pas 0, alors le pavé P ne peut contenir de points de la courbe d'équation $f(x,y) = 0$; sinon, si le pavé est suffisamment petit, il est inséré dans la couverture de la courbe, ou affiché ; sinon, le pavé est divisé en 4 pavés de côté 2 fois plus petit, qui sont étudiés à leur tour, récursivement, ou en utilisant une pile ou une autre structure d'attente.

L'intervalle encadrant les valeurs de f dans un pavé donné P peut être calculé par une arithmétique d'intervalles, naïve ou centrée. C'est ce qui est fait dans la rangée du haut de la Figure 5. L'encadrement peut aussi être calculé en utilisant les bases tensorielles de Bernstein, comme en §2.2.5. Pour cela, le polynôme est converti dans la base de Bernstein ; l'encadrement est donné par le coefficient de Bernstein le plus petit, et par le plus grand. Au besoin, la méthode de de Casteljau est utilisée pour calculer les coefficients de Bernstein de f dans les quatre pavés les plus petits ; un premier appel de la méthode de Paul de Casteljau calcule les coefficients de la moitié gauche et de la moitié droite ; puis, pour la moitié gauche, un second appel calcule les coefficients du quart inférieur gauche à partir des coefficients de la moitié gauche ; un troisième appel calcule les coefficients du quart inférieur droit à partir des coefficients de la moitié droite. La Figure 5 montre les couvertures obtenues par cette méthode dans la rangée inférieure. Clairement, la méthode utilisant les bases de Bernstein écarte beaucoup plus rapidement les pavés non traversés par la courbe d'équation $f(x,y) = 0$. Cette méthode est aussi utilisée en 3D pour calculer des couvertures de surfaces algébriques données par leur équation implicite : $f(x,y,z) = 0$, ou des couvertures de courbes 3D décrites par un système $f(x,y,z) = g(x,y,z) = 0$.

Cette méthode de calcul de couverture de courbe continue en 2D a été généralisée à des courbes fractales définies par des IFS (*Iterated Functions Systems*, systèmes de fonctions itérées), et plus récemment à des attracteurs étranges (Michelucci *et al.*, 2006a; Paiva *et al.*, 2006). Ceci suggère que les solveurs présentés dans cet article peuvent être généralisés à des systèmes d'équations représentant des objets non lisses.

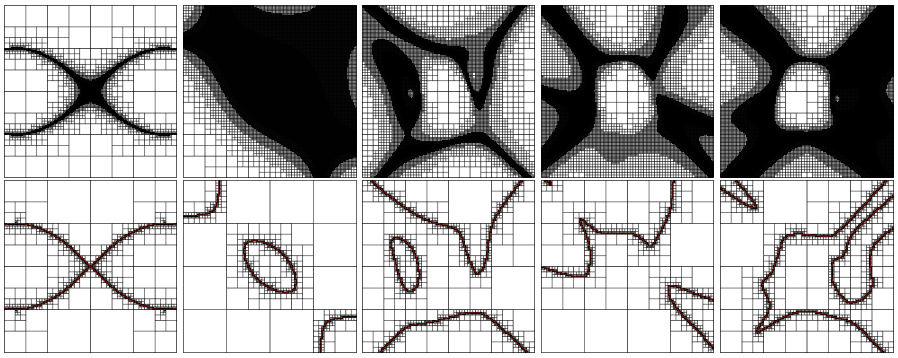

Figure 5. *En haut, les encadrements des polynômes sont calculés avec l'arithmétique d'intervalles naïve ; en bas, ils sont calculés en utilisant les bases de Bernstein. De gauche à droite, un ovale de Cassini* $C_{2,2}(x,y) = 0$ *dans* $[-2,2] \times [-2,2]$ *où* $C_{a,b}(x,y) = ((x+a)^2+y^2) \times ((x-a)^2+y^2) - b^4$*, une courbe due à Martin et al* $f(x,y) = 15/4+8x-16x^2+8y-112xy+128x^2y-16y^2+128xy^2-128x^2y^2 = 0$ *dans le carré* $(x,y) \in [0,1]^2$*, et trois courbes algébriques aléatoires de degré total 10, 14, 18.*

2.4.2. *Isolation de racines réelles de polynômes univariés*

Les bases de Bernstein permettent à un algorithme simple et rapide d'isoler les racines réelles d'un polynôme univarié dans l'intervalle $[0,1]$. Pour cela, il faut d'abord convertir le polynôme dans la base de Bernstein, et en déduire un encadrement du polynôme. Si l'encadrement ne contient pas 0 (tous les coefficients de Bernstein ont le même signe), alors le polynôme n'a pas de racine. Sinon l'encadrement contient 0 et l'intervalle étudié est susceptible de contenir une ou des racines. Si les coefficients de Bernstein augmentent (diminuent) de façon monotone, l'intervalle contient une seule racine simple, et la méthode standard de Newton, ou la méthode de la sécante, est garantie converger vers la racine. Sinon les deux moitiés de l'intervalle en x sont étudiées récursivement, en utilisant la méthode de de Casteljau pour calculer les coefficients de Bernstein des moitiés gauche et droite. Cependant, la récursion doit être arrêtée quand l'intervalle étudié en x est suffisamment petit, pour éviter de boucler indéfiniment en présence d'une racine singulière, car dans ce cas, les coefficients de Bernstein ne deviennent jamais monotones croissants ou décroissants.

Eventuellement, l'intersection de l'enveloppe convexe des points de contrôle $(i/d, b_i)$ avec l'axe des x peut être calculée : ce segment permet de contracter l'intervalle en x étudié sans perdre ses racines éventuelles ; mais, empiriquement, cette variante n'est pas intéressante, car la subdivision et la précision des encadrements due à la base de Bernstein éliminent rapidement les intervalles en x sans racine. Nous verrons en §2.4.4 que cette optimisation est par contre intéressante dans le cas multivarié.

Pour trouver les racines de $f(x) = 0$ dans $[1, +\infty[$, on définit $g(x) = x^d f(1/x)$: si $f(x) = \sum f_i x^i$ dans la base canonique, alors $g(x) = \sum f_i x^{d-i}$, c'est-à-dire que g est un polynôme avec les mêmes coefficients que f mais dans le sens inverse. Les racines de g dans $[0, 1]$ sont à l'évidence les inverses des racines de f dans $[1, +\infty[$. De cette façon, toutes les racines positives sont calculées. Pour trouver les racines négatives de $f(x) = 0$, on calcule les racines positives du polynôme $h(x) = f(-x)$: ce sont les opposées des racines négatives de f.

Ce type d'algorithme est utilisé en infographie pour le tracer de rayons sur les surfaces algébriques implicites $f(x, y, z) = 0$ avec un degré total d, c'est-à-dire, pour calculer les points d'intersections entre un rayon (une demi-ligne) et la surface. Le rayon est paramétré avec : $x = x_0 + at,\ y = y_0 + bt,\ z_0 + z = ct$, où l'origine (x_0, y_0, z_0) du rayon est connue, ainsi que sa direction (a, b, c). Remplacer x, y, z par leurs valeurs en t donne une équation algébrique univariée en t, de degré d, qui est résolue par une variante de la méthode précédente.

2.4.3. *Calcul de l'enveloppe convexe 2D*

Le calcul de l'enveloppe convexe d'un ensemble de points 2D est un problème fondamental de l'infographie (Cormen *et al.*, 2001; Preparata *et al.*, 1977). Soit (x_i, y_i) un ensemble de points dans le plan x, y. Nous expliquons seulement comment calculer la partie inférieure de l'enveloppe convexe des points (x_i, y_i) : la partie supérieure est calculée de la même manière. Premièrement, les points sont triés par x croissant. Dans notre application, c'est déjà fait puisque toutes les valeurs de x sont i/d, où d est le degré, et i un entier dans $0, ...d$. Nous supposons de plus qu'il n'y a qu'un seul point pour chaque valeur x_i, celui d'ordonnée y minimale. L'enveloppe convexe inférieure est initialisée avec les deux points de gauche $p_0 = (x_0, y_0)$; $p_1 = (x_1, y_1)$.

Ensuite, les points (x_k, y_k) sont considérés de gauche à droite (c'est-à-dire, par x_k croissant), pour $k = 2, \ldots d$, et l'enveloppe convexe est mise à jour comme suit. Soit b le point le plus à droite de l'enveloppe convexe courante, et a le point juste avant dans l'enveloppe ; tant que abp_k "tourne à droite" (l'angle abp_k est concave), le point b est retiré de l'enveloppe. A la fin de cette boucle, le point p_k est inséré en fin de l'enveloppe convexe inférieure. Trois points p, q, r "tournent à droite" lorsque le déterminant de $((p_x, p_y, 1), (q_x, q_y, 1), (r_x, r_y, 1))^t$ est négatif. Quand il s'annule, les points p, q, r sont alignés ; quand il est positif, ils tournent à gauche.

Cet algorithme est en $O(d \log d)$ si d est le nombre de points, le facteur $d \log d$ est dû à la phase de tri ; comme elle est ici inutile, la méthode est linéaire en d, le nombre de points.

2.4.4. *Préconditionnement et réduction*

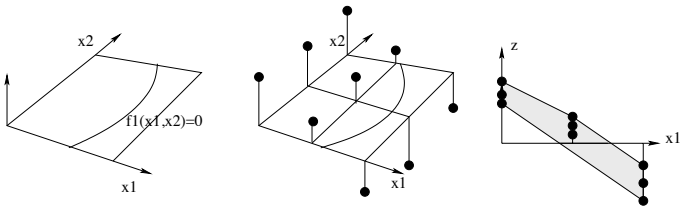

Figure 6. *L'équation $z = f_1(x_1, x_2) = 0$ est de degré 2 en x_1 et en x_2 et a une grille de points de contrôle de 3×3. La courbe $f_1(x_1, x_2) = 0$ est considérée comme l'intersection du plan $z = 0$ et de la surface $z = f_1(x_1, x_2)$. La surface est à l'intérieur de l'enveloppe convexe de ses points de contrôle $(i/2, j/2, b_{i,j}), i = 0, 1, 2, j = 0, 1, 2$. Il est facile de calculer l'enveloppe convexe 2D des projections sur le plan x_1, z des points de contrôle. L'intersection de ce polygone convexe et de l'axe x_1 est un segment qui englobe tous les x_1 des points de la courbe.*

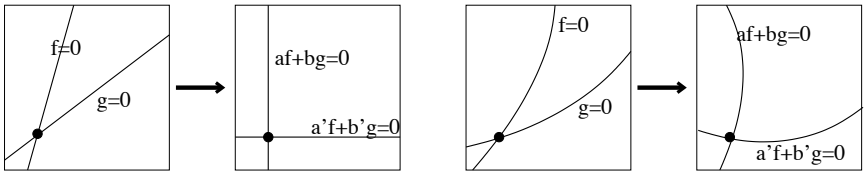

Figure 7. *Effet du préconditionnement sur un système linéaire (à gauche), sur un système non linéaire à droite.*

La réduction d'un pavé, aussi appelée contraction, fournit un pavé plus petit, inclus dans le pavé étudié, et qui est garanti contenir toutes les racines. Une réduction efficace est intéressante car elle évite l'explosion combinatoire due aux subdivisions. Les subdivisions sont cependant indispensables pour séparer des racines distinctes. Les bases tensorielles de Bernstein permettent des réductions très efficaces.

Le système $f(x) = 0$ est d'abord préconditionné, pour que le jacobien du système soit la matrice identité au centre du pavé étudié. La Figure 7 montre un exemple 2D de préconditionnement. Le système préconditionné est $g(x) = Mf(c)$, avec $M = (f'(c))^{-1}, c = (u+v)/2$ le centre du pavé $[u, v]$ étudié. g et f ont les mêmes racines, et le jacobien de g en c est $g'(c) = (f'(c))^{-1} f'(c)$ est bien l'identité. En fait, une inverse approximative est suffisante. Au passage, il est possible d'éviter le calcul explicite de l'inverse, en utilisant une décomposition LU.

Le préconditionnement ne peut être appliqué facilement que si tous les f_i ont les mêmes degrés d_1 en x_1, d_2 en x_2, etc ; les points de contrôle de $g_k = \sum_i \lambda_i f_i$ sont alors les λ_i combinaisons linéaires des points de contrôle des f_i. Un prétraitement global du système d'équations doit élever le degré des polynômes de degré inférieur.

L'élévation de degré est triviale dans la base canonique (il suffit de remplacer $x+2$ par $0x^2 + x + 2$, par exemple), et une simple formalité dans les bases de Bernstein (Farin, 1988), mais elle n'est pas anodine pour les grands systèmes (de plus de 6 ou 7 équations) : en effet, la représentation dans la base tensorielle de Bernstein a un coût exponentiel. Nous reviendrons ultérieurement sur les conséquences.

Après le préconditionnement, et pour un pavé suffisamment petit, la k ième hypersurface d'équation $g_k(x) = 0$ est géométriquement proche d'un hyperplan où x_k est à peu près constant, autrement dit la coordonnée x_k des points de cette hypersurface dans le pavé étudié est comprise dans un intervalle $[u'_k, v'_k]$ de largeur plus petite que celle du pavé initial : $w_k = v_k - u_k$. C'est cet intervalle réduit $[u'_k, v'_k]$ qu'il nous faut calculer, comme suit.

Dans l'espace $(x \in \mathbb{R}^n, z \in \mathbb{R})$, chaque hypersurface $z = g_k(x)$ est incluse dans l'enveloppe convexe de ses points de contrôle : voir Figure 6 pour un exemple où x est dans un pavé 2D. Le calcul de cette enveloppe convexe est trivial en dimension 1, facile en dimension 2, et difficile ou impossible en grande dimension. Heureusement, l'enveloppe convexe appartient à un prisme, dont la base est la projection sur le plan (x_k, z) de ses points de contrôle. Il suffit donc de calculer ce polygone convexe avec la méthode en §2.4.3. L'intersection de ce polygone avec l'axe x_k (ou plutôt le segment $[u_k, v_k]$ sur l'axe x_k) donne l'intervalle réduit $[u'_k, v'_k]$. Si cette intersection est vide, alors le pavé ne contient pas de racine.

Mourrain et Pavone (Mourrain *et al.*, August 2005) utilisent une variante de cette méthode ; au lieu de calculer l'enveloppe convexe inférieure des points $(x_k, \min z_k)$, et l'enveloppe convexe supérieure des $(x_k, \max z_k)$, ils considèrent ces points comme les points de contrôle du graphe d'un polynôme dont ils calculent la plus petite et la plus grande des racines dans $[u_k, v_k]$, avec une méthode similaire à celle de §2.4.3. Ils procèdent de même pour les points $(x_k, \max z_k)$. Ils en déduisent un encadrement $[u'_u, v'_k]$ plus précis.

Quelle que soit la méthode utilisée pour réduire selon la dimension x_k, chaque dimension du pavé est réduite tour à tour, pour $k = 1, \ldots n$. Une variante possible prend en compte les réductions précédentes selon les dimensions $1, \ldots k-1$ lorsque la réduction selon l'axe x_k est effectuée.

Dans le cas où le système est linéaire, le pavé réduit est soit l'ensemble vide (la solution est hors du pavé), soit le point de la racine en supposant qu'il n'y a pas d'imprécision numérique. En pratique, un pavé réduit, de quelques ULP (*Unit in the Last Place*) de côté, et contenant le point solution, est obtenu.

Les réductions du pavé étudié sont itérées tant qu'elles réduisent significativement les pavés, par exemple tant que la longueur de chacun des n côtés est au moins divisé par 2 ou davantage.

Si après toutes ces réductions, le pavé est suffisamment petit, il est ajouté à une liste de solutions ; éventuellement, un test d'unicité sur le pavé est effectué, et le pavé est éventuellement étiqueté comme contenant une racine unique ; plusieurs solveurs

permettent de "polir la racine" ou de préciser encore le pavé solution avec quelques itérations de Newton. Il se peut que le test d'unicité échoue, même pour un petit pavé : il contient une racine multiple, ou plusieurs racines simples proches, ou une quasi racine (par exemple, 0 est une quasi racine de $x^2 + 10^{-12} = 0$), et la précision finie de l'arithmétique flottante ne permet pas de décider.

Si le pavé reste gros et ne peut plus être significativement réduit, il contient vraisemblablement plusieurs racines. Le pavé doit être coupé en deux, par exemple selon son côté le plus long, ou bien selon le côté qui a été le moins réduit (en valeur relative) lors de la dernière réduction effectuée. Les deux pavés sont empilés. Il est aussi possible de couper en deux selon plusieurs dimensions, 2, 3, ou 4, ce qui donne $2^2, 2^3, 2^4$ sous pavés.

2.5. *Quelques tests utiles*

2.5.1. *Preuve qu'un pavé ne contient pas de racine*

Parfois, la méthode de réduction ne peut pas détecter rapidement qu'un pavé ne contient pas de racine. Par exemple, en 2D, pour deux courbes "parallèles" (*offset*) et proches, il faut subdiviser le long des deux courbes jusqu'à les séparer, ce qui est coûteux. Le test suivant détecte très vite ce genre de situation.

Soit $f_1(x) = f_2(x) = ... = f_n(x) = 0$ un système d'équations. Nous supposons que tous les f_i ont les mêmes degrés pour toutes les variables. S'il existe des nombres $\lambda_i \in \mathbb{R}^n$ tels que $g(x) = \sum_i \lambda_i f_i(x)$ a tous ses coefficients de Bernstein strictement positifs, par exemple supérieurs ou égaux à 1, alors $g(x)$ est toujours positif dans le pavé étudié ; or g s'annule pour toutes les racines communes des f_i. Ceci prouve donc que les f_i n'ont aucune racine commune dans le pavé. Les coefficients de Bernstein de g sont les λ_i combinaisons linéaires des coefficients de Bernstein des f_i, si bien que les λ_i existent ssi le problème correspondant de programmation linéaire a une solution.

Cette idée simple s'étend aux systèmes d'équations et aux inéquations. Une illustration en est donnée Figure 8. Supposons que le problème est de trouver x dans un pavé de $\mathbb{R}^n$ tel que $f_1(x) = f_2(x) = \ldots = f_n(x) = 0$ et que $g_1(x) \leq 0, \ldots g_m(x) \leq 0$. S'il existe des $\lambda_i \in \mathbb{R}^n$ et des $\mu_j \geq 0$ tels que $h = \sum_i \lambda_i f_i + \sum_j \mu_j g_j \geq 1$ (le dernier 1 peut être remplacé par n'importe quelle constante positive) pour tous les points x du pavé, alors le système n'a pas de solution. En effet, considérons x^* une racine commune des f_i. Alors comme $h(x^*) \geq 1$, il vient $h(x^*) = \sum_i \lambda_i f_i(x^*) + \sum_j \mu_j g_j(x^*) = \sum_j \mu_j g_j(x^*) \geq 1 > 0$; mais ceci contredit les contraintes $g_j(x) \leq 0$. Trouver si les λ_i et les μ_j existent est un problème de programmation linéaire.

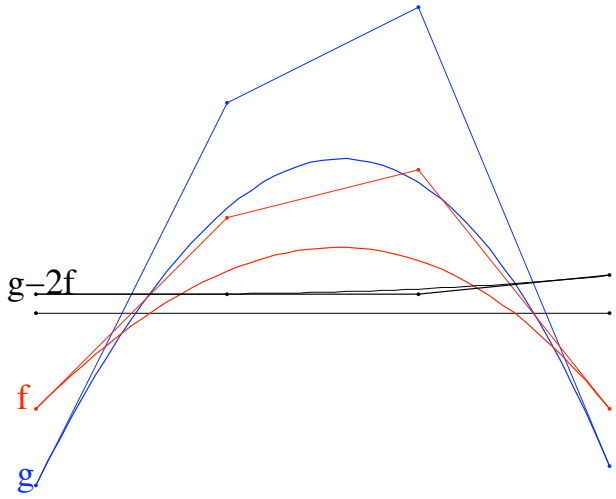

Figure 8. *Le problème est de trouver x tel que $f(x) = 0$ et $g(x) \leq 0$. Dans cet exemple, les coefficients de Bernstein de $g - 2f$ sont tous strictement positifs. Donc $g-2f$ est strictement positif sur l'intervalle considéré. $g(x^*)-2f(x^*) > 0$ et $f(x^*) = 0 \Rightarrow g(x^*) > 0$: donc il est prouvé que le système $f(x) = 0$ et $g(x) \leq 0$ n'a pas de solution dans l'intervalle.*

2.5.2. *Preuve d'existence d'une racine*

Le théorème de Miranda, aussi appelé théorème de Poincaré-Miranda fournit un test d'existence d'une racine dans un pavé. Il généralise le théorème de Rolle, dit des valeurs intermédiaires au cas multi-dimensionnel. Soient f_i, $i = 1, \ldots n$ fonctions continues de $\mathbb{R}^n$ vers $\mathbb{R}$; soit un pavé $[u, v]$ de $\mathbb{R}^n$, $u = (u_i), v = (v_i)$. Si pour tout k dans $1, \ldots n$, la fonction f_k est de signe constant sur la face $x_k = u_k$, et de signe constant mais opposé sur la face opposée du pavé $x_k = v_k$, alors il existe au moins une racine commune aux f_i dans le pavé. Une fonction f_k est de signe constant sur la face $x_k = u_k$ si l'encadrement de f_k sur le sous pavé correspondant (l'intervalle pour x_k est $[u_k, u_k]$ au lieu de $[u_k, v_k]$) ne contient pas 0. On remarquera que, pour un pavé suffisamment petit contenant une racine r commune aux f_i, le préconditionnement produit des hypersurfaces d'équation $g_k(x) = 0$ géométriquement très proches de l'hyperplan $x_k - r_k = 0$, et les conditions du théorème sont donc probablement vérifiées pour le système préconditionné.

2.5.3. *Preuve d'unicité d'une racine dans un pavé*

Elbert et Kim (Elber *et al.*, 2001) ont proposé un test pour décider si un pavé contient une seule racine simple (une racine est simple ssi le jacobien est de rang n). Ce test est bien adapté aux bases géométriques, comme les bases tensorielles des splines ou de Bernstein. Soit T_i le cône des vecteurs tangents aux hypersurfaces

$z = f_i(x)$ dans le pavé étudié. Elbert et Kim encadrent ces cônes avec un vecteur axial (la moyenne des vecteurs générateurs) et un intervalle angulaire. Quand le vecteur nul est le seul vecteur commun à tous les T_i, alors le pavé étudié contient au plus une racine commune aux f_i. Après le préconditionnement, quand le pavé considéré contient une seule racine simple r, cette condition est probablement vérifiée, car le préconditionnement tend à créer des hypersurfaces $g_k(x) = 0$ géométriquement proches des hyperplans d'équation $x_k - r_k = 0$ (Figure 7).

Elbert et Kim stoppent la subdivision dès que l'existence et l'unicité sont prouvées, et recourent alors à quelques itérations de Newton pour préciser la racine, en partant du centre du pavé. Cette optimisation peut accélérer quelque peu le solveur, mais la complexité théorique du solveur reste inchangée.

Un autre théorème donnant un test d'unicité est le théorème de Newton-Kantorovitch. Il est utilisé dans le solveur d'ALIAS (COPRIN, 2004). Comme ce théorème utilise des encadrements des dérivées secondes des équations, il est très commode pour les systèmes quadratiques où toutes les dérivées secondes sont constantes.

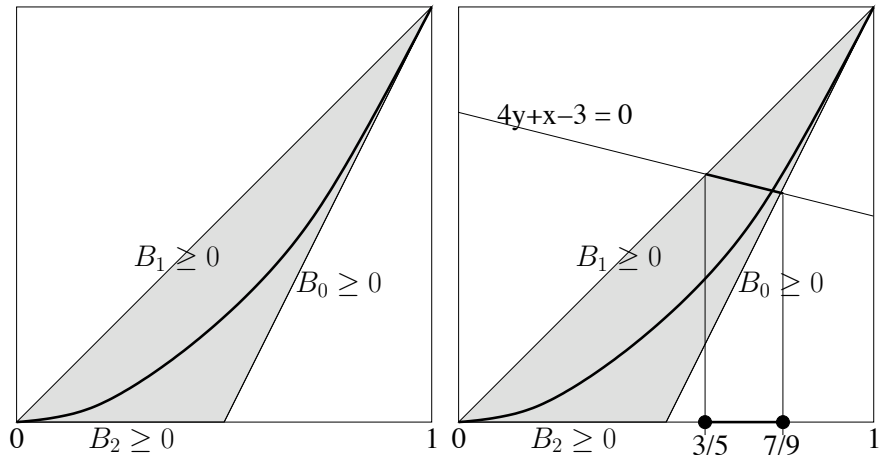

Figure 9. *À gauche : le polytope de Bernstein encadrant la courbe* $(x, y = x^2)$*, pour* $(x, y) \in [0, 1]^2$*. C'est l'intersection des demi-plans d'équations :* $\mathcal{B}_0(x) = (1-x)^2 = y - 2x + 1 \geq 0$*,* $\mathcal{B}_1(x) = 2x(1-x) = 2x - 2y \geq 0$*,* $\mathcal{B}_2(x) = x^2 = y \geq 0$*. À droite : résoudre* $4x^2 + x - 3 = 0$*, avec* $x \in [0, 1]$*, équivaut à calculer l'intersection de la droite* $4y + x - 3 = 0$ *avec la courbe* (x, x^2)*. La programmation linéaire donne l'intersection entre la droite et le polytope de Bernstein : l'intervalle en* x*, initialement* $[0, 1]$*, est réduit à* $[3/5, 7/9]$*.*

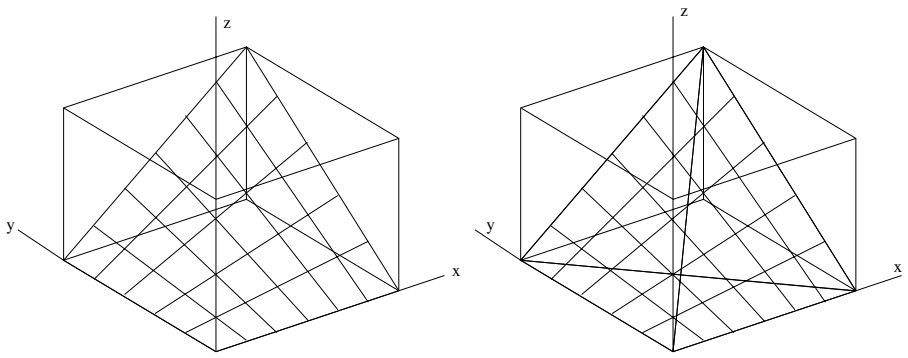

Figure 10. *Le polytope de Bernstein encadrant le carreau de surface, un PH :* $(x, y, z = xy)$*. Les inéquations des plans des 4 faces sont :* $\mathcal{B}_i(x)\mathcal{B}_j(y) \geq 0$*, où* $i = 0, 1$ *et* $\mathcal{B}_0(t) = 1 - t$ *et* $\mathcal{B}_1(t) = t$*.*

2.6. *Les limitations de ce premier type de solveur*

Ce type de solveur est très satisfaisant pour les systèmes assez petits qui sont rencontrés en infographie : le calcul des points d'intersection d'une droite avec une surface algébrique implicite d'équation $f(x, y, z) = 0$, ou avec une surface paramétrique polynomiale d'équations : $(x = X(u, v), y = Y(u, v), Z = Z(u, v))$ ou une surface paramétrique rationnelle d'équations : $H(u, v)\times x = X(u, v), H(u, v)\times y = Y(u, v)$, $H(u, v) \times z = Z(u, v)$, où $(u, v) \in [0, 1]^2$, et X, Y, Z, H sont des polynômes, souvent déjà donnés dans la base de Bernstein, ou des fonctions polynomiales par morceaux souvent données dans une base de splines, comme dans (Elber *et al.*, 2001). A raison de 24 ou 25 images par secondes, de un million ou plus de pixels pour chaque image, et de quelques dizaines de telles surfaces dans la scène, le calcul de film d'animation par la méthode du lancer de rayons permet de tester de façon intensive ce type de solveurs. Une autre application, moins intensive, est le calcul des points d'intersection entre 3 des surfaces précédentes.

Ce type de solveur se heurte à deux difficultés. La première difficulté, due à l'imprécision numérique, est abordée en §2.6.1 et facilement surmontée. La seconde est plus délicate, et due au coût de la représentation dans la base tensorielle de Bernstein, et est abordée en §2.6.2.

2.6.1. *Imprécision numérique*

Les difficultés dues à l'imprécision numérique ne sont pas propres à ce solveur.

Une première solution serait d'ignorer le problème, et donc effectuer les calculs en arithmétique flottante. Cette approche ne peut être justifiée que pour des applications non critiques, comme la synthèse d'images, où l'imprécision aura pour principal effet, imperceptible, de déplacer d'un pixel les contours apparents des objets. Mentionnons

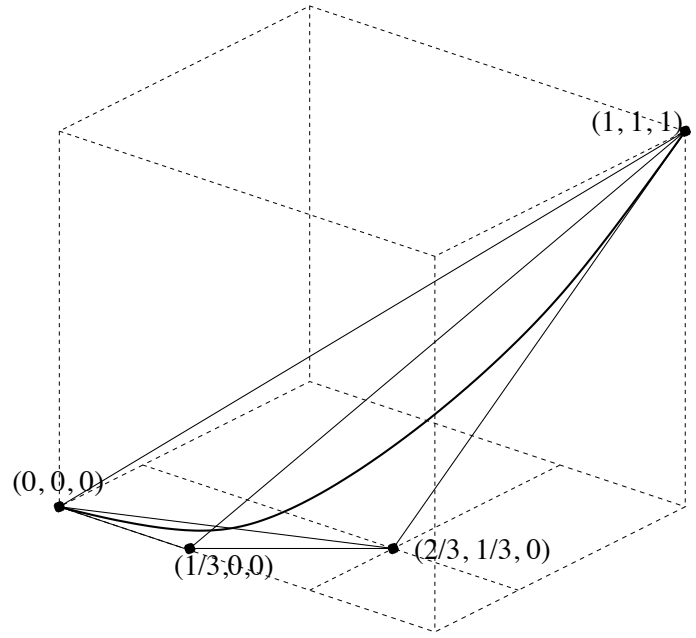

Figure 11. *Le polytope de Bernstein, un tétraèdre, encadrant la courbe* $(x, y = x^2, z = x^3)$ *où* $x \in [0, 1]$. *Ses sommets sont :* $v_0 = (0, 0, 0)$, $v_1 = (1/3, 0, 0)$, $v_2 = (2/3, 1/3, 0)$ *et* $v_3 = (1, 1, 1)$. v_0 *est sur le plan* $\mathcal{B}_1 = \mathcal{B}_2 = \mathcal{B}_3 = 0$, v_1 *sur* $\mathcal{B}_0 = \mathcal{B}_2 = \mathcal{B}_3 = 0$, *etc. Le tétraèdre est l'intersection des demi-espaces :* $\mathcal{B}_0(x) = (1-x)^3 = 1 - 3x + 3x^2 - x^3 \geq 0 \Rightarrow 1 - 3x + 3y - z \geq 0$, $\mathcal{B}_1(x) = 3x(1-x)^2 = 3x - 6x^2 + 3x^3 \geq 0 \Rightarrow 3x - 6y + 3z \geq 0$, $\mathcal{B}_2(x) = 3x^2(1-x) = 3x^2 - 3x^3 \geq 0 \Rightarrow 3y - 3z \geq 0$, $\mathcal{B}_3(x) = x^3 \geq 0 \Rightarrow 3z \geq 0$.

cependant que quelques précautions non complètement triviales sont nécessaires pour ne pas manquer la racine dans le cas de systèmes linéaires...

Une seconde solution représente les coefficients de Bernstein par des intervalles ; c'est ce que font Hu et Patrikalakis (Hu *et al.*, 1996) par exemple. La largeur de ces intervalles est initialement de un ou deux ULP, et croît, relativement lentement, au fur et à mesure des subdivisions de Casteljau. Il vient un moment où subdiviser devient inutile : ainsi, en supposant que les bornes des intervalles sont des entiers (les nombres flottants formant eux aussi un ensemble discret), la moitié gauche et la moitié droite de l'intervalle $[0, 1]$ sont toutes deux $[0, 1]$; plus généralement, la largeur des deux moitiés gauche et droite d'un intervalle de largeur 1 est 1. Cette seconde approche peut poser quelques difficultés techniques, lorsqu'est utilisée la programmation linéaire, comme en §2.5.1. La mise à l'échelle présentée dans le paragraphe §3.4.2 peut traiter ce problème.

2.6.2. *Coût exponentiel de la représentation*

Ces solveurs ont une limitation plus fondamentale, qui devient critique quand le nombre n d'inconnues et d'équations augmente, et devient supérieur à 6 ou 7 en pratique. Ces solveurs calculent tous les coefficients de Bernstein des polynômes, pour trouver le plus petit coefficient, et le plus grand. Or, si les polynômes sont creux dans la base tensorielle canonique, ils cessent de l'être dans la base tensorielle de Bernstein. Par exemple, le monôme 1 de la base tensorielle canonique est dense dans la base tensorielle de Bernstein ; il s'écrit :

$$1 = (\mathcal{B}_0^{(d_1)}(x_1) + \ldots \mathcal{B}_{d_1}^{(d_1)}(x_1)) \times \ldots \times (\mathcal{B}_0^{(d_n)}(x_n) + \ldots \mathcal{B}_{d_n}^{(d_n)}(x_n))$$

dans la base tensorielle de Bernstein. Pour un système linéaire en n inconnues, il nécessite 2^n coefficients, tous égaux à 1. Pour un système quadratique (chaque variable apparaît avec un degré au plus 2), il nécessite 3^n coefficients ; si en chaque variable apparaît avec un degré au plus d, il nécessite $(d+1)^n$ coefficients, tous égaux à 1. Ainsi le plus simple des monômes : 1, a une représentation dans la base tensorielle de Bernstein qui a une taille exponentielle. En conséquence, le calcul de l'encadrement des valeurs d'un polynôme dans un pavé, et la méthode de réduction, nécessitent un temps exponentiel, si bien que ces solveurs sont inutilisables pour les grands systèmes.

Comment lever cette limitation ? Il se trouve que, parmi une quantité exponentielle de coefficients de Bernstein, seuls le plus petit et le plus grand sont utiles : ils donnent les bornes des encadrements. Ne pourrait-on calculer uniquement ces deux coefficients ? Le nouveau solveur présenté en §3 utilise cette remarque de bon sens, en recourant à la programmation linéaire (ce que font déjà certains des solveurs du premier type, avec le test proposé en §2.5.1).

3. Solveur et polytope de Bernstein

Cette partie présente un nouveau type de solveur, qui utilise de la programmation linéaire sur un polytope. Il est défini en §3.1. Pour simplifier, cet article considère des systèmes quadratiques : tous les monômes des polynômes du système sont de degré total au plus 2. Ceci est sans perte de généralité. D'une part, beaucoup de systèmes en modélisation géométrique par contraintes se réduisent à la résolution de systèmes quadratiques ; d'autre part, plusieurs méthodes permettent de réduire tout système algébrique à un système quadratique ; c'est du reste ce qui est fait dans l'article (Fünfzig *et al.*, 2009) sur des monômes de degré 4. Enfin, il est aussi possible de généraliser le polytope de Bernstein à des degrés supérieurs ; si d est le degré total maximum des monômes, le polytope a $O(n^d)$ hyperplans et hyperfaces ; le nombre de sommets du polytope, par contre, est toujours exponentiel, plus grand que 2^n.

C. Fünfzig *et al.* ont réalisé un prototype de ce solveur (Fünfzig *et al.*, 2009). Il résout des systèmes de taille arbitraire, par exemple avec 60 inconnues et équations, générés en calculant le pavage de cercles représentant un graphe planaire complètement triangulé. Il résout aussi des systèmes non quadratiques.

Cette partie est structurée ainsi : §3.1 définit le polytope de Bernstein. Ensuite, §3.2 montre comment l'optimisation d'une fonction objectif linéaire sur le polytope de Bernstein, en recourant à la programmation linéaire, permet de calculer des encadrements des valeurs d'un polynôme dans un pavé (§3.2.1), et de réduire un pavé tout en préservant les racines qu'il contient (§3.2.2). Les principales caractéristiques du nouveau solveur sont ensuite présentées en §3.3. Ce solveur ne fait quasiment plus référence aux bases tensorielles de Bernstein. §3.4 discute quelques détails d'implantation à savoir la mise à l'échelle en §3.4.1 et surtout comment prévenir les erreurs dues à l'imprécision numérique de l'arithmétique flottante en §3.4.2.

3.1. *Définition du polytope de Bernstein*

L'idée essentielle est d'encadrer l'ensemble semi-algébrique (*patch*) :

$$S = (x_1, x_2, \dots x_n, x_1^2, x_1x_2, \dots x_1x_n, x_2^2, \dots x_2x_n, \dots x_n^2), x \in [0,1]^n$$

par un polytope (un polyèdre convexe) de $\mathbb{R}^N$, $N = n(n+3)/2$ défini par ses hyperplans. S et ce polytope vivent dans un espace à N dimensions, une dimension par monôme (à l'exception du monôme 1). D. Michelucci a appelé ce polytope le polytope de Bernstein, car ses hyperplans sont les inéquations : $\mathcal{B}_i^{(2)}(x_k) \geq 0$ avec $i = 0, 1, 2$, ou des inéquations : $\mathcal{B}_i^{(1)}(x_k)\mathcal{B}_j^{(1)}(x_l) \geq 0$ avec $i, j \in [0,1]^2$.

Ultérieurement, chaque monôme sauf 1 sera associé à une variable d'un problème de programmation linéaire, par exemple les monômes x_i, x_i^2, x_ix_j seront associés aux variables X_i, X_{ii}, X_{ij}. Il faudra optimiser une fonction objectif linéaire en les variables X_i, X_{ii}, X_{ij}. Détaillons d'abord les hyperplans du polytope. Soit x une variable parmi $x_1, \dots x_n$. L'arc de courbe (x, x^2) avec $0 \leq x \leq 1$ est encadré par un triangle, Figure 9. Soit y la variable représentant le monôme x^2, les inéquations définissant ce triangle sont :

$$\mathcal{B}_0^{(2)}(x) = (1-x)^2 = x^2 - 2x + 1 \geq 0 \Rightarrow y - 2x + 1 \geq 0$$
$$\mathcal{B}_1^{(2)}(x) = 2x(1-x) = -2x^2 + 2x \geq 0 \Rightarrow -2y + 2x \geq 0$$
$$\mathcal{B}_2^{(2)}(x) = x^2 \geq 0 \Rightarrow y \geq 0$$

Il y a au total $3n$ de ces hyperplans. Soient x_i et x_j deux variables distinctes parmi $x_1, \dots x_n$. Il y a $n(n-1)/2$ telles paires. Par commodité, on les renomme x et y, et le monôme xy est représenté par une variable z. Dans l'espace 3D (x, y, z), la surface contenant les points $(x, y, z = xy), x \in [0,1], y \in [0,1]$ est un morceau de

paraboloïde hyperbolique dont l'enveloppe convexe est un tétraèdre, Figure 10. Les inéquations des plans de ce tétraèdre sont :

$$\begin{aligned}
&\mathcal{B}_0^{(1)}(x)\mathcal{B}_0^{(1)}(y) = (1-x)(1-y) = 1-x-y+xy \geq 0 \\
&\quad \Rightarrow 1-x-y+z \geq 0 \\
&\mathcal{B}_0^{(1)}(x)\mathcal{B}_1^{(1)}(y) = (1-x)y = y-xy \geq 0 \Rightarrow y-z \geq 0 \\
&\mathcal{B}_1^{(1)}(x)\mathcal{B}_0^{(1)}(y) = x(1-y) = x-xy \geq 0 \Rightarrow x-z \geq 0 \\
&\mathcal{B}_1^{(1)}(x)\mathcal{B}_1^{(1)}(y) = xy \geq 0 \Rightarrow z \geq 0
\end{aligned}$$

Il y a au total $2n(n-1)$ tels hyperplans.

Au total, le polytope de Bernstein a $3n + 2n(n-1) = n(2n+1)$ hyperplans dans un espace à $N = n(n+3)/2$. En fonction du nombre n d'inconnues du système d'équations, il a un nombre polynomial $O(n^2)$ d'hyperplans et donc de faces. Par contre il a une quantité exponentielle de sommets dans le cas multivarié ($n > 1$). En effet, sur l'espace $(x_1, x_2, \ldots x_n)$, le polytope de Bernstein se projette selon l'hypercube $[0,1]^n$, qui a 2^n sommets ; le polytope de Bernstein a donc au moins autant de sommets (la projection d'un polyèdre convexe ne peut pas avoir plus de sommets que le polyèdre initial).

En pratique, d'autres hyperplans sont utilisés pour réduire encore le polytope. Pour toute paire d'inconnues x, y différentes, est ajoutée l'inéquation : $(x-y)^2 \geq 0 \Rightarrow x^2 + y^2 - 2(xy) \geq 0$: en remplaçant les monômes x^2, y^2 et xy par les variables correspondantes, cela donne une inéquation linéaire dans $\mathbb{R}^N$, *i.e.*, un hyperplan qui tronque le polytope de Bernstein. De plus, pour chaque inconnue x parmi $x_1, \ldots x_n$ l'inéquation : $(x-1/2)^2 \geq 0 \Rightarrow x^2 - x + 1/4 \geq 0$ est ajoutée (Figure 12).

Cette liste récapitule les inéquations des hyperplans (X_i est la variable du monôme x_i, X_{ii} celle de x_i^2, X_{ij} celle de $x_i x_j$) :

$0 \leq X_{ii} - 2X_i + 1$	$1 \leq i \leq n$	triangle
$0 \leq -2X_{ii} + 2X_i$	$1 \leq i \leq n$	triangle
$0 \leq X_{ii}$	$1 \leq i \leq n$	triangle
$0 \leq 1 - X_i - X_j + X_{ij}$	$1 \leq i < j \leq n$	tétraèdre
$0 \leq X_i - X_{ij}$	$1 \leq i < j \leq n$	tétraèdre
$0 \leq X_j - X_{ij}$	$1 \leq i < j \leq n$	tétraèdre
$0 \leq X_{ij}$	$1 \leq i < j \leq n$	tétraèdre
$0 \leq X_{ii} + X_{jj} - 2X_{ij}$	$1 \leq i < j \leq n$	complément
$0 \leq X_{ii} - X_i + 1/4$	$1 \leq i \leq n$	complément

Pour $n = 2$, le volume du polytope est approximativement 0.007, alors que le volume de l'hypercube en $N = 5$ dimensions est 1. Ce ratio des volumes décroît exponentiellement quand n augmente. Ce faible volume du polytope explique l'efficacité du solveur, quand on le compare à ceux qui utilisent l'arithmétique d'intervalles naïve :

cette dernière encadre S par l'hypercube, bien plus gros que le polytope de Bernstein. Le polytope de Bernstein se généralise à des degrés $d > 2$ plus élevés ; pour des monômes de degré maximal d, le polytope a $O(n^d)$ hyperplans, et un nombre exponentiel de sommets dans le cas multivarié, au moins 2^n, pour les mêmes raisons que dans le cas quadratique. La Figure 11 montre le polytope de Bernstein pour le cas univarié de degré 3 : il s'agit d'un tétraèdre.

3.2. *Le recours à la programmation linéaire*

3.2.1. *Calcul d'encadrements*

Calculer un encadrement de l'image de l'hypercube $[0,1]^n$ par un polynôme donné se réduit à un problème de programmation linéaire. La méthode est illustrée sur un exemple simple. Pour calculer un encadrement de $p(x) = 4x^2 + x - 3$, pour $x \in [0,1]$, la fonction linéaire objectif : $4y + x - 3$ est minimisée, puis maximisée, sur le polytope de Bernstein (le triangle en Figure 9) qui contient la courbe $(x, y = x^2), x \in [0,1]$. C'est un problème de programmation linéaire, après avoir remplacé le monôme x^2 par y. De gauche à droite, voici les tableaux de la méthode du simplexe pour le problème, pour le minimum, et pour le maximum. Les variables d'écart sont appelées b_0, b_1, b_2, en lien avec les polynômes de Bernstein $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2$; par exemple $\mathcal{B}_0$ est remplacé par la variable b_0.

$\min, \max : p = 4y + x - 3$	$\min p = -3 + x + 4y$	$\max p = 2 - 5b_0 - 9/2b_1$
$0 \leq b_0 = y - 2x + 1$	$b_0 = 1 - 2x + y$	$x = 1 - b_0 - b_1/2$
$0 \leq b_1 = -2y + 2x$	$b_1 = 2x - 2y$	$y = 1 - b_0 - b_1$
$0 \leq b_2 = y$	$b_2 = y$	$b_2 = 1 - b_0 - b_1$

La méthode du simplexe, due à George Dantzig en 1947 (Dantzing, 1990), effectue des pivots de Gauss sur les lignes du tableau initial, pour atteindre les tableaux finaux du minimum et du maximum. Seul le tableau du maximum est commenté. Dans $\max p = 2 - 5b_0 - 9/2b_1$, la valeur de p ne peut excéder 2 : en considérant que les variables de droite b_0 et b_1 sont nulles, les accroître ne peut que diminuer p à cause des coefficients négatifs $-5b_0 - 9/2b_1$ dans la fonction objectif ; et par ailleurs, toutes les variables, en particulier b_0 et b_1 doivent être positives. Les mêmes considérations s'appliquent pour le minimum $p = -3 + x + y$, qui est atteint pour $x = y = 0 : x$ et y peuvent seulement être augmentées, mais cela fait augmenter p à cause des coefficients positifs de x et y dans l'expression $p = -3 + x + y$. Quand le minimum (le maximum) se produit en un sommet extrêmal (c'est à dire "au dessus" d'un sommet de l'hypercube $[0,1]^n$), il est exact. Dans l'exemple, le minimum se produit en $x = 0$, et le maximum en $x = 1$, qui sont les 2 sommets de l'"hypercube" $[0,1]^1$; ils sont donc exacts.

On remarque qu'au sommet $v_0 = (0,0)$ où $b_1 = b_2 = 0$, la valeur de la fonction objectif p est $p_0 = p(0) = -3$; au sommet $v_1 = (1/2, 0)$ où $b_0 = b_2 = 0$, p vaut $p_1 = p(1/2) = -3/2$; au sommet $v_2 = (1,1)$ où $b_0 = b_1 = 0$, p vaut $p_2 = p(1) = 2$.

Ces valeurs p_0, p_1, p_2 sont les coefficients dans la base de Bernstein du polynôme $p(x) = p_0\mathcal{B}_0(x) + p_1\mathcal{B}_1(x) + p_2\mathcal{B}_2(x)$.

Cette propriété s'étend à tous les polynômes univariés, en admettant bien sûr que seules soient utilisées les inégalités de Bernstein $\mathcal{B}_i^{(d)}(x) \geq 0$; ceci est dû au fait que le polytope de Bernstein est un simplexe dans le cas univarié.

Un précurseur de cette méthode de calcul d'encadrement est Olivier Beaumont (Beaumont, 1999) ; dans sa thèse, il utilise les polynômes de Chebychev et la programmation linéaire pour encadrer des polynômes multivariés.

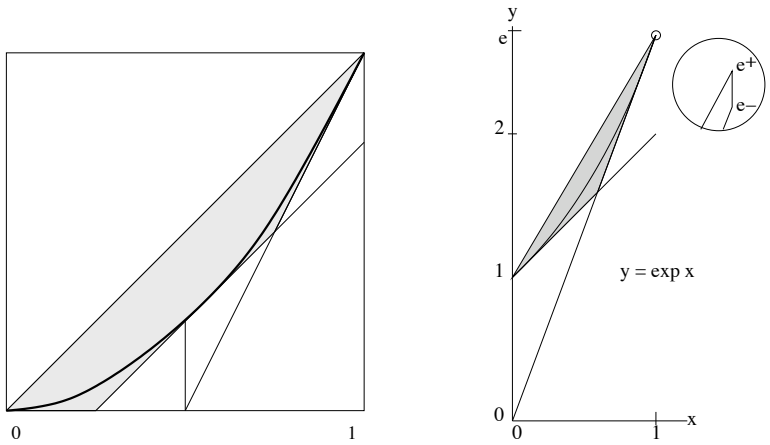

Figure 12. *Gauche : il est possible de réduire encore le polytope de Bernstein, e.g., avec l'inégalité* $x - y \leq 1/4$. *Droite : un polygone convexe enclôt l'arc de courbe* $(x, y = \exp x), 0 \leq x \leq 1$. $e = \exp 1$ *est inclus dans l'intervalle* $[e^-, e^+]$, *où* e^- *et* e^+ *sont deux nombres flottants successifs*.

3.2.2. *Réduction préservant les racines*

Ce paragraphe montre comment le solveur réduit les intervalles des inconnues x_i tout en préservant les solutions contenues dans ces intervalles.

Résoudre l'équation $4x^2 + x - 3 = 0$ avec $x \in [0, 1]$ équivaut à trouver les points d'intersection entre la droite d'équation $4y + x - 3 = 0$ et l'arc de courbe $y = x^2$, $x \in [0, 1]$. Cet arc de courbe est inscrit dans son polytope de Bernstein, le triangle de la Figure 9. Calculer l'intersection entre la droite et ce triangle, *i.e.,* trouver la valeur minimale et maximale de x de cette intersection, sont des problèmes de programmation linéaire. Les problèmes sont presque les mêmes que dans le paragraphe précédent,

seuls changent les fonctions objectifs ; ce coup ci, on souhaite minimiser et maximiser x. Les tableaux du simplexe sont :

$\min, \max : x$	$\min x = 3/5 + 2/5b_1$	$\max x = 7/9 - 4/9b_0$
$0 \leq b_0 = y - 2x + 1$	$y = 3/5 - 1/10b_1$	$y = 5/9 + 1/9b_0$
$0 \leq b_1 = -2y + 2x$	$b_0 = 2/5 - 9/10b_1$	$b_1 = 4/9 - 10/9b_0$
$0 \leq b_2 = y$	$b_2 = 3/5 - 1/10b_1$	$b_2 = 5/9 + 1/9b_0$

Donc l'intervalle $[0, 1]$ pour x est réduit à $[3/5, 7/9]$, et aucune racine n'est perdue. Pour réduire davantage l'intervalle, il faut utiliser une mise à l'échelle détaillée en §3.4.1 qui applique l'intervalle $x \in [3/5, 7/9]$ sur $x' \in [0, 1] : x = 3/5 + (7/9 - 3/5)x' = b + ax'$, et l'équation en fonction de x' est : $4a^2x'^2 + (8ab + b)x' + (b - 3) = 0$. La convergence autour d'une racine régulière est super quadratique, comme pour la méthode de Newton-Raphson. Informellement, l'argument est le suivant : si le système est linéaire, une seule réduction suffit, et le pavé réduit est égal au point solution ; or plus le pavé contenant une seule solution régulière est réduit, et plus le système est proche d'un système linéaire : géométriquement, les hypersurfaces deviennent de plus en plus proches de leurs hyperplans tangents. Une preuve mathématique de la vitesse de convergence est donnée par Mourrain et Pavone (Mourrain *et al.*, August 2005) ; cette preuve s'applique aux deux solveurs (§2, §3). Une conséquence importante de la vitesse de convergence est qu'une pile de petite taille est suffisante.

Remarque : si la droite de l'exemple ne coupe pas le triangle de Bernstein, ou plus généralement si le problème de programmation linéaire n'est pas réalisable, ceci prouve que le pavé considéré ne contient pas de racine.

3.3. *Caractéristiques du nouveau solveur*

Le polytope de Bernstein permet de proposer un nouveau solveur, qui ne se réfère plus à la méthode de Newton par intervalles, ni même aux bases de Bernstein.

Dans ce nouveau solveur, l'ensemble $S = (x_k, x_i x_j), i \in [1, n], j \in [1, n-1], k \geq j, x_i \in [0, 1]^n$ dans $\mathbb{R}^N$ est encadré comme décrit précédemment par un polytope. De plus chaque équation du système (*a priori* non linéaire) en x_i, $i \in [1, n]$ donne une équation linéaire qui décrit un hyperplan de $\mathbb{R}^N$ quand chaque monôme est remplacé par la variable correspondante. De même les inéquations algébriques du système, s'il y en a, sont converties en inéquations linéaires. Cette méthode prend donc en compte très facilement les inéquations algébriques.

Comme les solveurs classiques par intervalles, le nouveau solveur gère une pile de pavés à étudier. La pile est initialisée avec le pavé où on souhaite trouver les solutions du système. Tant que la pile est non vide, le pavé en sommet de pile est dépilé. Ce pavé est réduit, en réduisant l'intervalle pour chaque inconnue x_i ; au total, $2n$ optimisations linéaires sont donc effectuées : minimiser et maximiser x_i pour i de 1 à n. Si le pavé réduit est vide, alors il est garanti ne pas contenir de racine. S'il est non vide et

suffisamment petit, alors il est inséré dans une liste des solutions (il est possible de garantir qu'un pavé contient une seule racine, mais cette procédure n'est pas spécifique à ce solveur (Elber *et al*., 2001)). S'il est non vide et significativement réduit, alors une autre réduction est effectuée. Sinon, on peut il contient probablement plusieurs racines, et le couper en 2 est le seul moyen de séparer les racines ; le pavé est donc coupé en 2, par exemple selon l'inconnue x_i dont l'intervalle est le plus grand, ou bien selon l'inconnue x_i dont l'intervalle a été le moins réduit, et les deux pavés moitiés sont empilés. Plusieurs définitions pour "significativement réduit" sont possibles, et plusieurs tests de terminaison peuvent être imaginés ; cette question n'est pas détaillée ici.

Un avantage de ce solveur est que le préconditionnement du système, et l'inversion du jacobien, deviennent inutiles : la programmation linéaire fait l'essentiel du travail. Ce solveur est donc très simple, conceptuellement, et facile à programmer.

La Figure 9 à droite montre l'application de cette méthode sur l'équation $4x^2 + x - 3 = 0$, avec $x \in [0, 1]$. Soit y la variable représentant le monôme x^2. Alors l'intersection du polytope de Bernstein et de la droite d'équation $4x^2 + x - 3 = 0$ permet de réduire l'intervalle pour x à $[3/5, 7/9]$ sans perdre de racines. Ensuite cette intervalle est appliqué sur $[0, 1]$ par la mise à l'échelle : $x = 3/5 + (7/9 - 3/5)x'$, $x' \in [0, 1]$ (§3.4.1).

La Figure 13 montre des exemples 2D, donc pour des systèmes avec 2 inconnues et 2 équations. Tous les pavés sont dessinés, et représentés par des rectangles, aux côtés horizontaux ou verticaux. Le cadre de chaque sous figure est le pavé initial. Les points solutions sont représentés par un petit disque noir ; le pavé visible le plus petit qui contient ce disque noir est réduit en une opération de réduction à un pavé plus petit que le disque, et donc non visible sur la figure. Ceci illustre la convergence super-quadratique du solveur, pour les solutions régulières. Dans le cas d'une solution singulière (le point d'intersection entre deux coniques tangentes entre elles), la convergence n'est plus que linéaire, mais les réductions demeurent suffisamment significatives pour que le pavé ne soit pas coupé en deux. Les pavés sans solution sont détectés très vite.

Ce solveur a été utilisé dans (Fünfzig *et al*., 2009) pour résoudre des contraintes géométriques en 3D : la plateforme de Gough-Stewart (ou octaèdre), le calcul des droites tangentes à quatre sphères données, le calcul de pavages de cercles, etc. Ce dernier problème génère des systèmes de taille arbitrairement grande, qui sont insolubles avec le type classique des solveurs de Bernstein du §2.

3.4. *Détails de réalisation*

3.4.1. *Mise à l'échelle*

Un pavé réduit n'est plus $[0, 1]^n$. Une mise à l'échelle est nécessaire pour appliquer le pavé réduit $[u, v]$, où $u_i \leq v_i$, sur l'hypercube unité $[0, 1]^n$: soit $x_i = u_i + (v_i -$

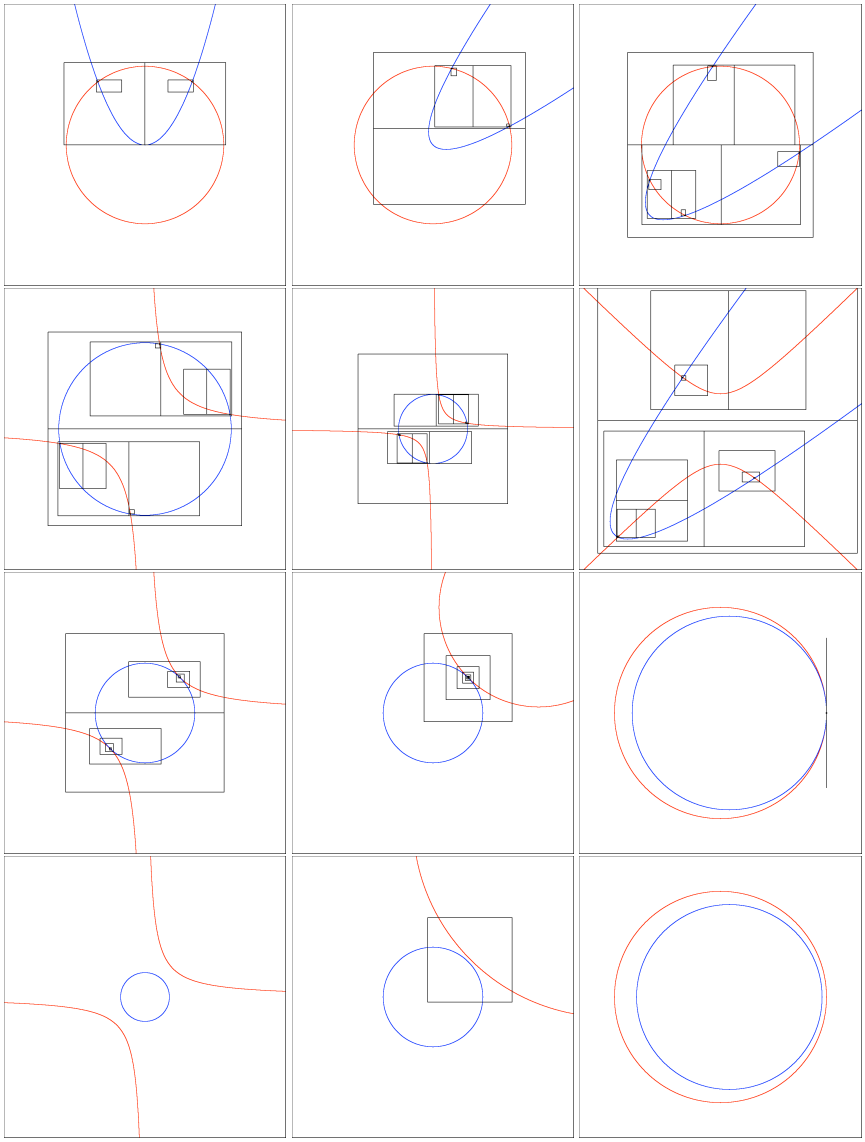

Figure 13. *Chaque figure montre tous les pavés rectangulaires, réduits ou coupés en 2. Deux premières rangées : la convergence est super quadratique autour d'un point solution non singulier. Troisième rangée : la convergence est linéaire pour les points d'intersection singuliers (aux contacts tangentiels entre deux courbes) ; cependant chaque côté du pavé est divisé par plus de 2 : autrement, le pavé serait divisé en deux. Dernière rangée : les pavés sans racine sont très vite détectés, une ou deux réductions donnant le pavé vide.*

$u_i)x'_i$, $w_i = v_i - u_i$, $x' \in [0,1]^n$. Alors $x_i^2 = w_i^2 x_i'^2 + 2u_i w_i x'_i + u_i^2$, $x_i x_j = w_i w_j x'_i x'_j + u_i w_j x'_j + u_j w_i x'_i + u_i u_j$. La mise à l'échelle est donc une transformation linéaire (affine, plus précisément) dans l'espace $\mathbb{R}^N$ des monômes, les variables de programmation linéaire.

Une autre méthode est possible ; elle adapte le polytope de Bernstein au pavé des x_i et laisse inchangées les équations et inéquations algébriques du système : $f(x) = 0, g(x) \leq 0$. Pour un pavé $x_i = [u_i, v_i]$, les inéquations des hyperplans du polytope de Bernstein sont modifiées comme suit :

$\mathcal{B}_0^{(2)}(x_i) = (1-x_i)^2 \geq 0$ devient : $(v_i - x_i)^2 = x_i^2 - 2v_i x_i + v_i^2 = q_i - 2v_i x_i + v_i^2 \geq 0$.

$\mathcal{B}_1^{(2)}(x_i) = 2x_i(1-x_i) \geq 0$ devient : $2(x_i - u_i)(v_i - x_i) = 2(-q_i + (u_i + v_i)x_i - u_i v_i) \geq 0$.

$\mathcal{B}_2^{(2)}(x_i) = x_i^2 \geq 0$ devient : $(x_i - u_i)^2 = q_i - 2u_i x_i + u_i^2 \geq 0$.

$\mathcal{B}_0^{(1)}(x_i)\mathcal{B}_0^{(1)}(x_j) = (1-x_i)(1-x_j) \geq 0$ devient : $(v_i - x_i)(v_j - x_j) = x_{ij} - v_i x_j - v_j x_i + v_i v_j \geq 0$.

$\mathcal{B}_0^{(1)}(x_i)\mathcal{B}_1^{(1)}(x_j) = (1-x_i)x_j \geq 0$ devient : $(v_i - x_i)(x_j - u_j) = -x_{ij} + u_j x_i + v_i x_j - v_i u_j \geq 0$.

$\mathcal{B}_1^{(1)}(x_i)\mathcal{B}_1^{(1)}(x_j) = x_i x_j \geq 0$ devient : $(x_i - u_i)(x_j - u_j) = x_{ij} - u_j x_i - u_i x_j + u_i u_j \geq 0$. De même pour les autres inéquations.

3.4.2. *Imprécision numérique*

Le polytope épouse si bien l'ensemble quadratique $S \subset \mathbb{R}^N$ des points de coordonnées $(X_i = x_i, X_{ij} = x_i x_j)$ que certaines racines sont perdues, suite aux erreurs d'arrondi de l'arithmétique flottante. Par exemple, pour résoudre $x^2 - x = 0$ avec $x \in [0,1]$, la droite $y - x = 0$ est considérée (Figure 9) ; si l'équation de cette droite devient : $y - x = \epsilon$ avec $\epsilon > 0$ à cause d'une erreur d'arrondi, alors aussi petit que soit ϵ les deux racines sont perdues. Par concision, on ne mentionne que le principe de trois solutions pour résoudre le problème d'imprécision numérique.

La première et la plus simple est le recours à une arithmétique rationnelle exacte (éventuellement paresseuse (Michelucci *et al*., 1997)). Cette solution a été programmée par D. Michelucci et fonctionne, mais elle est bien sûr très lente. Il est possible d'accélérer un peu cette méthode en arrondissant vers l'extérieur les bornes des intervalles des inconnues x_i : ainsi, si x_i est dans l'intervalle $[5001/10000, 10029/10031]$, cet intervalle peut être arrondi sur $[1/2, 1]$: ce dernier n'est guère plus grand et utilise des rationnels bien plus courts (avec moins de chiffres).

La seconde solution recourt à l'arithmétique d'intervalles (Beaumont, 1999; Kearfott, 1996) ; justement la recherche sur la programmation linéaire avec intervalles ou avec des tolérances est très active actuellement.

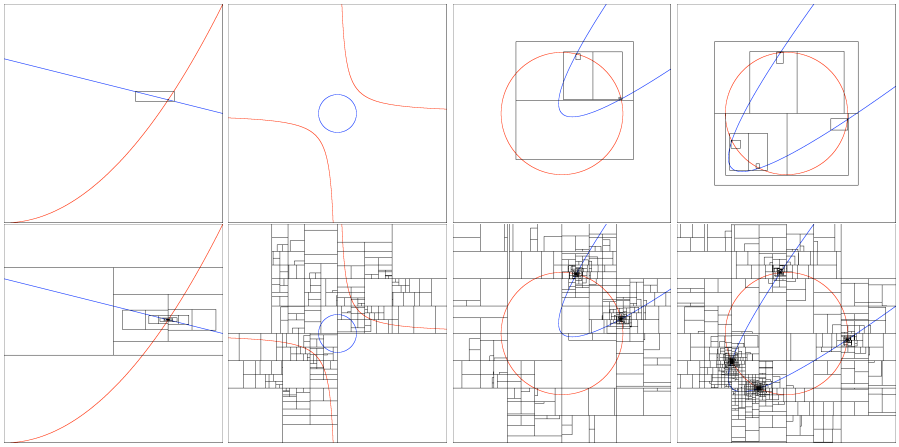

Figure 14. *Ces exemples 2D de systèmes à 2 équations et 2 inconnues sont les mêmes en haut et en bas. En haut, les pavés sont réduits avec le nouveau solveur, utilisant le polytope de Bernstein. En bas, les pavés sont réduits avec un solveur de type Newton par intervalles. Clairement, le nouveau solveur détecte bien plus vite les pavés sans racine, et converge avec moins de subdivisions. Sur ces exemples en faible dimension, le solveur classique en §2 et le nouveau solveur en §3 ont des performances très voisines.*

C. Fünfzig *et al.* ont utilisé une troisième solution, détaillée dans (Fünfzig *et al.*, 2009). L'analyse due à Wilkinson de la propagation d'erreur dans la résolution de systèmes linéaires permet de majorer les erreurs relatives et absolues. De plus, cette solution a l'avantage d'être compatible avec les nombreux solveurs de programmation linéaire existants, qui utilisent l'arithmétique flottante, tels SoPlex 1.4.1 (qui est utilisé dans (Fünfzig *et al.*, 2009)), GLPK, etc.

4. Conclusion

Dans la première partie de cet article nous avons revu la définition et les propiétés essentielles des bases tensorielles de Bernstein, ainsi que les conversions entre ces bases et la base canonique, avant de présenter les applications de ces bases pour l'isolation des racines des polynômes univariés et multivariés, et de discuter les difficultés des solveurs utilisant ces bases. Ces difficultés sont dues à l'imprécision numérique d'une part et au coût de la représentation de polynômes dans la base tensorielle de Bernstein d'autre part.

Dans la seconde partie nous avons introduit une nouvelle notion mathématique appelée "polytope de Bernstein" et présenté une méthode de réduction en temps polynomial pour surmonter la difficulté due au coût exponentiel de la représentation des polynômes multivariés dans la base tensorielle de Bernstein. Le solveur obtenu ne se réfère plus à la méthode de Newton par intervalles, ni même aux bases de Bernstein. Ce solveur a déjà été utilisé pour résoudre des contraintes géométriques en 3D : la plateforme de Gough-Stewart (ou octaèdre), le calcul des droites tangentes à quatre sphères données, le calcul de pavages de cercles, etc. Ce dernier problème génère des systèmes de taille arbitrairement grande, qui sont insolubles avec le type classique des solveurs fondés sur les bases tensorielles de Bernstein.

Des implantations GPU de ce solveur sont envisageables : en extrapolant le gain en performances que permettent les cartes GPU sur des programmes numériques voisins, elles pourraient résoudre en temps réel des systèmes de plusieurs dizaines d'inconnues.

5. Bibliographie

Beaumont O., Algorithmique pour les intervalles, PhD thesis, IRISA, projet Aladin, 1999.

Chvatal V., *Linear Programming (Series of Books in the Mathematical Sciences)*, W. H. Freeman, September, 1983.

COPRIN, ALIAS-C++ : A C++ Algorithms Library of Interval Analysis for equation Systems, Version 2.3, Technical report, INRIA Sophia-Antipolis, 2004.

Cormen T., Leiserson C., Rivest R., Stein C., *Introduction to Algorithms. Section 33.3, pp. 947-957 : Finding the convex hull*, 2nd edition edn, MIT Press and McGraw-Hill, 2001.

Dantzing G., *A history of scientific computing*, Reading, Ma, USA, chapter Origins of the simplex method, p. 141-151, 1990.

Delanoue N., Jaulin L., Cottenceau B., « Guaranteeing the homotopy type of a set defined by nonlinear inequalities », *Reliable Computing*, vol. 13, n° 5, p. 381-398, 2007.

Durand C. B., Symbolic and Numerical Techniques for Constraint Solving, PhD thesis, Purdue University, 1998.

Elber G., Kim M.-S., « Geometric constraint solver using multivariate rational spline functions », *SMA'01 : Proc. of the 6th ACM Symp. on Solid Modeling and Applications*, ACM Press, New York, NY, USA, p. 1-10, 2001.

Farin G., *Curves and Surfaces for CAGD : A Practical Guide*, Academic Press Professional, Inc., San Diego, CA, 1988.

Fünfzig C., Michelucci D., Foufou S., « Nonlinear System Solver in Floating-Point Arithmetic using LP Reduction », *Proc. of the ACM Symp. on Solid and Physical Modeling, San-Francisco, California, october 5-8*, p. 123-134, 2009.

Garloff J., Smith A. P., « Investigation of a subdivision based algorithm for solving systems of polynomial equations », *Journal of nonlinear analysis : Series A Theory and Methods*, vol. 47, n° 1, p. 167-178, 2001.

Hu C.-Y., Patrikalakis N., Ye X., « Robust Interval Solid Modelling. Part 1 : Representations. Part 2 : Boundary Evaluation », *CAD*, vol. 28, n° 10, p. 807-817, 819-830, 1996.

Jermann C., Michelucci D., Schreck P., « Modélisation géométrique par contraintes », *in* B. Péroche, D. Bechmann (eds), *Informatique graphique, modélisation géométrique et animation*, Hermès - Lavoisier, p. 185-214, 2007.

Kearfott R., *Rigorous Global Search : Continuous Problems*, Kluwer, Dordrecht, Netherlands, 1996.

Martin R., Shou H., Voiculescu I., Bowyer A., Wang G., « Comparison of Interval Methods for Plotting Algebraic Curves », *Computer Aided Geometric Design*, vol. 19, n° 7, p. 553-587, 2002.

Michelucci D., « Solving geometric constraints by homotopy », *IEEE Trans on Visualization and Computer Graphics*, vol. 2, p. 28-34, 1996.

Michelucci D., Foufou S., « Interval-based Tracing of Strange Attractors », *Int. J. Comput. Geometry Appl.*, vol. 16, n° 1, p. 27-40, 2006a.

Michelucci D., Foufou S., « Bernstein basis for interval analysis : application to geometric constraints systems solving », *in* Bruguera, Daumas (eds), *8th Conference on Real Numbers and Computers*, p. 37-46, July 2008.

Michelucci D., Foufou S., Lamarque L., Schreck P., « Geometric constraints solving : some tracks », *ACM Symp. on Solid and Physical Modelling*, p. 185-196, 2006b.

Michelucci D., Moreau J.-M., « Lazy Arithmetic », *IEEE Transactions on Computers*, vol. 46, n° 9, p. 961-975, September, 1997.

Mourrain B., Pavone J.-P., Subdivision methods for solving polynomial equations, Technical Report n° RR-5658, INRIA, August 2005.

N. Delanoue L. J., Cottenceau B., « Using interval arithmetic to prove that a set is path-connected », *Theoretical Computer Science, Special issue : Real Numbers and Computers*, vol. 351, n° 1, p. 119-128, 2006.

Paiva A., de Figueiredo L. H., Stolfi J., « Robust visualization of strange attractors using affine arithmetic », *Computers & Graphics*, vol. 30, n° 6, p. 1020-1026, 2006.

Preparata F., Hong S., « Convex Hulls of Finite Sets of Points in Two and Three Dimensions », *Commun. ACM*, vol. 20, n° 2, p. 87-93, 1977.

Reuter M., Mikkelsen T. S., Sherbrooke E. C., Maekawa T., Patrikalakis N. M., « Solving nonlinear polynomial systems in the barycentric Bernstein basis », *Vis. Comput.*, vol. 24, n° 3, p. 187-200, 2008.

Sherbrooke E. C., Patrikalakis N. M., « Computation of the solutions of nonlinear polynomial systems », *Comput. Aided Geom. Des.*, vol. 10, n° 5, p. 379-405, 1993.

Sommese A., Wampler C., *Numerical solution of polynomial systems arising in engineering and science*, World Scientific Press, Singapore, 2005.

Varadhan G., Krishnan S., Sriram T., Manocha D., « Topology Preserving Isosurface Extraction for Geometry Processing », *Second Eurographics Symposium on Geometry Processing*, p. 235-244, 2004.

Biographies des auteurs

Abdelkarim Tahari est maître assistant au département de Génie Informatique et membre du Laboratoire de mathématiques et d'informatique (LMI) de l'Université de

Laghouat, Algérie, Il a obtenu son diplôme d'ingénieur en informatique option machines et logiciels en 1992 de l'université d'Oran et son diplôme de magister en informatique option systèmes informatiques de l'Institut nationale d'Informatique d'Alger, Algérie. Depuis Novembre 2002, il poursuit son doctorat à l'Institut nationale d'Informatique d'Alger en collaboration avec les membres de l'équipe synthèse d'images du laboratoire Le2i de l'Université de Bourgogne, Dijon. Sa thèse concerne la modélisation géométrique des courbes et surfaces, ainsi que les contraintes géométriques et leurs applications en 2D et en 3D.

Christoph Fünfzig a obtenu un master d'informatique de l'Université de Bonn et un doctorat de l'Université Technique de Braunschweig, Allemagne, sur l'utilisation de requêtes géométriques dans les systèmes de graphe de scène. De Mai 2007 à Janvier 2008, il a travaillé en tant que chercheur invité au laboratoire PRISM de l'Arizona State University, Tempe, USA. Il est actuellement en postDoc au laboratoire Le2i à l'Université de Bourgogne, Dijon, France. Ses principaux intérêts de recherche portent sur la géométrie algorithmique, les contraints géométriques, la visualisation scientifique, et les nouvelles interfaces pour les modeleurs géométriques.

Dominique Michelucci soutient sa thèse d'informatique en 1987 et son habilitation à diriger des recherches en 1995, toutes deux à l'Ecole Nationale Supérieure des Mines de Saint-Etienne ; il y est ingénieur de recherche de 1990 à 2001. Il est aussi architecte DPLG de l'Ecole d'Architecture de Marseille-Luminy depuis 1989. En 2001, il devient professeur d'informatique au LE2I (UMR CNRS 5158, Laboratoire Electronique, Informatique et Image) à Dijon. Ses recherches portent sur la robustesse des calculs géométriques, sur la modélisation géométrique par contraintes, et sur la résolution numérique des systèmes de contraintes. En 1992, il est l'un des concepteurs de l'arithmétique exacte paresseuse, aujourd'hui très utilisée dans les logiciels de géométrie algorithmique, sous le nom de filtres arithmétiques. En 1993, il propose l'utilisation de la théorie du couplage dans les graphes bipartis pour l'étude qualitative des contraintes géométriques : cette étude détecte les sur-contraintes, et décompose en sous systèmes irréductibles. Il a par la suite proposé plusieurs méthodes pour surmonter cette limitation notamment : la méthode du témoin (2006), et les polytope de Bernstein (2008).

Sebti Foufou a soutenu une thèse sur les intersections de surfaces paramétriques en Juillet 1997 à l'Université Claude Bernard Lyon I. Il est en poste à l'Université de Bourgogne à Dijon, depuis Sept. 1998, comme Maître de conférences, puis professeur, en informatique. Ses activités de recherche concernent : La modélisation géométrique des courbes et surfaces et la modélisation des données du cycle de vie du produit (PLM). Il a travaillé sur la représentation des formes 3D par les cyclides de Dupin (surfaces algébriques de degré 4), sur la comparaison et l'analyse multirésolution de maillages, sur la reconstruction 3D, et sur les systèmes de contraintes géométriques (la formalisation, détection de dépendances, et résolution). Sebti Foufou a également travaillé sur les problématiques du PLM au National Institute of Standards and Technology, MD, Etats-Unis. Dans ce cadre, il a étudié la définition des modèles

de données pour représenter les informations de produit et la classification des normes pour le PLM. Il est en détachement à l'Université du Qatar depuis Sept. 2009.

Samy Ait-Aoudia a obtenu un diplôme d'études approfondies en traitement d'images de l'Université de Saint-Etienne, France, en 1990, et un doctorat en informatique de l'école des Mines de Saint-Etienne en 1994. Il est actuellement professeur à l'Institut national d'informatique d'Alger, Algérie ou il assure l'enseignement et la coordination de plusieurs cours aux élèves ingénieur et en post graduation. Ses activités de recherche concernent le traitement d'images et la modélisation par contraintes géométriques des courbes et surfaces.