BERNSTEIN BASES for INTERVAL ANALYSIS

Draft

1 Introduction

This text intents to be a self contained introduction for using tensorial Bernstein bases to compute sharp ranges for the values of a multivariate polynomial inside a box, *i.e.,* a vector of intervals. This introduction should be useful to people in interval analysis, since several reference textbooks on interval analysis (or on the resolution of polynomial systems) do not mention Bernstein bases.

In a nutshell, tensorial Bernstein bases provide sharp enclosures for the value of a multivariate polynomial inside a box. It is also possible to decide if the bounds are tigh or not: when the min or max value of the range is the value of the polynomial at a vertex of the box, then this bound is exact (up to rounding errors). The superiority of Bernstein bases over the naive interval arithmetic is illustrated Fig. 1: implicit algebraic curves $f(x, y) = 0$ are displayed with the classical subdivision method; in the first row, the naive interval arithmetic is used; in the last row, bounds are computed with the tensorial Bernstein base. Clearly, the last method needs much less subdivision to cull domains which are not crossed by the curve.

Bernstein bases are optimal regarding the problem of control of inaccuracy and numerical instability; this is basic common knowledge in CAD CAM [3, 4] and in the computer geometry community: there, people deal with models of curves and surfaces, and they expect a small perturbation on the model to result in a small, smooth and predicable motion or deformation on the curve/surface; it is achieved with Berstein bases, or some of their extensions (*e.g.,* splines for piecewise algebraic functions, or subdivision schemes). In comparison, the canonical base is terribly unstable, as illustrated by Wilkinson's polynomials. Last but not least, Bernstein bases bring geometric insight and intuition on algebraic and numerical problems.

Let us mention now the main features of our approach. It performs some simple symbolic operations on polynomials (sums, products, derivatives) but it does not need expensive operations used in computer algebra, such as resultants, gcds or standard bases. The coefficients of polynomials are represented by intervals, to account for rounding errors: thus these interval remain narrow during computation (some ULPs). Classically, in Interval analysis, the use of interval arithmetic is two-folds: on one hand, intervals enclose rounding errors

of floating point arithmetic, and on the other hand, interval arithmetic is used to compute the range of functions on large domains: the width of these intervals can be huge. Our approach uses interval arithmetic only to account for rounding errors, *i.e.,* no computation involves intervals. Indeed, sharp ranges of polynomials inside boxes are computed relying on properties of Bernstein bases (and not on the classical interval arithmetics, naive or centered): partition of unity, convex hull property, variation diminishing property, affine invariance.

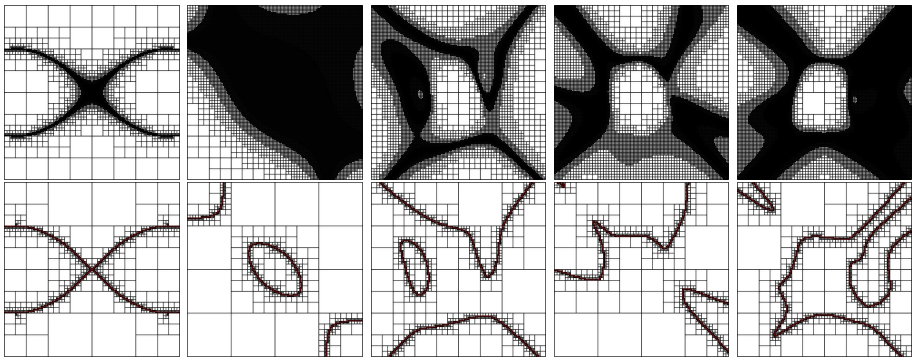

Figure 1: *Above*: naive interval arithmetic. *Below*: Bernstein based arithmetic. *Left to right columns*: Cassini oval: $C_{2,2}(x,y) = 0$ in $[-2,2] \times [-2,2]$, where $C_{a,b}(x,y) = ((x+a)^2 + y^2) \times ((x-a)^2 + y^2) - b^4$. The curve [6] $f(x,y) = 15/4 + 8x - 16x^2 + 8y - 112xy + 128x^2y - 16y^2 + 128xy^2 - 128x^2y^2 = 0$ on the square $[0,1] \times [0,1]$. Random algebraic curves with total degree 10, 14, 18.

Section 2 provide basic formulas for conversion back and from the Bernstein base and the canonical base, and essential properties of the Bernstein base. Section 3 presents basic tools solving algebraic systems with Bernstein bases. Section 4 discusses some miscellaneous issues. Section 5 concludes, presenting limitations of the approach and future works.

None of the methods presented here is new; they are well known and widely used in computer graphics, computer geometry, CAD-CAM; all principles were laid down in CAD-CAM with de Casteljau's work in 1959 (in industry, at Citroën), which have been covered until 1975 when W. Böhm made them public. Meanwhile, P. Bézier began and published his work on UNISURF in the sixties, in Renault, another french car company; thus free form curves and surfaces are today called Bézier curve and surfaces. Arguably, Bernstein bases and Bézier curves and surfaces, and their offspring (spline bases for piecewise algebraic functions, and the today fashionable subdivision curves and surfaces or volumes) are at the heart of CAD-CAM.

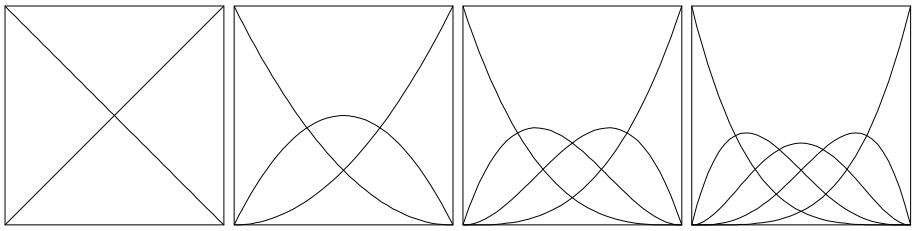

Figure 2: The Bernstein polynomials of degree 1, 2, 3, 4, from left to right. Each square is the unit square. The maximum (or minimum) value of $B_i^{(d)}$ occurs at $x = i/d$.

2 All useful formulas

$C(i,n) = \frac{n!}{(n-i)!i!}$ is the number of ways to choose i objects among n.

2.1 Univariate polynomial

Consider first the univariate case. Let $P(x) = \sum c_i x^i$ be a polynomial in the canonical base $(x^0, x^1, x^2, \dots x^d)$. $P(x)$ can also be expressed in the Berstein base. The Bernstein base polynomials of degree d are the $d+1$ polynomials $(B_0^{(d)}(x), B_1^{(d)}(x), \dots B_n^{(d)}(x))$ defined by

$$B_i^{(d)}(x) = C(i,d) \times x^i \times (1-x)^{d-i}, i = 0, 1, \dots d$$

The Bernstein polynomials appear in the Newton expansion of the expression $(x+(1-x))^d = \sum_{i=0}^d B_i^{(d)}(x)$; this implies that for all x, the sum of the Bernstein polynomials equals 1. Conversely:

$$x^k = \frac{1}{C(k,d)} \sum_{i=k}^{d} C(k,i) B_i^{(d)}(x) \quad \text{thus } x = (1/d) \times \sum_{i=0}^{d} i B_i^{(d)}(x) \quad \text{and } x^0 = 1 = \sum_{i=0}^{d} B_i^{(d)}(x)$$

The Bernstein base is especially interesting for $x \in [0,1]$. In this interval, all Bernstein polynomials obviously lie between 0 and 1: $0 \leq B_i^d(x) \leq 1$, and their sum equals 1 (it is said that Bernstein polynomials realize a partition of unity). Thus for $x \in [0,1]$, the sum $f(x) = \sum_{i=0}^d b_i B_i^d(x)$ is a linear convex combination of the coefficient b_is, thus this sum lies between the minimum and the maximum of the coefficients b_is. It gives an enclosure of $f(x)$ for $x \in [0,1]$. It is possible to be even more precise, see Fig. 3, expressing the polynomial x in the Bernstein base: $x = (1/d) \times \sum_{i=0}^d C(k,i) B_i^{(d)}(x)$; then for each $x \in [0,1]$, the point $(x, y = f(x))$ is a linear convex combination of the points $(i/d, b_i)$; thus this arc of curve $(x \in [0,1], y = f(x))$ lies inside the convex hull of the points $(i/d, b_i)$, *i.e.,* the smallest convex and close polygon containing points $(i/d, b_i)$. In formula, $(x, f(x)) = \sum_i B_i^{(d)}(i/d, b_i)$. These points are called control points,

the polygon with vertices $(i/d, b_i)$ with $i = 0, \ldots d$ is called the control polygone, and the b_is are called Bernstein coefficients. Note the curve passes through the first point where $i = 0, x = 0, y = b_0$ and the last one where $i = d, x = 1, y = b_d$ (usually the curve does not pass through the other intermediary points). These 2 extreme (relatively to x) points are vertices of the convex hull. Thus if the minimum (maximum) of the coefficients b_is is b_0 or b_d, then this bound for $f(x \in [0,1])$ is exact, up to rounding errors. Moreover, the roots of $f(x) = 0$ inside $[0,1]$ must lie in the intersection between the convex hull and the x axis. This last property can be used to shrink the interval containing roots of $f(x) = 0$ (section 3.4).

The convex hull property is a consequence of the partition of unity property. Another consequence is the variation diminishing property: a 2D Bézier curve can not cross a line more often than its control polygon. Finally, a nice feature of the Bernstein base is its affinity invariance: if ϕ is any affine transform (rotation, translation, scaling) and C any Bézier curve, surface, etc, with control points P_is, then the control points of $\phi(C)$ are $\phi(P_i)$ (this property does not hold with the canonical base). All offsprings of Bernstein bases fulfil the same properties, which make them relevant and essential for the geometric modelling of curves, surfaces, volumes, etc in CAD-CAM and computerized geometry.

To ease the conversion from the canonical base to the Bernstein one, it is convenient for the programmer to define the contribution, or weight, of the monomial x^k to the i th Bernstein polynomial of degree d; it is noted $W([0,1], x^k, B_i^{(d)})$ and:

$$W([0,1], x^k, B_i^{(d)}) = C(k,i)/C(k,d)$$

So far we have considered x to lie in the $[0,1]$ interval. To compute the range of $f(x)$ for $x \in [u,v]$, just use the linear mapping: $x = u+(v-u)x'$: when x spans $[u,v]$, x' spans $[0,1]$. Thus, posing $w = v - u$ for convenience, the contribution of x^k, with $x \in [u,v]$, to $B_i^{(d)}(x)$ which is noted $W([u,v], x^k, B_i^{(d)}(x))$, equals $W([0,1], (u + (v-u)x)^k, B_i^{(d)}(x))$:

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

The formula holds for any w, positive, negative, or zero.

Conversion algorithm: to enclose $f(x)$ for $x \in [u,v]$, we convert $f(x)$ from the canonical base to the Bernstein base; first initialize all coefficients of the

Bernstein form to 0, then for each canonical term $c_k \times x^k$ in the polynomial $f(x)$, and for all i in $0, 1, \ldots d$, add $c_k \times W([u, v], x^k, B_i^{(d)})$ to the coefficient of the i th Bernstein polynomial $B_i^{(d)}$. The 2 loops on term x^k and Bernsten $B_i^{(d)}$ can be done in any order; indices k and i can be considered in any order. If terms $c_k x^k$ are stored in a list, the same power x^k (for the same k) may harmlessly occur several times.

2.2 Multivariate polynomial

To examplify the multivariate case, and for simplicity, we use polynomials in 2 variables, x and y. The canonical base is the cartesian product of the canonical base in x: $(x^0, x^1, \ldots x^d)$ and of the canonical base in y: $(y^0, y^1, \ldots y^e)$, where d is the maximal degree in x and e the maximal degree in y for all monomials of the considered polynomial. d and e can be different. For $d = 3, e = 2$, the base is (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). Similarly, the tensorial Bernstein base is the cartesian product of the Bernstein base in x: $(B_0^{(d)}(x), B_1^{(d)}(x), \ldots B_d^{(d)}(x)$ and of the Bernstein base in y: $(B_0^{(e)}(y), B_1^{(e)}(y), \ldots B_e^{(e)}(y))$ This base has the same number of elements as the tensorial canonical base (all bases of a vectorial space have the same cardinal, in finite dimension). The notation $B_{i,j}^{(d,e)}(x, y) = B_i^{(d)}(x) \times B_j^{(e)}(y)$ is used for convenience.

The polynomial is expressed as a set of terms: $c_{k,l}x^k y^l$ in the canonical base. The contribution of each term $c_{k,l}x^k y^l$ to the Bernstein polynomial $B_{i,j}^{(d,e)}(x, y) = B_i^{(d)}(x) \times B_j^{(e)}(y)$ is $c_{k,l}$ times the product of the contribution of x^k to $B_i^{(d)}(x)$ and of the contribution of y^l to $B_j^{(d)}(y)$. In formula:

$$W([0,1], c_{k,l}x^k y^l, B_{i,j}^{(d,e)}(x, y)) = c_{k,l} \times W([0,1], x^k, B_i^{(d)}(x)) \times W([0,1], y^l, B_j^{(e)}(y))$$

To convert a polynomial in n variables from the canonical base to the tensorial base, initialize to 0 all coefficients in the Bernstein representation. Then, for each term $c_k x^k = c_{k_1,k_2,\ldots k_n} x_1^{k_1} x_2^{k_2} \ldots x_n^{k_n}$ of the polynomial, and for each index $i = i_1 i_2 \ldots i_n$, with $0 \leq i_1 \leq d_1, 0 \leq i_2 \leq d_2, \ldots 0 \leq i_n \leq d_n$ add the contribution of the term $c_k x^k$ to the Bernstein polynomial $B_i^{(d)} = B_{i_1}^{(d_1)}(x_1) \times B_{i_2}^{(d_2)}(x_2) \ldots \times B_{i_n}^{(d_n)}(x_n)$.

For computing the range of a multivariate polynomial in a box defined by two opposite vertices $U = (u_1, u_2, \ldots u_n)$ and $V = (v_1, v_2, \ldots v_n)$, define $w = (w_i = v_i - u_i)$. Then the contribution of each term $c_k x^k$ to the Bernstein polynomial B_i is (the formulas work for any w_i, negative, zero or positive)

$$W([u, v], c_k x^k, B_i(x) = c_{k_1,k_2,\ldots k_n} \times W([u_1, v_1], x_1^{k_1}, B_{i_1}^{(d_1)}(x_1)) \times \ldots \times W([u_n, v_n], x_n^{k_n}, B_{i_n}^{(d_n)}(x_n))$$

2.3 Convenient indexing of multivariate monomials

It is convenient to map integers and monomials of multivariate polynomials, so that multi-indexed coefficients can be stored in a one-dimensional array.

Each monomial $x^a = x_0^{a_0} x_1^{a_1} \dots x_{n-1}^{a_{n-1}}$ and each Bernstein polynomial $B_a^{(d)} = B_{a_0}^{(d_0)}(x_0) B_{a_1}^{(d_1)}(x_1) \dots B_{a_{n-1}}^{(d-1)}(x_{n-1})$ is indexed with the multi index $a = (a_0, a_1, \dots a_{n-1})$. Let $d = (d_0, d_1, \dots d_{n-1})$ be the vector of degrees of the polynomial in variables $x = (x_0, x_1, \dots x_{n-1})$. For convenience $d' = (d'_i = 1 + d_i)$. A possible indexing matches the monomial $a = (a_0, a_1, \dots a_{n-1})$ and the number $A = a_0 + a_1 d'_0 + a_2 d'_1 d'_0 + a_3 d'_2 d'_1 d'_0 + \dots$. Conversely, a can be computed from A with the algorithm: $\alpha_0 = A$, $a_k = \alpha_k \bmod d'_k$, $\alpha_{k+1} = \lfloor \alpha_k / d'_k \rfloor = (\alpha_k - a_k)/d'_k$ for $k = 0, \dots n-1$.

2.4 Matricial formulation

2.4.1 The univariate case.

The conversion from the canonical base to the Bernstein one and the inverse conversion are linear mappings, mathematically speaking, in other words they can be expressed with matrice multiplication, to use a parlance maybe more familiar to programmers. Pose $X = (x^0, x^1, \dots x^d)$ and $B = (B_0(x), B_1(x), \dots B_d(x))$. The polynomial $f(x) = \sum_{k=0}^{d} c_k x^k$ in the canonical base is represented by the vector $c = (c_0, c_1, \dots c_d)$, *i.e.,* $f(x) = X c^t$. Its representation in the Bernstein base is $f(x) = \sum_{i=0}^{d} b_i B_i^{(d)}(x)$, and is stored in a vector $b = (b_0, b_1, \dots b_d)$; *i.e.,* $f(x) = B b^t$. Moreover $X = BM$ where $M_{k,i} = W([0,1], x^k, B_i^{(d)})$ is a square matrice with $d+1$ rows and lines. From $f(x) = X c^t = B b^t$ and $X = BM$, we conclude: $b^t = M c^t$. It gives another way for the conversion.

2.4.2 The bivariate case.

Pose $X = (x^0, x^1, \dots x^d)$, $Y = (y^0, y^1, \dots y^e)$. The polynomial in the canonical base can be expressed as $f(x, y) = XCY^t$, with C a matrice with $d+1$ rows and $e+1$ columns; $C_{r,c}$ is the coefficient of $x^r y^c$. Using $X = B_x M_x$ and $Y = B_y M_y$ with the natural conventions:

$$f(x, y) = XCY^t = B_x M_x C (B_y M_y)^t = B_x (M_x C M_y^t) B_y^t$$

thus the coefficients of $p(x, y)$ in the tensorial Bernstein base are given by the matrice $T = M_x C M_y^t$, *i.e.,* $p(x, y) = B_x T B_y^t$.

2.4.3 Multivariate case.

The previous matricial formulation is often used in CAD-CAM, for parametric curves and surfaces. Unfortunately, the matricial formulation does not fit nicely to the trivariate case (parametric volumic functions) and beyond, since matrices are bi-dimensional arrays. Extensions to overcome this limitation have been proposed but are less natural and simple than the tensorial formalism used in this presentation, which is straightforward to program.

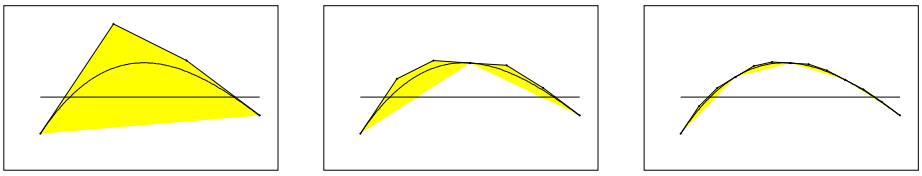

Figure 3: Two Casteljau iterations on a Bézier curve which is the graph of an univariate polynomial of degreee 3. The filled polygons are the convex hulls of the control points.

2.5 Bisections and range computations

Subdivision algorithm bisect boxes and need to compute the range for the two halves. Several algorithms can be used; first it is possible to use the previous formulas, *i.e.,* each computation starts from the coefficients in the canonical base and converts them to the Bernstein bases, with different boxes $[U, V]$; second, it is possible to use Casteljau method: given the Bernstein coefficients for a box, the Casteljau method computes the Bernstein coefficients for the two halves of the box, without conversion to the canonical base; third, it is possible to multiply the Bernstein coefficients in the mother box by a left (right) Casteljau matrice to obtain the Bernstein coefficients in the left (right) half of the mother box. All methods give equal results if exact arithmetic is used, and have the same time complexity; but regarding numerical stability, the first method is terribly bad, as the degrees increase, due to the bad condition number of the conversion matrice (thus, when intervals are used to represent Bernstein or canonical coefficients, their width increase a lot); Casteljau method is the best one, and Casteljau matrices can be used for mathematical considerations.

2.6 Casteljau algorithm

Casteljau method is first explained for the univariate case. Let $b^d = (b_0, b_1, \dots b_d)$ be the Bernstein coefficients of some polynomial $y = f(x), x \in [0, 1]$. Compute the vectors of the average values: $b^{d-1} = ((b_0 + b_1)/2, \dots (b_i + b_{i+1}))/2 \dots)$; b^{d-1} has one element less than b^d. From b^{d-1}, compute b^{d-2}, etc, until b^0 which contains only 1 element. In formula: $b_i^k = (b_i^{k+1} + b_{i+1}^{k+1})/2$, for $k = d-1, \dots 0$ and $i = 0, \dots k$. Then the first entry of $b^d, b^{d-1}, \dots b^0$ gives the vector of the Bernstein coefficients of the first half (left) of the curve, and the last entry of $b^0, \dots b^{d-1}, b^d$ gives the vector of the Bernstein coefficients of the second (right) half of the curve. In formula, the left half has coefficients $l_i = b_0^{d-i}$ and the right half has coefficients $r_i = b_i^i$, for $i = 0, \dots d$.

Instead of coefficients $1/2$, $1/2$ for averaging, any coefficient $1-t, t$ are usable (though numerical stability is lossed as t is outside $[0, 1]$ and $|t|$ grows). The first entries give the Bernstein coefficients of the arc of curve inside $[0, t]$ (instead of $[0, 1/2]$) and the last entries give Bernstein coefficients inside $[t, 1]$ (instead of

$[1/2, 1]$).

The averaging method $b_i^k = (b_i^{k+1} + b_{i+1}^{k+1})/2$ applies in any dimension, *i.e.,* when b_i^{k+1}, b_{i+1}^{k+1} are vectors instead of scalar values. Thus Casteljau method applies as well.

2.7 Casteljau matrices

$\mathbb{P}^{(d)}$, or $\mathbb{P}$ if d is omitted, is sometimes called Pascal matrice; it is the square matrice with $d+1$ rows and colums, with $\mathbb{P}_{l,c} = C(l, c)$. It is an upper triangular matrice, containing the Pascal triangle. Its inverse has entries $(\mathbb{P}^{-1})_{l,c} = \mathbb{P}_{l,c} \times (-1)^{l+c}$. For instance with $d = 4$:

$$\mathbb{P} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 & 6 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbb{P}^{-1} = \begin{pmatrix} 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & -2 & 3 & -4 \\ 0 & 0 & 1 & -3 & 4 \\ 0 & 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$\mathbb{L}^{(d)}$ or $\mathbb{L}$ could be called de Casteljau left matrice. $\mathbb{L}^{(d)}$ is equal to $\mathbb{P}^{(d)} \times \mathrm{diag}(1, 1/2, 1/4, \ldots 1/2^d)$: the columns of $\mathbb{L}^{(d)}$ are the columns of $\mathbb{P}^{(d)}$, divided by increasing powers of 2. $\mathbb{L}^{(d)}$ appears in de Casteljau algorithm: if $b = (b_0, b_1, \ldots b_d)$ is the vector of coefficients of an univariate polynomial in the Bernstein base, then $b\,\mathbb{L}$ is the vector of coefficients of the left half of the polynomial, in the Bernstein base. $\mathbb{R}^{(d)}$ or $\mathbb{R}$ could be called the de Casteljau right matrice. $b\,\mathbb{R}$ gives the vector of coefficients of the right half of the polynomial, in the Bernstein base. $\mathbb{R}^{(d)}$ has the same columns as $\mathbb{L}^{(d)}$, but in the reverse order. For example, for degree $d = 4$, $\mathbb{L}$ and $\mathbb{L}^{-1}$ are:

$$\mathbb{L} = \begin{pmatrix} 1 & 1/2 & 1/4 & 1/8 & 1/16 \\ 0 & 1/2 & 2/4 & 3/8 & 4/16 \\ 0 & 0 & 1/4 & 3/8 & 6/16 \\ 0 & 0 & 0 & 1/8 & 4/16 \\ 0 & 0 & 0 & 0 & 1/16 \end{pmatrix} \qquad \mathbb{L}^{-1} = \mathrm{diag}(1, 2, 4, 8, 16)\mathbb{P}^{-1} = \begin{pmatrix} 1 & -1 & 1 & -1 & \\ 0 & 2 \times 1 & -2 \times 2 & 2 \times 3 & -2 \times \\ 0 & 0 & 4 \times 1 & -4 \times 3 & 4 \times \\ 0 & 0 & 0 & 8 \times 1 & -8 \times \\ 0 & 0 & 0 & 0 & 16 \times \end{pmatrix}$$

If the coefficients for averaging are $1 - t, t$ instead of $1/2, 1/2$, then the left Casteljau matrice becomes $\mathbb{L}^{(d)} = \mathrm{diag}(1, \beta, \beta^2, \ldots \beta^d)\mathbb{P}^{(d)}\mathrm{diag}(1, \alpha, \alpha^2, \ldots \alpha^d)$ where $\alpha = 1 - t, \beta = t/\alpha$. Again, the right matrice has the same columns, but in reverse order.

The ubiquitous de Casteljau matrices, or Pascal matrice, appear not only in the "halving" de Castejau algorithm, but also in the conversion of an univariate polynomial from the canonical base to the Bernstein base; if $c = (c_0, c_1, \ldots c_d)$ is the coefficients vector in the canonical base, and $b = (b_0, b_1, \ldots b_d)$ the vector of coefficients if the Bernstein base: $(B_0^{(d)}, B_1^{(d)}, \ldots B_d^{(d)})$, then b and c are related by:

$$\begin{aligned} b &= c \times \mathrm{diag}_k(1/C(k, d)) \times \mathbb{P}^{(d)} \\ &= c \times \mathrm{diag}_k(1/C(k, d)) \times \mathbb{L}^{(d)} \times \mathrm{diag}(1, 2, 4, \ldots 2^d) \end{aligned}$$

3 Resolution of algebraic systems

3.1 Bézier curves and surfaces, interface between algebra and geometry

Bézier curves, surfaces, solids etc are basic primitives in computerized geometry. A Bézier curve is a parametric curve $(x(t), y(t)) = \sum_i B_i^{(d)}(t) p_i$ where $p_i = (x_i, y_i)$ are its given control points, and $t \in [0, 1]$. Of course the curve can lie in 3D: just add a $z(t)$ and z_i components, and beyond. Similarly, a Bézier surface is a parametric surface $p(u, v) = (x(u, v), y(u, v), z(u, v)) = \sum_i \sum_j B_i(u) B_j(v) p_{ij}$, with parameters u, v lying in the unit square. A trivariate parametrization u, v, w, and a 3D net of control points p_{ijk}, are used to defined Bézier volumes.

Computing intersection points between Bézier surfaces is a basic and well known problem in CAD-CAM. It is equivalent to solving an algebraic system. For instance, solving the bivariate system $f(x, y) = g(x, y) = 0$ with $0 \leq x, y \leq 1$ is equivalent to finding the intersection points between three Bézier surfaces; the first is $F(u, v) = (u, v, f(u, v))$, the second is $G(u, v) = (u, v, g(u, v)$, the third is $H(u, v) = (u, v, 0)$. Expressing the involved polynomials in the tensorial Bernstein base gives the control points of these Bézier surfaces. Thus the algebraic problem: solving an algebraic system, can be converted into a geometric problem: computing intersection points between Bézier (hyper)surfaces; this geometric formulation enables more intuitive methods and insight (it is possible to display curves and surfaces, to see them, to modify them interactively moving control points, etc), a better control on the numerical instability issues, and of course mathematical methods from numerical analysis, interval analysis, computer algebra, automatic differentiation still apply.

3.2 Isolating real roots of univariate polynomials

Bernstein bases enables a simple and fast algorithm to enclose real roots of a polynomial in an interval, wlog $[0, 1]$. Convert the polynomial in the Bernstein base. If all Bernstein coefficients have the same sign, the polynomial has no root. Otherwise study recursively the two halves of the interval. When the Bernstein coefficients increase (decrease) monotonously, and are negative at one end and positive at the other end, then the interval contains a single, regular root, and the standard Newton method is guaranteed to converge to the root. Intervals without roots are culled quickly; it is known that the convergence of Bernstein based subdivision methods is quadratic. Possibly, the intersection of the convex hull of the control points $(i/d, b_i)$ with the zero level can be computed, to contract the interval without losing roots.

To find roots of $f(x) = 0$ in $[1, +\infty)$, define $g(x) = x^d f(1/x)$: if $f(x) = \sum f_i x^i$ in the canonical base, then $g(x) = \sum f_i x^{d-i}$, *i.e.,* g is a polynomial with the same coefficients than f but in reverse order in the canonical base; and the roots of g inside $[0, 1]$ are clearly the inverses of the roots of f inside $[1, +\infty)$. This way all positive roots are found. To find negative roots of $f(x) = 0$, compute the positive roots of the polynomial $h(x) = f(-x)$.

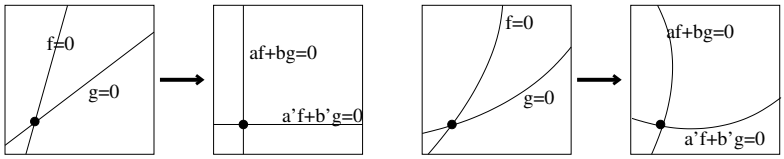

Figure 4: The effect of preconditionning for a linear system at left, for a non linear one, at right.

This kind of algorithm is used in computer graphics to ray trace algebraic implicit surfaces $f(x, y, z) = 0$ with total degree d, *i.e.,* to compute the intersection points between a ray (a half line) and the surface. The ray is parameterized with: $x = x_0 + at, y = y_0 + bt, z = z_0 + ct$, where the origin (x_0, y_0, z_0) of the ray is known, as well as its direction (a, b, c). Replacing x, y, z by their values in t gives an univariate algebraic equation in t, with degree d, which is solved by some variant of the previous method.

3.3 2D convex hull computations

Computing the convex hull of a set of 2D points is a basic problem of computerized geometry. Let (x_i, y_i) be a set of points in the plane x, y. We only explain how to compute the lower part of the convex hull of the (x_i, y_i) points: the upper part is computed in a similar way. First sort the points by increasing x_i; actually it is already done, since in our application, all x values are some i/d, where d is a degree, and i an integer in $0, \dots d$. Thus we can assume wlog that points are sorted in x increasing order: $x_0 < x_1 < \dots x_d$, and that there is only one point for each x_i value (the one with the smallest y). Initialize the lower convex hull with the two leftmost points $p_0 = (x_0, y_0), p_1 = (x_1, y_1)$, and $h = 1$. Then scan the (x_k, y_k) points from left to right (*i.e.,* with increasing x), for $k = 2, \dots d$, and update the lower convex hull as follows. Note p_h the rightmost point of the lch, and p_{h-1} the point juste before. While $p_{h-1}p_hp_k$ "turns right" (the angle $p_{h-1}p_hp_k$ is concave), remove the point p_h from the lower hull. Then add point p_k at the end of the lower convex hull. Three points p, q, r "turn right" when the determinant of $((p_x, p_y, 1), (q_x, q_y, 1), (r_x, r_y, 1))^t$ is negative. When it vanishes, points p, q, r are aligned; when it is positive, they turn left.

This algorithm is in $O(d \log d)$ if d is the number of points, the $d \log d$ factor is due to the sorting stage, which here is useless, so the method is linear in the number of points (the number of Bernstein coefficients).

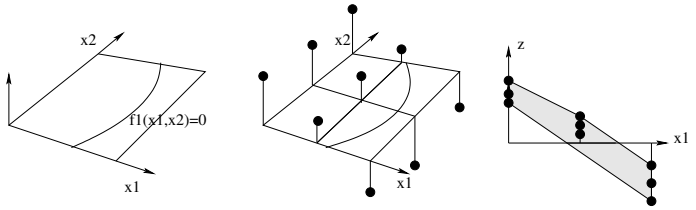

Figure 5: Equation $z = f_1(x_1, x_2) = 0$ has degree 2 in x_1 and x_2, and a grid of 3×3 control points. This curve is seen as the intersection between the zero level set $z = 0$ and the surface $z = f_1(x_1, x_2)$. This surface lie inside the convex hull of its control points $(i/2, j/2, b_{i,j}), i = 0, 1, 2, j = 0, 1, 2$. It is easy to compute the 2D convex hull of the projection on the x_1, z plane of the control points. The intersection of this 2D convex hull with the x_1 axis encloses all points of the curve.

3.4 Reducing a domain, preserving roots of a polynomial system

As it is well known in interval analysis, contraction methods, *i.e.,* methods which contract the considered domain while preserving the roots it contains, are interesting since they can avoid useless branchings (bisections), the cost of which is exponential. Bernstein bases enable to very efficiently contract domains around their contained roots.

First the polynomial system is preconditionned, so that its jacobian is the identity matrice at the center x_0 of the studied box. Let $f(x) = 0$ be the studied system, the preconditionned system is $g = Mf$ for some matrice M, so that $g'(x_0) = I_{nn}$; of course, f and g have the same roots. Since $g'(x_0) = Mf'(x_0) = I_{nn}$, it turns out that $M = f'(x_0)^{-1}$ is the inverse of the jacobian at x_0. After preconditionning, it is hoped that for each equation in $g(x) = 0$, the k th one is close to an hyperplane having equation $x_k = c_k$, where c_k is some constant. Now, each hypersurface $z = g_k(x) = 0$ lies inside the convex hull of its control points. The convex hull is a polytope in high dimension, which is not convenient. But this polytope lies inside a prism, the base of which is the 2D convex hull of the projections of the control points on the (x_k, z) plane, as examplified in Fig. 5. Computing a 2D convex hull is cheap and easy (section 3.3).

Notes: if all equations have the same degree, the Bernstein coefficients of a linear combination $\sum a_i f_i(x)$ is just the linear combination of the Bernstein coefficients of the f_is. This remark makes possible to not really compute the jacobian inverse, but to rather solve several linear systems. Finally, this contraction method also partially applies when the jacobian has not full rank, *i.e.,* is not inversible.

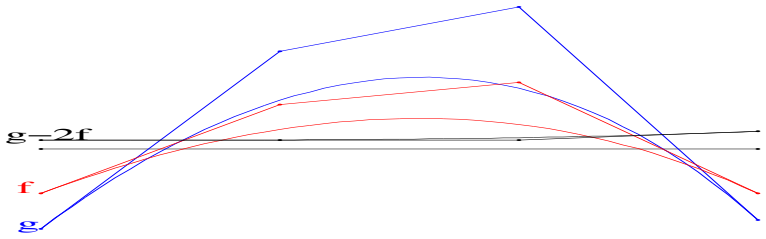

Figure 6: The problem is to find x such that $f(x) = 0$ and $g(x) \leq 0$. Since the Bernstein coefficients of (say) $g - 2f$ are all strictly positive, then $g - 2f$ is strictly positive on the considered interval, thus $f(x) = 0 \Rightarrow g(x) > 0$, and the system has no solution in the interval.

3.5 Proving a domain does not contain roots

Sometimes, the previous reduction method can not detect quickly that a box does not contain roots. For instance, with 2 cercles with the same center and close radius, the method has to subdivide all along the 2 cercles, until it separates them. The following test detects very quickly this kind of situation.

Let $f_1(x) = f_2(x) = \ldots = f_n(x) = 0$ be a system of equations. Assume wlog that all f_i have equal degrees. Then, if there are numbers $\lambda_i \in \mathbb{R}^n$ such that $g(x) = \sum \lambda_i f_i(x)$ has only positive Bernstein coefficients, then $g(x)$ is always positive in the studied domain; since g vanishes at all common roots of f_is, it proves that the f_is have no common root in the domain.

The Bernstein coefficients of g are just λ_i linear combinations of the Bernstein coefficients of the f_is, so such λ_is exist if the corresponding linear programming problem has feasible solutions.

This idea straightforwardly extends to systems of equations and inequalities. Assume the problem is to find x in some box of $\mathbb{R}^n$ such that $f_1(x) = \ldots f_e(x) = 0$ and $g_1(x) \leq 0, \ldots g_i(x) \leq 0$. If there are λ_is and $\mu_j \geq 0$ such that $\sum_i \lambda_i f_i + \sum_j \mu_j g_j > 0$ (say $\sum_i \lambda_i f_i + \sum_j \mu_j g_j \geq 1$) for all points x in the box B, then the system has no solution. Such λ_i and μ_j exist iff the corresponding linear programming problem is feasible, *i.e.,* this question reduces to linear programming. This idea is illustrated in Fig. 6.

3.6 Solving polynomial systems with reduction and bisection

To find real roots of an algebraic system inside an initial box, contract the box until it is no more possible to significantly reduce it; then try to prove it contains no root (these 2 procedures can be used in the other order). Otherwise, bisect the studied box (for instance along its longuest side) and study recursively the two halves. To quote a few, Patrikalakis and Maekawa [8], Garloff and Smith [5], Mourrain and Pavone [7] have proposed variants of this method. Mourrain

and Pavone prove that the needed stack is small, and other essential properties.

The refinement is stopped either when some accuracy is reached, or when it is no more possible to refine, due to the finite numerical accuracy which is available. To understand this last point, consider that numbers (*i.e.,* Bernstein coefficients) are represented with intervals with integer bounds (for instance, 1 means 10^{-6} meter). When an interval width is equal to one, bisecting it is no more possible: the left half (and the right half) of $[0, 1]$ is itself; actually, all intervals with widt 1 are "atomic", and can no more be bisected. The same kind of thing occurs with floating point numbers, instead of integers, due to discreteness.

3.7 Proving existence and unicity

The solver terminates with a set of small boxes it can not eliminate. There are very few spurious boxes (*i.e.,* boxes without roots), due to the optimality of Bernstein bases (actually, it is not obvious to build problems with spurious roots).

To prove that a resulting box contains a root, or contains a unique root, all tests available in interval analysis apply, of course. Moreover, their computation time is not an issue, due to the small number of spurious boxes; their computation time is an issue only for solvers which are driven by such tests.

Even if all tests apply, some fit the Bernstein scheme in a very natural way. For instance, after Miranda (or Miranda-Poincaré) theorem, if n continuous functions f_k from $\mathbb{R}^n$ to $\mathbb{R}$ are such that the k th function has constant sign on the hyperface $x_k = a_k$ of the hypercube $a_k \leq x_k \leq b_k$ and constant but opposite sign on the opposite face $x_k = b_k$, for every $k = 1, \ldots d$, then the system $f_1(x) = \ldots = f_n(x) = 0$ with $x \in \mathbb{R}^n$ has a common root inside the hypercube. See left half of Fig. 7. Now, after preconditionning, in a box containing a regular root $r = (r_1, \ldots r_d)$, the hypersurface of the k th equation is very close to the hyperplane $x_k = r_k$; the hyperfaces $x_k = a_k$ and $x_k = b_k$ are on opposite sides of the hyperplane $x_k = r_k$; the fact that f_k has constant sign on an hyperface $x_k = a_k$ or $x_k = b_k$ is proved as soon as the Bernstein coefficients on the hyperface have constant sign, and Miranda theorem then applies. This approach is smart because it is simple and uses only available data.

Elbert and Kim [2] propose an unicity test, which fits nicely with Bernstein bases (and to B-splines bases, which extend Bernstein bases to piecewise algebraic functions, and which they use): let N_i be some enclosure of the cone of normals to the function $f_i(x)$, and T_i be the cone of vectors orthogonal to vectors in N_i. Elbert and Kim enclose such cones with a central vector and an interval of angles, N_i and T_i sharing the same axial vector. When the null vector is the only vector common to all T_i, then there is at most one common root in the studied box, due to the mean value theorem. After preconditionning, when the box contains a unique regular root, this condition very likely holds (see Fig. 7). Elbert and Kim stop the reduction process as soon as existence and unicity are proved, and resort to a Newton-Raphson iteration, or some variant.

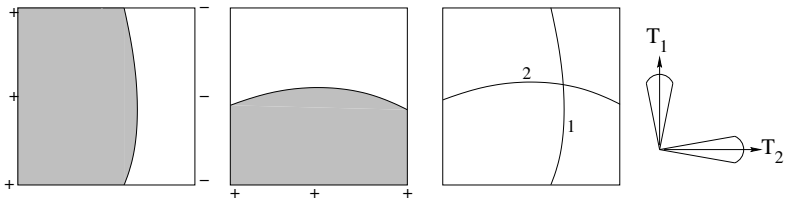

Figure 7: Left half: (Poincaré -) Miranda theorem in 2D proves existence of a root. Right half: the unicity test, tangent cones are disjoint.

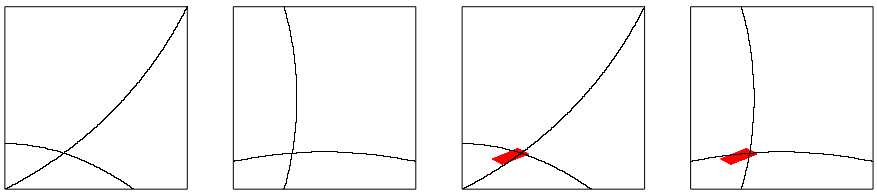

Figure 8: Left: the two curves of the system $f(x,y) = g(x,y) = 0$, inside the unit square. Middle: the two curves after preconditioning. Right 1 and 2: the image of the unit square by the secant map. $f(x,y) = (x+1)^2 + (y-2)^2 - 5, g(x,y) = y + x^2/2 - 1/4$.

This improvement may speed up the solver, though its complexity is unchanged: in this case, both the reduction process and Newton iteration have quadratic convergence.

3.8 Revisiting the secant method with the insight of Bernstein bases

To solve a polynomial system $f(x) = 0$, the interval method computes fix points of the "secant map" $s(x) = x - Mf(x)$ where the matrice M is typically the inverse (up to rounding errors) of the jacobian matrice at the centre of the studied box; the entries in M are floating point numbers (not interval). Then $s(x)$ is an algebraic map, the control points of which are computable (with interval arithmetic), and a sharp enclosure is deducible. In low dimension, it is possible to display the image of the studied box by the secant map (see Fig. 8), or to draw its control net, for pedagogical purposes; note the prevous method computes more quickly and simply the smallest box with sides parallel to axis and enclosing this image (actually it is better, because the reduction according y benefits from the reduction according x). Such control nets of algebraic maps are used routinely in computer graphics, to describe spacial deformations, morphings, animations: the graphist interactively moves the control points.

4 Miscellaneous

4.1 Bounds for derivatives

In the univariate case, if $f(x)$ has degree d and control values F_k, then the d control values of its derivative $f'(x)$ are $F'_k = (F_{k+1} - F_k)/d$, for $k = 0, \ldots d-1$. This straightforwardly extends to the multivariate case, and to derivatives of higher order.

4.2 Enclosing sums of polynomials

To enclose the sum of plynomials $p_1(x), p_2(x) \ldots p_n(x)$, the naive method encloses each polynomial p_i, then add the n enclosing intervals. Of course it is much more accurate to compute the Bernstein form of the polynomial $p_1 + p_2 + \ldots p_n$. When all polynomials have the same degree, the Bernstein coefficients of the sum is just the sum of the Bernstein coefficients. Degree elevation makes all polynomials have the same degree.

Enclosing linear combinations $\lambda_1 p_1(x) + \ldots \lambda_n p_n(x)$ where λ_i are given numbers (or sharp intervals to account for rounding errors) reduces to the previous problem, because the Bernstein coefficients of $\lambda p(x)$ are just λ times the Bernstein coefficients of $p(x)$.

4.3 Enclosing products

Here we meet one drawback of Bernstein bases: the product of two polynomials can not be expressed in the same base, usually, and elevation degree is unavoidable. It means that Bernstein polynomials can not be used for methods or problems with a non constant computation depth (*i.e.,* degree of polynomials), *e.g.,* for enclosing the determinant of a matrice with polynomial entries. We know no reference to deal with this kind of problems; note however that Bernstein bases begin to be used in computer algebra, *e.g.,* for computing gcd polynomials.

4.4 Enclosing $v(x) \,.\, u$

Let $v(x) = (v_1(x), \ldots v_n(x))$ a vector of multivariate polynomials in x. We can assume wlog that $x \in [0,1]^n$. A basic problem is to enclose the scalar product $v(x).u$ where u is a vector of intervals; it is the same problem as enclosing $v(x) \,.\, y$ where $y = (y_1, \ldots y_n)$ and y_i are new independent variables which can take any value inside v.

A first method computes the min and max $[v_i^-, v_i^+]$ of each polynomial v_i with the Bernstein bounds, then add all $[v_i^-, v_i^+] \times u_i$ using standard interval arithmetic. It is correct but terribly conservative, since polynomials v do not usually reach their minimum/maximum value at the same x. It is the famous "loss of dependence between variables" of interval analysis.

A second method considers $\sum v_i(x) \times y_i$ as a multivariate polynomial in x, y, and compute Bernstein bounds. It is easy to implement once a Bernstein package is available, very accurate, but costly.

The third method has the same accuracy but is much faster. After degree elevation, all polynomials in v have the same degrees in x. Denote V^k the vector of the Bernstein coefficients of polynomial v_k (so V_i^k is a coordinate of V^k, the coefficient of $B_i(x)$) . Then, to find an upper bound, use your favorite linear programming package[1] to solve the linear programming problem with n unknowns $u_i^- \leq y_i \leq u_i^+$, with constraints $\sum V_i^k y_k \leq \phi$ for i traversing all possible degrees, and with objective maximize ϕ. Proceed similarly for the minimum.

4.5 Inner approximation

Sometimes inner approximations of a polynomial $p(x)$ are wanted, for $x \in B$ a given box; an inner approximation of $p(x), x \in B$, is an interval $[y^-, y^+]$ such that for every of its values y there is at least one value x in B such that $p(x) = y$. Several remarks provide such inner approximations: the graph of the polynomial (the curve, the surface, etc) passes through its extremal vertices (we already mentioned that if the smaller Bernstein coefficient is at a vertex, then it gives the true mimimum of the polynomial; idem for the max); it is also possible to compute $p(x)$ for some samples in the box B, for instance to perform some subdivision: the vertices provide inner bounds; the graph and the control polyhedron can not be too far away, and several formulas give computable lower and upper bounds for these distances [1], which yield reliable inner approximations. Finally, to obtain exact (or sharp, to account for numerical inaccuracy) inner approximations, it is possible to solve polynomial system of equations.

4.6 Range of the norm of a matrice

Several algorithms of IA needs to enclose the norm of a matrice (typically a jacobian) with polynomial entries, *e.g.,* for proving the existence or unicity of a root in a given domain. Assume the norm for a vector is the max norm: the greatest absolute value of its coordinates; then the norm of a matrice M is max $||Mu||$ for $||u|| = 1$, *i.e.,* for $u = (u_1, \ldots u_n)$ and $u_i \in [-1, 1]$. Basically, a range for the scalar product $y.f$ between vector $y = (y_1 \in [-1, 1], y_2 \in [-1, 1], \ldots y_n \in [-1, 1])$ and vector $f(x) = (f_1(x), \ldots f_n(x))$ for x in a box X is needed. Methods for solving this problem are given section 4.4.

4.7 Functionnal analysis

The projection reduction does not help when the zero sets of equations are close: a typical 2D example of such a case is 2 concentric circles with very close radius. In such a case, the algoritm has to subdivide all along the two cercles, until it can

[1] In practice, we use GLPK, the GNU Linear Programming Kit, and the GSL, Gnu Scentific Library. Thanks to GNU.

separate them. It does that for nothing since there is no root at all. To prevent this behaviour, Mourrain and Pavone [7] propose to make the f_i independent, *i.e.,* orthogonal in some functional space. A scalar product between polynomials is needed, for instance a standard one is $(f.g) = \int_0^1 f(x)g(x)dx$. Assume for simplicity that f and g have the same degrees. Then

$$f.g = \int_0^1 f(x)g(x)dx = \int_0^1 (\sum_i F_i B_i(x))(\sum_j G_j B_j(x))dx = \sum_i \sum_j F_i G_j \int_0^1 B_i(x)B_j(x)dx$$

where i and j are multi indices. The scalar product between two Bernstein polynomials in the multivariate case reduces to the univariate case; for instance, in the bivariate case:

$$\begin{aligned} B_{i_1 i_2}^{d_1 d_2}(x_1,x_2) \,.\, B_{j_1 j_2}^{d_1 d_2}(x_1,x_2) &= \int_0^1 \int_0^1 B_{i_1}^{d_1}(x_1) B_{i_2}^{d_2}(x_2) B_{j_1}^{d_1}(x_1) B_{j_2}^{d_2}(x_2) dx_1 dx_2 \\ &= \left[\int_0^1 B_{i_1}^{d_1}(x_1) B_{j_1}^{d_1}(x_1) dx_1\right] \times \left[\int_0^1 B_{i_2}^{d_2}(x_2) B_{j_2}^{d_2}(x_2) dx_2\right] \\ &= (B_{i_1}^{d_1} \,.\, B_{j_1}^{d_1}) \times (B_{i_2}^{d_2} . B_{j_2}^{d_2}) \end{aligned}$$

In the univariate case, the scalar product between $B_i(x)$ and $B_j(x)$ is:

$$\begin{aligned} \int_0^1 B_i(x)B_j(x)dx &= \int_0^1 C(i,d)x^i(1-x)^{d-i}C(j,d)x^j(1-x)^{d-j}dx \\ &= C(i,d)C(j,d)\int_0^1 x^{i+j}(1-x)^{d-i-j}dx \\ &= \frac{C(i,d)C(j,d)}{C(i+j,2d)}\int_0^1 B_{i+j}^{(2d)}(x)dx = \frac{C(i,d)C(j,d)}{(2d+1)C(i+j,2d)} = R_{ij} \end{aligned}$$

using the fact that $\int_0^1 B_i^{(n)}(x)dx = 1/(n+1)$. Thus the scalar product of f and g is $f.g = (f_0, f_1, \ldots)R(g_0, g_1, \ldots)^t$ where R_{ij} defined just above. Actually, any base can be used to compute the scalar product $(f.g)$: if F and G are coordinates of f and g in this base, and if R_{ij} is the Gram matrix of the base vectors, then $(f.g) = FRG^t$.

5 Conclusion

This paper has promoted the use of tensorial Bernstein bases for interval analysis. Bernstein bases have restriction however. First, they do not account for transcendental functions. A possible solution is to bracket transcendental functions between two polynomial functions; after all, it is the way that arithmetic units compute such functions (with argument reduction, since a polynomial with prescribed degree can not approximate closely a transcendental function on a too large interval). Another possible track is to resort to Taylor expansion: the polynomial part is enclosed with Bernstein bases, and the remainder part, which

is transcendental, is enclosed with centered interval arithmetics. A second restriction of tensorial Bernstein bases is that they have an exponential numbers of elements (it also applies for the canonical base, but very often polynomials are sparse in the canonical base). The size is $(d_0+1)(d_1+1)\dots(d_{n-1}+1)$ where n is the number of variables and d_i the partial degree of the polynomial in the variable number i. A solution is to resort to another Bernstein base, the simplicial base. Though simplices may be less palatable than boxes for programmers, simplicial bases have polynomial size in the degrees and the number of variables. Simplicial bases, subdivision schemes, accounting for transcendentals are very active research topics in computer geometry.

References

[1] Carl de Boor. B-form basics. *Geometric Modeling: Algorithms and New Trends*, pages 131–148, 1987.

[2] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 1–10, New York, NY, USA, 2001. ACM Press.

[3] Gerald Farin. *Curves and Surfaces for Cagd: A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, 1988.

[4] R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in bernstein form. *Comput. Aided Geom. Des.*, 4(3):191–216, 1987.

[5] Jürgen Garloff and Andrew P. Smith. Solution of systems of polynomial equation by using bernstein expansion. In *Symbolic Algebraic Methods and Verification Methods*, pages 87–97. Springer, 2001.

[6] R. Martin, H. Shou, I. Voiculescu, A. Bowyer, and G. Wang. Comparison of interval methods for plotting algebraic curves. *Computer Aided Geometric Design*, 19(7):553–587, 2002.

[7] Bernard Mourrain and Jean-Pascal Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA, August 2005. RR-5658.

[8] Nicholas M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Verlag, 2002.