Solving nonlinear polynomial systems in the barycentric Bernstein basis

Citation: Reuter, Martin et al. "Solving Nonlinear Polynomial Systems in the Barycentric Bernstein Basis." The Visual Computer 24.3 (2007) : 187-200. © 2007 Springer-Verlag

As Published: http://dx.doi.org/10.1007/s00371-007-0184-x

Publisher: Spring Berlin/Heidelberg

Persistent URL: http://hdl.handle.net/1721.1/65558

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

The Visual Computer manuscript No.
(will be inserted by the editor)

Martin Reuter · Tarjei S. Mikkelsen · Evan C. Sherbrooke · Takashi Maekawa · Nicholas M. Patrikalakis

Solving Nonlinear Polynomial Systems in the Barycentric Bernstein Basis

submitted: February 23, 2007; revised: September 5, 2007

Abstract We present a method for solving arbitrary systems of N nonlinear polynomials in n variables over an n-dimensional simplicial domain based on polynomial representation in the barycentric Bernstein basis and subdivision. The roots are approximated to arbitrary precision by iteratively constructing a series of smaller bounding simplices. We use geometric subdivision to isolate multiple roots within a simplex. An algorithm implementing this method in rounded interval arithmetic is described and analyzed. We find that when the total order of polynomials is close to the maximum order of each variable, an iteration of this solver algorithm is asymptotically more efficient than the corresponding step in a similar algorithm which relies on polynomial representation in the tensor product Bernstein basis. We also discuss various implementation issues and identify topics for further study.

Keywords CAD, CAGD, CAM, geometric modeling, solid modeling, intersections, distance computation, engineering design

1 Introduction

A fundamental problem in computer-aided design and manufacturing is the efficient computation of all solutions of a system of nonlinear polynomial equations within some finite domain. This rather difficult task can be very unstable and prone to errors even in the univariate case. Lane and Riesenfeld [14] first approached the computation of roots of univariate functions by using robust subdivision techniques, employing Bernstein-Bézier basis functions.

M. Reuter · T. Mikkelsen · E. Sherbrooke · N. Patrikalakis
Massachusetts Institute of Technology,
Cambridge, MA 02139-4307, USA
E-mail: reuter@mit.edu

T. Maekawa
Yokohama National University,
79-5 Tokiwadai, Hodogaya, Yokohama 240-8501, Japan

Sherbrooke and Patrikalakis [18] have developed a method (called the interval projected polyhedron algorithm, or IPP) for solving such systems in higher dimensions within an n-dimensional rectangular domain, relying on the representation of polynomials in the tensor product Bernstein basis. Their method, as the method presented in this paper, is a reduction approach that contracts the domain containing a solution. If a reduction is not effective, a subdivision step is necessary, for example to separate roots within a domain. A similar approach has been applied to more general B-spline representations by Elber and Kim [3], who rely only on subdivision but manage to eliminate redundant subdivisions at the final stage by switching to Newton-Raphson iterations. Mourrain and Pavone [15] improve upon the interval projected polyhedron algorithm by using efficient univariate solvers and a preconditioning step to optimize the reduction steps. Their experiments show, that reduction methods can save a lot of subdivision steps and usually outperform plain subdivision methods. Recent advances by Hanniel and Elber [11] present termination criteria to ensure that exactly one solution is contained within a region. After such regions are identified, it is possible to switch to other methods that quickly converge to the root.

In this paper we introduce a method for solving systems within an **n-dimensional simplex**, which relies on the representation of polynomials in the **barycentric Bernstein basis**. This approach is independent of the coordinate directions of the original system. In order to isolate all of the roots within a given simplex we construct a series of bounding simplices by intersecting the 2D convex hulls of the control points of bounding curves for every coordinate. We will present and analyze an algorithm implementing this method in rounded interval arithmetic also focusing on convergence and complexity analysis issues. We note that a similar technique for the $n = 2$ special case has been developed by Sederberg [17], although without detailed analysis.

The barycentric Bernstein basis of degree m has the advantage to represent polynomials with total order m

without any overhead, when the maximum order of every variable is equal to m. The number of control points matches exactly the degree of freedom in any dimension. The tensor product Bernstein basis used in [18] with degree m in every dimension on the contrary has more control points than the degree of freedom of a polynomial of total order m. It produces much overhead with problems where the maximum order of a single variable is close to the total order. On the other hand, it is well suited for problems where no single variable has an order higher than m/n (with n being the number of variables) and where the total order is reached in a place where the exponents of all variables are added.

For example the two dimensional ($n = 2$) polynomial

$$p_1(x, y) = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 \quad (1)$$

of total order $m = 2$ can be represented over a triangular domain in the barycentric Bernstein basis of degree 2 with exactly 6 control points. Since the maximal order of every variable is 2 the tensor product Bernstein basis has to use $3\times3 = 9$ control points. However, a polynomial of the form

$$p_2(x, y) = a_1x^2y^2 + a_2x^2y + a_3xy^2 + a_4x^2 + a_5xy + a_6y^2 + a_7x + a_8y + a_9 \quad (2)$$

of total order $m = 4$ could be represented in the tensor product Bernstein basis with the same effort while in barycentric Bernstein basis it requires 15 control points. Nevertheless these 15 control points are enough to represent all kinds of two dimensional polynomials with total order $m = 4$ while in the tensor product Bernstein basis we even need 25 control points to do that. In higher dimensions this discrepancy between the two approaches is even larger (see Table 1). Therefore it makes sense to consider each of the different methods in cases where they are best suited.

This paper is organized in the following manner: Section 2 introduces the notation we will use, and contains a brief review of the properties of the barycentric Bernstein polynomials. Section 3 gives the formulation of the problem and describes the construction of smaller bounding simplices. Section 4 describes an algorithm which can be used in implementing a solver based on this method. Section 5 discusses various implementation issues with an analysis and Section 6 provides some numerical examples. Section 7 identifies some topics recommended for further study.

2 Notation and Definitions

1. Lower-case letters in boldface, such as $\mathbf{x}$ and $\mathbf{y}$ will denote vectors of real numbers, or points. It should be clear from the context what the dimension of any given vector is. When we work with the components of the vector $\mathbf{x}$, we will assume that x_i is the i-th component of $\mathbf{x}$.
2. Upper-case boldface letters such as $\mathbf{F}$ denote vector-valued functions, sets or matrices. The i-th component of $\mathbf{F}$ is denoted by F_i.
3. The upper-case letter I denotes a multi-index (defined below).
4. To avoid confusion, some variables may have superscripts instead of or in addition to subscripts. For example, the matrix $W^{(k)}$ is superscripted so that we can refer to the element in the ith row, jth column as $w_{ij}^{(k)}$, rather than as the ambiguous w_{ijk}.
5. $\mathbf{R^n}$ is the set of ordered n-tuples of real numbers. $\mathbf{R^{n+}}$ is the set of ordered n-tuples of non-negative real numbers. $\mathbf{R^{\geq n}}$ is the set of ordered m-tuples of real numbers with a fixed $m \geq n$.

Definition 1 *Multi-index* I is an ordered $n+1$-tuple of nonnegative integers $(i_1, \ldots, i_{n+1})$. I is *bounded* by the positive integer m if $i_1 + i_2 + \ldots + i_{n+1} = m$.

We will use the notation $\sum_I^m$, where m is a bounding integer, to indicate that a sum is to be taken over all possible multi-indices I which are bounded by m. For example, if $m = n = 2$, then $\sum_I^m w_I \equiv w_{002} + w_{011} + w_{020} + w_{101} + w_{110} + w_{200}$. Multi-indices are used as a shorthand for subscripting multi-dimensional triangular arrays. For example, if $I = (0, 1, 1)$, then w_I is equivalent to w_{011}, an element of the triangular matrix $\{w_{ijk}\}$. This becomes clear by viewing $i + j + k = 2$ as $i + j \leq 2$:

$$\begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & \\ w_{20} & & \end{pmatrix}.$$

Definition 2 The convex hull of the set of $n+1$ points $\mathbf{S} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\}$, denoted by $conv(\mathbf{S})$, is the set of all points such that for any vector $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$:

$$\sum_{i=1}^{n+1} x_i\mathbf{p}_i \in conv(\mathbf{S}). \quad (3)$$

Definition 3 For a set $\mathbf{S} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\}$ where $\mathbf{p}_i \in \mathbf{R}^{\geq n}$ for $i \in \{1, \ldots, n+1\}$, $conv(\mathbf{S})$ is called an n-dimensional *simplex* or *n-simplex*.

For example, for $n = 1, 2, 3$ the simplex reduces to a straight line segment, a triangle and a tetrahedron, respectively. The $n+1$ vertices of the n-simplex have to be embedded at least in $\mathbf{R}^n$ or in a higher dimensional space.

Lemma 1 *An n-simplex has $\frac{(n+1)n}{2}$ edges connecting every vertex with every other vertex.*

Definition 4 The I-th n-dimensional *barycentric Bernstein polynomial* [4] of total degree m is defined by

$$B_{I,m}^n(\mathbf{x}) := \frac{m!}{I!}\mathbf{x}^I \quad (4)$$

where $I! := i_1! \ldots i_{n+1}!$ and $\mathbf{x}^I := x_1^{i_1} \ldots x_{n+1}^{i_{n+1}}$, for $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$ and I bounded by m.

Lemma 2 *The barycentric Bernstein polynomials of total degree m and dimension n form a* partition of unity *[4, 12]:*

$$\sum_{I}^{m} B_{I,m}^{n}(\mathbf{x}) = 1 \tag{5}$$

for any $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$.

Lemma 3 *The sum of barycentric Bernstein polynomials of total degree m and dimension n with index $i_j = t$ is simply the t-th univariate Bernstein polynomial in the j-th coordinate:*

$$\sum_{I|i_j=t}^{m} B_{I,m}^{n}(\mathbf{x}) = (x_j)^t (1-x_j)^{m-t} \frac{m!}{t!(m-t)!} = B_{t,m}^{1}(x_j) \tag{6}$$

where x_j is the j-th element of $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$*, and where i_j (the j-th element of the multi-index I) is restricted to t.*

Proof We set $\mathbf{x}' := \left(\frac{x_1}{1-x_j}, \frac{x_2}{1-x_j}, ..., \frac{x_{j-1}}{1-x_j}, \frac{x_{j+1}}{1-x_j}, ..., \frac{x_{n+1}}{1-x_j}\right)$ by removing x_j and adjusting the other coefficients so that $\sum_{i=1}^{n+1} x'_i = 1$. Also set $I' := (i_1 ... i_{j-1} i_{j+1} ... i_{n+1})$ bounded by $(m-t)$. Then

$$\begin{aligned}
\sum_{I|i_j=t}^{m} B_{I,m}^{n}(\mathbf{x}) &= \sum_{I|i_j=t}^{m} x_1^{i_1} ... x_j^t ... x_{n+1}^{i_{n+1}} \frac{m!}{i_1!...t!...i_{n+1}!} \\
&= x_j^t \sum_{I'}^{m-t} \mathbf{x}'^{I'} (1-x_j)^{m-t} \frac{m!(m-t)!}{I'!t!(m-t)!} \\
&= x_j^t (1-x_j)^{m-t} \frac{m!}{t!(m-t)!} \sum_{I'}^{m-t} \mathbf{x}'^{I'} \frac{(m-t)!}{I'!} \\
&= x_j^t (1-x_j)^{m-t} \frac{m!}{t!(m-t)!}
\end{aligned}$$

using Lemma 2 in the last step. □

Lemma 4 *The barycentric Bernstein polynomials of total degree m and dimension n satisfy the* linear precision property *[4, 12]:*

$$x_j = \sum_{I}^{m} \frac{i_j}{m} B_{I,m}^{n}(\mathbf{x}) \tag{7}$$

where x_j is the j-th element of $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$*, and i_j is the j-th element of the multi-index I.*

For a more complete treatment of the barycentric Bernstein polynomials, refer to [4], [6] and [12].

3 Formulation

3.1 Barycentric Coordinates

Let there be a set of N functions $f_1, f_2, \ldots, f_N$, each of which is a polynomial in the n independent parameters $u_1, u_2, \ldots, u_n$. Let $m^{(k)}$ denote the total degree of f_k. Furthermore, let $\mathbf{S}$ denote a set of points $\{\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\}$ where $\mathbf{p}_i \in \mathbf{R}^n$. Our objective is to find all points $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in conv(\mathbf{S})$ such that

$$f_1(\mathbf{u}) = f_2(\mathbf{u}) = \ldots = f_N(\mathbf{u}) = 0. \tag{8}$$

Each f_k can be converted to the barycentric Bernstein basis with respect to $\mathbf{S}$ using the algorithm referred to in Section 5.1.2. This algorithm gives the unique $n+1$-dimensional triangular array of real coefficients $\{w_{I,\mathbf{S}}^{(k)} \mid I$ bounded by $m^{(k)}\}$ such that

$$f_k(\mathbf{u}) = f_{k,\mathbf{S}}(\mathbf{x}) = \sum_{I}^{m^{(k)}} w_{I,\mathbf{S}}^{(k)} B_{I,m^{(k)}}^{n}(\mathbf{x}) \tag{9}$$

$$\text{for } \mathbf{u} = \sum_{i=1}^{n+1} x_i \mathbf{p}_i \tag{10}$$

where $\mathbf{p}_i$ is the i-th vertex of $\mathbf{S}$ and x_i is the i-th element of $\mathbf{x}$.

Our objective now becomes to find all points $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$, such that

$$f_{1,\mathbf{S}}(\mathbf{x}) = f_{2,\mathbf{S}}(\mathbf{x}) = \ldots = f_{N,\mathbf{S}}(\mathbf{x}) = 0. \tag{11}$$

From Eq. 10 we see that if we can find every $\mathbf{x}$ that satisfies Eq. 11, we can find every root of the original problem.

3.2 Coordinate Bounding Functions

Let us define the following univariate functions:

Definition 5 For any $f(\mathbf{x})$ as in Eq. 9 and coordinate $j = 1, ..., n+1$

$$min_j(x_j) := \sum_{t=0}^{m} \left(\min_{I|i_j=t} w_I \right) B_{t,m}^{1}(x_j)$$

$$max_j(x_j) := \sum_{t=0}^{m} \left(\max_{I|i_j=t} w_I \right) B_{t,m}^{1}(x_j)$$

Then we have the following property:

Theorem 1 *For any* $\mathbf{x} = (x_1, ..., x_{n+1}) \in \mathbf{R}^{(n+1)+}$ *with* $\sum_{i=1}^{n+1} x_i = 1$ *and any* $j = 1, ..., n+1$ *we have*

$$min_j(x_j) \leq f(\mathbf{x}) \leq max_j(x_j).$$

Proof

$$f(\mathbf{x}) = \sum_{I}^{m} w_I B^n_{I,m}(\mathbf{x}) \leq \sum_{t=0}^{m} \left(\max_{I|i_j=t} w_I \right) \sum_{I|i_j=t}^{m} B^n_{I,m}(\mathbf{x})$$
$$= \sum_{t=0}^{m} \left(\max_{I|i_j=t} w_I \right) B^1_{t,m}(x_j) = max_j(x_j)$$

where we used Lemma 3. For $min_j(x_j)$ simply use $\geq$ and take the min. □

This result gives an upper and lower bound of any $f(\mathbf{x})$ for each coordinate j. This can help narrowing the interval containing a root:

Corollary 1 *For any root* $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$ *of* $f(\mathbf{x})$ *we can find* $\lambda_j \leq x_j \leq \mu_j$ *for every coordinate* j *with*

$min_j(x_j) \leq 0 \leq max_j(x_j)$ *in* $x_j \in [\lambda_j, \mu_j]$

where

1. λ_j *and/or* μ_j *is a root of any of the two functions* $min_j(x_j)$ *or* $max_j(x_j)$,
2. *or* λ_j *(resp.* μ_j*) is equal to* 0 *(resp.* 1*) if it is no root.*

Mourrain et al. [15] constructed bounding curves and used similar bounds in the tensor product case to reduce the domain. In this work we use the simple convex hull algorithm as explained in the next section. Obtaining closer approximations of λ_j and μ_j would improve the shrinking process.

3.3 Graphs

In order to find the roots by the subdivision method, we will restate the problem as the intersection of curves with the horizontal axis. Because of Theorem 1 we can work with a single coordinate at a time and, therefore, onlywith univariate intersection problems. We can simply construct the graphs of the functions $min_j^{(k)}(x_j)$ and $max_j^{(k)}(x_j)$ for every f_k from Eq. 9 and for every coordinate j:

$$\mathbf{min}'^{(k)}_j(x_j) := (x_j, min_j^{(k)}(x_j)), \tag{12}$$

$$\mathbf{max}'^{(k)}_j(x_j) := (x_j, max_j^{(k)}(x_j)) \tag{13}$$

Employing the linear precision property Eq. 7 yields

$$\mathbf{min}'^{(k)}_j(x_j) = \sum_{t}^{m^{(k)}} \mathbf{v}^{(k)}_{j,t} B^1_{t,m^{(k)}}(x_j) \tag{14}$$

$$\mathbf{max}'^{(k)}_j(x_j) = \sum_{t}^{m^{(k)}} \mathbf{w}^{(k)}_{j,t} B^1_{t,m^{(k)}}(x_j) \tag{15}$$

where $t = 0, ..., m^{(k)}$ and

$$\mathbf{v}^{(k)}_{j,t} := \left(\frac{t}{m^{(k)}}, \min_{I|i_j=t} w^{(k)}_I \right) \tag{16}$$

$$\mathbf{w}^{(k)}_{j,t} := \left(\frac{t}{m^{(k)}}, \max_{I|i_j=t} w^{(k)}_I \right) \tag{17}$$

define the 2D control points to their curves. We denote the set of control points with $\mathbf{V}^k_j$ and $\mathbf{W}^k_j$ respectively. With this in mind, we can now approach the problem by considering the well known convex hull property:

Lemma 5 *Let* $A := \{\mathbf{a}_t \mid t = 0, ..., m\}$ *be the set of control points of a Bézier curve* α*. Let* $conv(A)$ *be the convex hull of this set. Then for* $x \in [0, 1]$

$\alpha(x) \in conv(A)$.

Therefore we can now state the following

Theorem 2 *For any common root* $\mathbf{x}$ *of the system* $f_k(\mathbf{x})$ *with* $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$ *and for every coordinate* $j = 1, ..., n+1$*:*

$$(x_j, 0) \in \bigcap_{k=1}^{N} conv(\mathbf{V}^k_j \cup \mathbf{W}^k_j).$$

Proof From Corollary 1 it follows that $(x_j, 0)$ has to lie between $(\lambda_j, 0)$ and $(\mu_j, 0)$ for any k. Furthermore, $(\lambda_j, 0)$ (resp. $(\mu_j, 0)$) lies either on one of the functions $\mathbf{min}'^{(k)}_j$ or $\mathbf{max}'^{(k)}_j$ (case 1) or between them (case 2) since $min_j^{(k)}(x_j) \leq 0 \leq max_j^{(k)}(x_j)$ for all $x_j \in [\lambda_j, \mu_j]$ (especially for λ_j and μ_j). Therefore $(\lambda_j, 0)$, $(\mu_j, 0)$ and finally $(x_j, 0) \in conv(\mathbf{V}^k_j \cup \mathbf{W}^k_j)$. Since $\mathbf{x}$ is a common root, this has to be true for every $k = 1, ..., N$. Therefore $(x_j, 0) \in \bigcap_{k=1}^{N} conv(\mathbf{V}^k_j \cup \mathbf{W}^k_j)$. □

This also means that all points not contained in the intersection $(x_j, 0) \notin \bigcap_{k=1}^{N} conv(\mathbf{V}^k_j \cup \mathbf{W}^k_j)$ cannot be a common root and can be discarded. By intersecting $conv(\mathbf{V}^k_j \cup \mathbf{W}^k_j)$ first with the horizontal axis (for a single k) and then with the intervals of the other functions k, we can efficiently narrow down the region containing possible roots for every coordinate j. Thus we obtain a lower and upper bound (λ'_j, μ'_j) for every j that are not quite as tight as the bounds in Corollary 1, but very easy to compute.

3.4 Generating Bounding Simplices

By applying Theorem 2 to every coordinate $j = 1, ..., n+1$ we obtain a pair (λ'_j, μ'_j) for every j such that $\lambda'_j \leq x_j \leq \mu'_j$. Let us focus on a lower bound λ'_j. Because we deal with barycentric coordinates $\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\}$ this bound affects all of the n other parameters of $\mathbf{x}$, such that for every $l \in \{1, \ldots, n+1 \mid l \neq j\}$, we have

$$0 \leq x_l \leq 1 - \lambda'_j \tag{18}$$

Accumulating the effects of these bounds gives a set of bounds where for every $j \in \{1, 2, \ldots, n+1\}$:

$$\lambda'_j \leq x_j \leq 1 - \sum_{1 \leq i \leq n+1, i \neq j} \lambda'_i =: \nu_j \tag{19}$$

By defining the subset $\mathbf{S}' \subseteq \mathbf{S}$ as the set of all

$$\mathbf{x} \in \{\mathbf{R}^{(n+1)+} \mid \sum_{i=1}^{n+1} x_i = 1\} \tag{20}$$

which satisfy Eq. 19 for every j, we have guaranteed that $\mathbf{S}'$ is a simplex. By using the upper bound μ'_j instead of ν_j a smaller solution space can be found, but this will in general not be a simplex and will therefore complicate an implementation of this method. The vertices of $\mathbf{S}'$ can be found from the following formula:

$$p_{ij} = \begin{cases} \nu_j \text{ if } j = i, \\ \lambda'_j \text{ if } j \neq i, \end{cases} \tag{21}$$

where p_{ij} is the j-th coordinate of the i-th vertex of $\mathbf{S}'$.

Even though this method is written down a little differently, it is actually the root finding approach developed by Sherbrooke and Patrikalakis [18] generalized to the barycentric Bernstein form. They describe the projection of the high dimensional control polyhedra onto the plane (for every coordinate) followed by the computation of the convex hull and intersection with the horizontal axis. In this work we look at the single coordinate right from the start, eliminating the description of the high dimensional control polyhedra. It should also be noted, that generally the intersection of the convex hulls of high dimensional point sets yields a much smaller result than intersecting their projections, but is far more difficult to compute.

3.5 Isolating Roots

If there is more than one root present in the simplex $\mathbf{S}$, the procedure described in Section 3.4 will fail, because the set $\mathbf{S}'$ might stop shrinking, before it becomes small enough. We must therefore formulate a complementary procedure for subdividing $\mathbf{S}'$ in order to isolate the roots. The choice of a subdivision scheme is critical as it will affect both the efficiency and correctness of the solver algorithm. Sederberg [17] subdivides the search domain into a regular tiling of triangles, but this strategy does not generalize well to arbitrary dimensions. Instead, we recursively split $\mathbf{S}$ at the midpoint of the longest edge that has not decreased sufficiently with respect to some tolerance. Note that splitting one edge affects the length of other edges. This approach is similar to the method used by Sherbrooke and Patrikalakis [18], but analysis is complicated by the fact that there is no simple, direct relationship between the edges of $\mathbf{S}$ and the parameters of $\mathbf{x}$.

4 Algorithm for Solving Nonlinear Polynomial Systems

Now we will describe an algorithm, based on our method, for solving systems of nonlinear polynomials over an n-dimensional simplicial domain. In order to ensure the numerical robustness we used rounded interval arithmetic to convert the polynomials into the Bernstein base and to solve the systems. In very ill-conditioned examples we used rational arithmetic for the conversion. A discussion of these and other relevant implementation issues can be found in Section 5. Let N be the number of equations, and n is the dimension of the system (i.e., the number of scalar variables of $\mathbf{u}$). The set $\{f_1(\mathbf{u}), \ldots, f_N(\mathbf{u})\}$ is the polynomials we wish to solve.

Furthermore, some *tolerances* are needed. The tolerance EPS is simply whatever accuracy the root is required to satisfy. To decide whether a region is decreasing in size too slowly we use a threshold which we will call CRIT. Our experiments suggest that 0.7 is a reasonable value for this threshold. We will always assume that 1 >CRIT> 0.5. The presented algorithm is valid for under-constrained, balanced and over-constrained systems in arbitrary dimensions.

A1 [Initialization] Create a *problem stack* for storing simplices and corresponding N polynomials. Push the initial bounding simplex $\mathbf{S}$ and the initial polynomials onto the stack. Create a *solution stack* for storing solution simplices.

A2 [New problem] If the *problem stack* is empty, return the *solution stack*. If not, pop the topmost simplex and corresponding polynomials from the *problem stack* and bind them as *current simplex* and *current system*.

A3 [Intersection I] For each of the $n+1$ variables j of the *current system*, find the control points $\mathbf{V}_j^k \cup \mathbf{W}_j^k$ for $k = 1, .., N$ and intersect their convex hulls with the horizontal axis, as described in Section 3.4. If one or more of the convex hulls do not intersect with the horizontal axis, conclude that there is no root in the *current simplex* and jump to A2.

A4 [Intersection II] For every one of the $n+1$ variables of the *current system*, intersect the N intervals found in step A3, and keep the lowest bound of their intersection as the *lower bound* (λ'_j) for that variable. If one or more of the bounds do not intersect, conclude that there is no root in the *current simplex* and jump to A2.

A5 [Shrink simplex] Use the *lower bounds*(λ'_j) from step A4 to construct a *smaller simplex* ($\mathbf{S}'$) from the current simplex.

A6 [Accuracy] If every edge in the *smaller simplex* ($\mathbf{S}'$) is small enough (less than EPS), push it onto the *solution stack* and go back to A2.

A7 [Extraction] Construct a *new system* from the *current system* according to the *smaller simplex* ($\mathbf{S}'$) applying $(n+1)$ deCasteljau subdivisions with the *lower bounds* found in A5 to each of the N polynomials.

A8 [Tolerance] If every edge in the *smaller simplex* is less than CRIT times the corresponding edge in the *current simplex*, it shrinks fast enough. Then push the *smaller simplex* and the *new system* onto the *problem stack* and go back to A2.

A9 [Subdivision] By reaching this point, we know that the *current simplex* is not shrinking fast enough. Perform a binary split at the midpoint of the longest edge of the *smaller simplex*. Compute two *new systems* by applying one deCasteljau subdivision at the split point. Push both sub-simplices with the corresponding new system onto the *problem stack* and go to A2.

END

5 Implementation and Analysis

5.1 Methods

Here we will describe various functions and features needed to implement the solver algorithm in Section 4.

5.1.1 Rounded Interval Arithmetic

To ensure numerical robustness the solver algorithm was implemented using rounded interval arithmetic (RIA) as described in [1] and [13]. Rounded interval arithmetic increase the running time of the algorithm by a factor of 2 to 5, but the method will frequently fail when implemented with standard floating point operations.

5.1.2 Converting Polynomials to the Barycentric Bernstein Basis

For the purpose of converting polynomials in the power basis to the barycentric Bernstein basis, we used an algorithm by Waggenspack and Anderson [19]. This algorithm is robust and reasonably efficient. It is based on regrouping and expansion of terms using Horner's rule and handles degeneracies well. In addition to the rounded interval arithmetic, we implemented exact rational arithmetic especially for the conversion of high order problems, where huge numbers lead to inexact interval results. Problems formulated in the barycentric Bernstein basis from the very beginning can of course skip the conversion step.

5.1.3 Binary Subdivision

The polynomials control points with respect to a subdivided simplex are found by a deCasteljau subdivision as described in [4], [5] and [7]. Since the simplex is split at the midpoint of its longest edge, the deCasteljau algorithm can be executed faster than for an arbitrary point in the interior of the simplex. Every deCasteljau point can be computed by two multiplications instead of $n+1$. Fig. 1 depicts the top view of the subnets in case of a triangle (degree two) for an arbitrary point as well as for an edge split. It can be seen that for the interior point (top) the deCasteljau points in each step are a barycentric combination off all the vertices of the sub-simplices. In case of the edge split (bottom) every deCasteljau point is simply a barycentric combination of the two edge vertices.

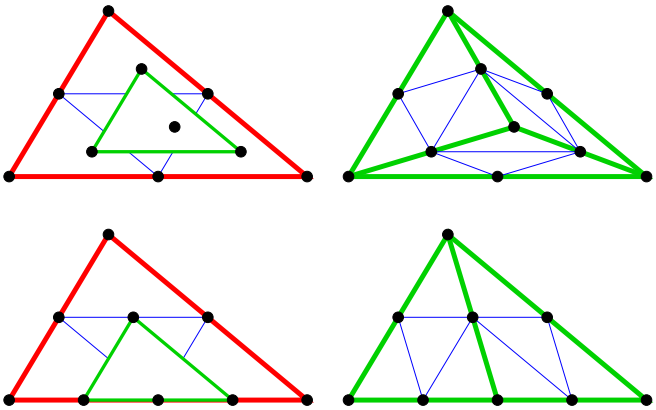

Fig. 1 Split a triangle (degree two) at an interior point (top) and along its longest edge (bottom). Left: the deCasteljau steps in both cases. Right: the subdivided triangles

We will now compute the computational steps needed for a binary subdivision. Please refer to [10] for complete treatment and proofs of the binomial identities used.

Lemma 6 *The number of computational steps needed to perform one deCasteljau subdivision of an n-dimensional barycentric Bernstein polynomial of total degree m along a single edge of the corresponding simplex is $O\left(\frac{(m+n)!}{(m-1)!(n+1)!}\right)$.*

Proof The number of coefficients describing an n-dimensional barycentric Bernstein polynomial of total degree m is [6]:

$$C_B(m,n) := \binom{m+n}{m} = \frac{(m+n)!}{m!n!}. \tag{22}$$

In one subdivision of an n-dimensional barycentric Bernstein polynomial of total degree m, the coefficients of m polynomials of degrees $m-1$ to 0 must be calculated. Therefore the number of coefficients calculated is

$$D_B(m,n) := \sum_{t=0}^{m-1} C_B(t,n) = \frac{(m+n)!}{(m-1)!(n+1)!}. \tag{23}$$

Generally it takes $(n+1)$ multiplications to calculate the value of a single coefficient, but in our case all deCasteljau subdivisions can be performed along a single edge of the simplex; therefore, we only need two multiplications. Thus only

$$\frac{2(m+n)!}{(m-1)!(n+1)!} \tag{24}$$

multiplications are needed to complete one subdivision step. No other procedure adds more than a constant multiple to this number of steps. Hence Lemma 6 follows. □

5.1.4 Extraction

The polynomials control points with respect to a given simplex are found by repeated applications of the de-Casteljau algorithm. In order to extract the coefficients on the smaller simplex that was constructed from the lower bounds it is sufficient to apply $(n+1)$ full de-Casteljau subdivisions (one for each vertex) and adjust the coordinates of the remaining points in between. Fig. 2 shows the 2D case of a triangle containing a sub-triangle (left) and the first deCasteljau subdivision (right). After deCasteljau we can discard two of the three resulting smaller triangles. To continue the coordinates of the remaining vertices of the sub-triangle need to be computed with respect to the smaller triangle obtained from deCasteljau.

Lemma 7 *Let $x_{i,j}$ be the j-th barycentric coordinate of vertex i of the final sub-simplex (with respect to the original simplex $\mathbf{S}$). By applying one full deCasteljau computation to split the simplex at the k-th vertex of the final sub-simplex we obtain a smaller simplex $\mathbf{S}'$ (containing the final simplex, where the k-th vertex is already at its final location). The new coordinates $x'_{i,j}$ of the remaining vertices of the final sub-simplex with respect to $\mathbf{S}'$ are*

$$x'_{i,j} = \begin{cases} \frac{x_{i,k}}{x_{k,k}} & \text{for } j = k \\ x_{i,j} - \frac{x_{i,k}}{x_{k,k}} x_{k,j} & \text{for } j \neq k \end{cases} \tag{25}$$

Proof By working on vertex k, we can easily compute the k-th coordinate of the remaining vertices $x'_{i,k} := \frac{x_{i,k}}{x_{k,k}}$. This is due to the fact that the face opposing the vertex k is not changed by the deCasteljau subdivision (as can be seen in Fig. 2 for one of the edges). So for the k-th coordinate, we simply scale the parameter space accordingly. The other coordinates $(j \neq k)$ can be computed in the following manner. The (global) position of every vertex $\mathbf{x}_i$ of the final sub-simplex can be computed by the barycentric combination (see Definition 2 and 3) $\mathbf{x}_i = \sum_j x_{i,j} p_j$, where p_j describes the vertices of the original simplex $\mathbf{S}$. These positions have to remain the same after representing every vertex with barycentric coordinates inside the sub-simplex $\mathbf{S}'$ obtained through de-Casteljau (note that $\mathbf{S}'$ keeps all vertices except for vertex k, which is now at its final position $\mathbf{x}_k$):

$$\begin{aligned} \sum_j x_{i,j} p_j &= \sum_{j \neq k} x'_{i,j} p_j + x'_{i,k} \mathbf{x}_k \\ &= \sum_{j \neq k} x'_{i,j} p_j + \frac{x_{i,k}}{x_{k,k}} \left(\sum_j x_{k,j} p_j \right) \\ &= \sum_{j \neq k} \left(x'_{i,j} + \frac{x_{i,k}}{x_{k,k}} x_{k,j} \right) p_j \; + x_{i,k} p_k \\ \Rightarrow x'_{i,j} &= x_{i,j} - \frac{x_{i,k}}{x_{k,k}} x_{k,j} \end{aligned} \tag{26}$$

□

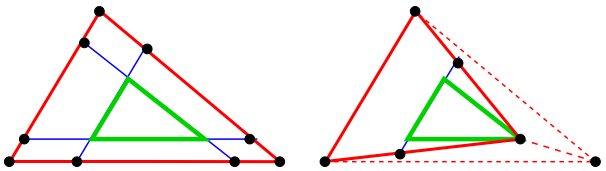

Fig. 2 Extraction using interior deCasteljau splits

Lemma 8 *The number of computational steps needed to perform one extraction of an n-dimensional barycentric Bernstein polynomial of total degree m on a smaller simplex is $O\left(\frac{(m+n)!}{(m-1)!(n-1)!}\right)$.*

Proof To extract an n-dimensional barycentric Bernstein polynomial of total degree m on a smaller simplex $n+1$ subdivisions are performed (at each of its $n+1$ vertices). Therefore we have to compute

$$(n+1) D_B(m,n) = \frac{(m+n)!}{(m-1)! n!} \tag{27}$$

coefficients. For every coefficient we need $(n+1)$ multiplications (since we split at an inner vertex as opposed to the edge splits from the last section). The computation of the new vertex coordinates with respect to the new simplex adds only $\frac{n(n+1)^2}{2}$ multiplications in total. Therefore we have the costs:

$$O\left(\frac{(m+n)!(n+1)}{(m-1)! n!} \right) = O\left(\frac{(m+n)!}{(m-1)!(n-1)!} \right) \tag{28}$$

Hence Lemma 8 follows. □

It is also possible to use blossoms to describe this extraction process (see [9] for background). Straight forward blossoming, where every coefficient is computed by a full deCasteljau subdivision, is very slow, since many identical values are computed over and over again. Even the improved approach suggested by Sablonnière [16] that reuses existing common values is only faster than

our approach for linear problems. We implemented it for arbitrary dimensions and it needs

$$(n+1)\left(\binom{m+2n+1}{m} - C_B(m,n)\right) \tag{29}$$

multiplications. The advantage of Sablonnière's algorithm is that the final coefficients are computed directly form the original coefficients, not from intermediate values as described here. Therefore his approach is very stable (which comes at a cost). Since we deal with Bézier representations; therefore with convex combinations only, the additional robustness is not really necessary here. The algorithm presented here can be seen as the further improved approach by Goldman [8] applied in higher dimensions. Goldman first described it for knot insertion into B-spline curves. He represents the deCasteljau algorithm by an $n+1$ simplex (in his case a triangle). After the first full deCasteljau computation, he restarts it a second and final time along one of its edges (with adjusted coordinates). We basically do the same for an arbitrary high dimensional simplex.

5.2 Analysis

In this section, we will prove the correctness of the algorithm. We will also analyze the time complexity of one loop of the algorithm.

5.2.1 Proof of Termination

Let us first prove that the algorithm terminates. In order to do this, we first define a few terms relating to the algorithm:

Definition 6

1. An *iteration* or *step* is one execution of steps A2 through A9.
2. The *children* of a bounding simplex **S** are the simplices we are left with after step A9, and **S** is said to be the *parent* of these children. If we determine in step A6 that we are sufficiently close to a root, then **S** will have no children. Typically, **S** will only have one child (see A8); if splitting is necessary in step A9, then **S** will have exactly two children.

Splitting simplices is more complicated than splitting boxes, since new edges are introduced during the process and their length depends on the shape of the parent simplex. Nevertheless, we know that the new edges cannot be longer than the longest edge of the parent, since they lie completely inside the parent simplex (a simplex is always convex). We therefore know:

Lemma 9 *Let l_{max} be the length of the longest edge of a simplex* **S**. *After maximal $s_{split} = \frac{n(n+1)}{2}$ binary splitting steps of* **S** *and its children the maximal edge length of any grandchild is less than $\frac{l_{max}}{2}$.*

Proof We know that a simplex has $\frac{n(n+1)}{2}$ edges. After splitting the longest edge the two children have at least one edge with edge length $\frac{l_{max}}{2}$. The lengths of all other edges have to be less or equal to l_{max}, since newly created edges cannot be longer than l_{max}. After repeating the split step $s_{split} = \frac{n(n+1)}{2}$ times we end up with $2^{\frac{n(n+1)}{2}}$ grandchildren each containing no edges that are longer than $\frac{l_{max}}{2}$. □

Of course s_{split} is just an upper bound, generally the children shrink much faster since the newly created edges are much shorter than l_{max}. Also in most cases not all edges of **S** have the same length l_{max}, many edges are generally shorter than the longest edge.

We now analyze the case when a simplex shrinks fast enough so that we do not need a split.

Lemma 10 *Let l_{max} be the length of the longest edge of a simplex* **S**. *After maximal $s_{shrink} = -\frac{\log 2}{\log CRIT}$ shrinking steps of* **S** *and its children the maximal edge length of any grandchild is less than $\frac{l_{max}}{2}$.*

Proof When a simplex is shrinking fast enough, all edges have to be less or equal to $CRIT$ times their original length (since CRIT < 1). Of course after s steps they decrease at least by the factor of $CRIT^s$. Therefore it takes maximal $s_{shrink} = -\frac{\log 2}{\log CRIT}$ steps for them to reach half of their original length. □

Corollary 2 *Suppose we are given some number $\varepsilon > 0$. Then there exists a number s_{max} such that after s_{max} or fewer steps, every edge of every child of a simplex* **S** *is shorter than ε.*

Proof Let again l_{max} be the longest edge of **S**. Set $s_{0.5} := s_{split} + s_{shrink}$ then we know from Lemma 9 and 10 that every edge of every child is shorter than $\frac{l_{max}}{2}$ (by shrinking or splitting). It is clear that there exists some integer $i > 0$ such that $\frac{l_{max}}{2^i} < \varepsilon$ for any $\varepsilon > 0$. Setting $s_{max} = i(s_{0.5})$ completes the proof. □

Corollary 2 does not reveal anything about the actual speed of convergence. Nevertheless it ensures that after a maximum of s_{max} iteration steps all edges of the remaining simplices are shorter than a given ε. However, we cannot tell how many roots are contained in such a simplex or if it contains any roots at all. The algorithm only ensures that no roots are missed. Anyhow, by using rounded interval arithmetic and checking the solutions against the equations most false positives can be discarded.

5.2.2 Complexity Analysis

It is difficult a priori to predict how many iterations will actually be needed to complete the whole algorithm. However, we can analyze the amount of time required to execute each *iteration* of the algorithm individually (see

Definition 6). In the analysis, we will assume that each n dimensional equation is of total degree m.

For each iteration two major procedures are performed: (A3) the generation of lower bounds and (A7) the extraction (deCasteljau subdivision). We will first consider these functions separately, and subsequently analyze how they affect the complexity of a single iteration.

We will begin with the generation of lower bounds, which is used in step A3.

Lemma 11 *The number of computational steps needed to obtain the lower bound of one variable on a simplex from the coefficients of an n-dimensional barycentric Bernstein polynomial of total order m is $O\big(\frac{(m+n)!}{m!n!}\big)$.*

Proof The number of coefficients describing an n-dimensional barycentric Bernstein polynomial of total degree m is given by Eq. 22. For the selected variable the 2D control points of the corresponding $\mathbf{min}'$ and $\mathbf{max}'$ curves must be created from the minimal and maximal coefficients (see Section 3.2). Comparison of coefficients and construction of the control points can be carried out with a single run through all coefficients. The resulting $2m+1$ control points may be traversed as in Graham's scan [2] to compute their convex hull. The entire running time of this procedure is $O\big(\frac{(m+n)!}{m!n!}\big)$. The time needed for the convex hull traversal in order to compute the intersections with the x-axis is no greater. Hence Lemma 11 follows. □

To extract the new system in step A7 several subdivisions are performed using the deCasteljau algorithm as described in Sections 5.1.3 and 5.1.4. We computed the cost of one subdivision along a single edge in Lemma 6. Furthermore, Lemma 8 states the cost for an extraction on a smaller simplex. With these Lemmas and Lemma 11, we can now consider a complete iteration.

Theorem 3 *The number of computational steps needed to complete one iteration as in Definition 6 for a system of N n-dimensional barycentric Bernstein polynomials of total order m is $O\big(N\frac{(m+n)!}{(m-1)!(n-1)!}\big)$.*

Proof For each iteration the projection (and intersection with the x-axis) in step A3 has to be executed for N polynomials and for every variable n. Hence by Lemma 11 the number of computations performed in that step is $O\big(Nn\frac{(m+n)!}{m!n!}\big)$. The intersection of the N intervals in step A4, the shrinking in A5, the accuracy computation in A6, and the tolerance computation in A8 do not contribute significant time to the execution. Step A7 (extraction) is also performed for N polynomials which by Lemma 8 results in $O\big(N\frac{(m+n)!}{(m-1)!(n-1)!}\big)$ computations. If the binary split in step A9 is executed, it does not add any significant time, since it is just a single subdivision. Hence the number of computational steps for one iteration is

$$\begin{aligned} & O\left(N\frac{(m+n)!}{(m-1)!(n-1)!}+N\frac{(m+n)!}{m!(n-1)!}\right) \\ = \quad & O\left(N\frac{(m+n)!}{(m-1)!(n-1)!}\right) \end{aligned} \tag{30}$$

□

Equation 30 gives a tight bound for the running time of one iteration. In fact, if it is expressed in terms of the number of control points C_B given by Eq. 22, we get

$$O(NnmC_B). \tag{31}$$

5.2.3 Comparison with tensor product method

The running time of the tensor product Bernstein approach is [18]

$$O(Nnm^{n+1}) = O(NnmC_T) \tag{32}$$

where $C_T := (m+1)^n$ is the number of control points. Therefore the reduced complexity of a single iteration of the barycentric approach is due to the reduced number of control points used. Table 1 shows the differences $C_T - C_B$ for a few m and n. It can be seen that there is no difference for $n = 1$ but for larger n the difference becomes huge. Therefore an iteration step of the algorithm with the barycentric Bernstein base will be much faster than a corresponding step in the tensor product Bernstein base.

Note that the number of iterations actually executed may be quite different for systems with the same n and m. Local convergence properties would give a better idea of how many iteration steps and how much time is required. Because of the nature of barycentric coordinates, it is rather difficult to give a complete analysis of the convergence. As opposed to the algorithm based on a tensor product Bernstein basis, where all variables are independent, barycentric formulations have a dependent variable, leading to more complex computations. Nevertheless, in one-dimensional problems the tensor product method described in [18] is identical with the barycentric formulation used here. This is due to the fact, that one can always express one of the two barycentric coordinates by the other one ($x_2 = 1 - x_1$) on the line segment $x_1, x_2 \in [0, 1]$. Therefore the convergence in one-dimensional problems for barycentric coordinates is quadratic as in the tensor product case.

6 Numerical Examples

In this section we will present several numerical examples.

m										
10	0	55	1045	13640	158048	1763553	19467723	214315123	2357855313	
9	0	45	780	9285	97998	994995	9988560	99975690	999951380	
8	0	36	564	6066	57762	528438	4776534	43033851	387396179	
7	0	28	392	3766	31976	260428	2093720	16770781	134206288	
6	0	21	259	2191	16345	116725	821827	5761798	40348602	
5	0	15	160	1170	7524	46194	279144	1678329	10075694	
4	0	10	90	555	2999	15415	77795	390130	1952410	
3	0	6	44	221	968	4012	16264	65371	261924	
2	0	3	17	66	222	701	2151	6516	19628	
1	0	1	4	11	26	57	120	247	502	
	1	2	3	4	5	6	7	8	9	n

Table 1 Difference of number of control points $C_T - C_B$

Root#	root interval
1	[0.9999999999999957, 1.0000000000000040]
2	[0.9499999774232312, 0.9500000008271431]
3	[0.8999999999900907, 0.9000000000019098]
4	[0.8499999373741308, 0.8500000000663835]
5	[0.7999999898226192, 0.8000000002309731]
6	[0.7499999951486940, 0.7500000012060561]
7	[0.6999999827678465, 0.7000000051018760]
8	[0.6499999741539105, 0.6500000124953802]
9	[0.5999999771104132, 0.6000000228526872]
10	[0.5499999663267969, 0.5500000334627025]
11	[0.4999999627040412, 0.5000000374906769]
12	[0.4499999678939680, 0.4500000318330576]
13	[0.3999999791813783, 0.4000000210180419]
14	[0.3499999895897444, 0.3500000105424416]
15	[0.2999999961907331, 0.3000000038425950]
16	[0.2499999989989713, 0.2500000010073070]
17	[0.1999999998191774, 0.2000000001785774]
18	[0.1499999999800867, 0.1500000000210968]
19	[0.0999999999987760, 0.1000000000015307]
20	[0.0499999999924749, 0.0500000004950866]

Table 2 The 20 roots of the Wilkinson polynomial

6.1 1D - Wilkinson Polynomial

As a first example we will solve a scaled version of the Wilkinson polynomial of degree 20 which was shown by Wilkinson [20] to be extremely ill-conditioned. The one-dimensional polynomial

$$P(u) = \prod_{k=1}^{20} (u - \frac{k}{20}) \tag{33}$$

has 20 equidistant roots in the interval $[0, 1]$. Because the conversion process into a Bernstein basis is very ill-conditioned, we used exact rational arithmetic to compute the Bernstein coefficients. The computation of the zeros can then be done in rounded interval arithmetic. We used a tolerance EPS= 10^{-7} for the precision and immediately obtained all 20 roots (in 77 iterations with 21 binary edge subdivisions) (see Table 2).

6.2 2D - Curve Intersection

As a two dimensional example we obtain the stationary points of the function

$$\begin{aligned} f(u_1, u_2) = &[(u_1 - \frac{1}{4})^3 + (u_2 - \frac{1}{4})^3 - (\frac{3}{5})^3]^2 \\ &+[(u_1 - \frac{3}{4})^3 + (u_2 - \frac{3}{4})^3 - (\frac{3}{5})^3]^2 \end{aligned} \tag{34}$$

by computing the zeros of its partial derivatives

$$\begin{aligned} f_1(u_1, u_2) = &12u_1^5 - 30u_1^4 + \tfrac{75}{2}u_1^3 - \tfrac{31467}{1000}u_1^2 + \tfrac{128361}{8000}u_1 \\ &+12u_1^2u_2^3 - 18u_1^2u_2^2 + \tfrac{45}{4}u_1^2u_2 - 12u_1u_2^3 \\ &+\tfrac{45}{2}u_1u_2^2 - \tfrac{63}{4}u_1u_2 + \tfrac{15}{4}u_2^3 - \tfrac{63}{8}u_2^2 + \tfrac{369}{64}u_2 \\ &-\tfrac{5871}{1600} \\ f_2(u_1, u_2) = &\tfrac{15}{4}u_1^3 - \tfrac{63}{8}u_1^2 + \tfrac{369}{64}u_1 + 12u_1^3u_2^2 - 12u_1^3u_2 \\ &-18u_1^2u_2^2 + \tfrac{45}{2}u_1^2u_2 + \tfrac{45}{4}u_1u_2^2 - \tfrac{63}{4}u_1u_2 \\ &+12u_2^5 - 30u_2^4 + \tfrac{75}{2}u_2^3 - \tfrac{31467}{1000}u_2^2 + \tfrac{128361}{8000}u_2 \\ &-\tfrac{5871}{1600} \end{aligned} \tag{35}$$

each of total order $m = 5$. Function $f_2 := f_1(u_2, u_1)$ is a mirrored version of f_1. These two functions can be viewed as implicitly defined curves. Therefore the common roots are their intersection points. Fig. 3 shows both curves (for the computation we used our method as described in Section 6.5).

In this example we searched for the common root in the triangle $[(0, 0); (2, 0); (0, 2)]$ and used a tolerance of EPS= 10^{-12}. After 127 iteration steps (with 53 binary subdivisions) we obtained the root:

$$u_1 = [0.726602621583884, 0.726602621590015] \tag{36}$$

$$u_2 = [0.726602621584005, 0.726602621590439] \tag{37}$$

The solver based on tensor product Bernstein bases [18] needed 539 iterations (with 160 subdivisions) to obtain the slightly larger intervals:

$$u_1 = [0.726602621580483, 0.726602621593556] \tag{38}$$

$$u_2 = [0.726602621580798, 0.726602621593114] \tag{39}$$

In this example both approaches run very fast in less than 0.1 seconds.

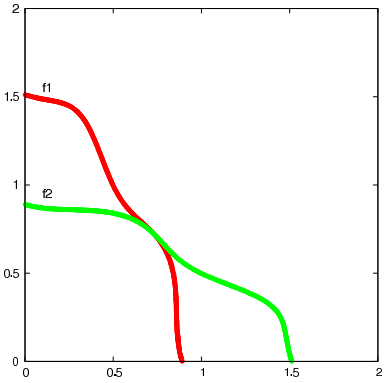

Fig. 3 Intersection of two implicitly defined curves

6.3 4D - Example

In order to compute the extrema of the squared distance between two circles, a system of four unknowns has to be solved. Let (u_1, u_2) be a point on one circle and (u_3, u_4) be a point on the other circle, then the two circles are

$$(u_1 - \frac{1}{5})^2 + (u_2 - \frac{1}{5})^2 - \frac{1}{25} = 0 \tag{40}$$

$$(u_3 - \frac{1}{5})^2 + (u_4 - \frac{4}{5})^2 - \frac{1}{25} = 0 \tag{41}$$

The squared distance between points on the two circles is simply

$$D = (u_1 - u_3)^2 + (u_2 - u_4)^2. \tag{42}$$

We consider u_1, u_3 to be independent variables while u_2 depends on u_1 and u_4 depends on u_3. For (u_1, u_2, u_3, u_4) to be an extremum of D, the partial derivatives of D with respect to the independent parameters must be zero.

$$\frac{\partial D}{\partial u_1} = 2(u_1 - u_3) + 2(u_2 - u_4)\frac{\partial u_2}{\partial u_1} = 0 \tag{43}$$

$$\frac{\partial D}{\partial u_3} = -2(u_1 - u_3) - 2(u_2 - u_4)\frac{\partial u_4}{\partial u_3} = 0 \tag{44}$$

By implicit differentiation of Eq. 40 and Eq. 41, we obtain $\frac{\partial u_2}{\partial u_1}$ and $\frac{\partial u_4}{\partial u_3}$ which will be substituted into Eq. 43 and Eq. 44 to yield:

$$(u_1 - u_3)(u_2 - \frac{1}{5}) - (u_2 - u_4)(u_1 - \frac{1}{5}) = 0 \tag{45}$$

$$(u_1 - u_3)(u_4 - \frac{4}{5}) - (u_2 - u_4)(u_3 - \frac{1}{5}) = 0 \tag{46}$$

Together with the two equations of the circles we now have four equations. We used a tolerance of EPS= 10^{-7} and the initial bounding simplex with one vertex at the origin and the others on the four axes at $u_i = 3$. Table 3 shows the four roots. Instead of showing the interval of every coordinate we rounded to the 7th decimal place so that the left and right interval limits become equal.

Root number	root (u_1, u_2, u_3, u_4)
1	(0.2 , 0.0 , 0.2 , 0.6)
2	(0.2 , 0.4 , 0.2 , 1.0)
3	(0.2 , 0.4 , 0.2 , 0.6)
4	(0.2 , 0.0 , 0.2 , 1.0)

Table 3 Four solutions

It should be noted that this example actually runs faster in the tensor product Bernstein basis approach, since we need many more iteration steps to converge. This is due to the fact, that we only use the lower bounds of the intersection intervals for the computation of the next smaller simplex.

6.4 More Equations than Unknowns

Having more equations than unknowns, as for example when computing singular points of an implicit algebraic curve where $f(u_1, u_2) = \frac{\partial f}{\partial u_1} = \frac{\partial f}{\partial u_2} = 0$, does not lead to any problems. In fact the additional equation helps to discard subsimplices without roots quicker, leading to a faster convergence. As an example we use a simple implicit 2D curve (see also Fig. 4 where we used our method as described in Sect. 6.5 to compute the curve):

$$f(u_1, u_2) = u_1^3 + u_2^3 - 3u_1u_2 \tag{47}$$

Together with the two partial derivatives:

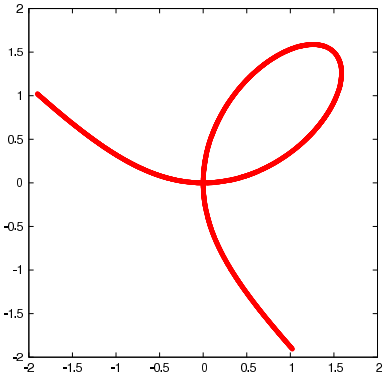

Fig. 4 Singular point

$$\begin{aligned} f_1(u_1, u_2) &= 3u_1^2 - 3u_2 \\ f_2(u_1, u_2) &= 3u_2^2 - 3u_1 \end{aligned} \tag{48}$$

we computed the common zero at $(0, 0)$ with an initial bounding simplex of

$$S = \left[\begin{pmatrix} 1 \\ -0.2 \end{pmatrix}; \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \begin{pmatrix} -1 \\ -0.2 \end{pmatrix} \right] \tag{49}$$

and a tolerance of EPS= 10^{-8} in 87 iteration steps (with 34 subdivisions) to be in the interval:

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \begin{pmatrix} \left[-3.988 \cdot 10^{-09}, 3.988 \cdot 10^{-09}\right] \\ \left[-6.245 \cdot 10^{-15}, 4.786 \cdot 10^{-09}\right] \end{pmatrix} \tag{50}$$

Using only two equations instead (either f and f_1 or f and f_2) results in a much slower convergence due to the tangency at the intersection. But even when using only f_1 and f_2 with a smaller total order of $m = 2$ and orthogonal intersection, we need more steps (91 iterations and 35 subdivisions).

6.5 More Unknowns than Equations

If less equations than unknowns are given, a computation is still possible. Of course the roots will not be isolated points anymore, but rather a connected point set, e.g. a curve. By adjusting the tolerance EPS the number of solutions can be controlled. As a result we end up with many small simplices, covering the solution point set. As an example we computed the intersection of two spheres with radii $r = 0.5$ and centers $(0, 0, 0)$ and $(0.75, 0, 0)$:

$$f_1(u_1, u_2, u_3) = u_1^2 + u_2^2 + u_3^2 - \frac{1}{4}; \tag{51}$$

$$f_2(u_1, u_2, u_3) = u_1^2 + u_2^2 + u_3^2 - \frac{3}{2}u_1 + \frac{5}{16}; \tag{52}$$

Figure 5 shows the intersection result, where the center of the intervals are plotted as dots. The circle lies parallel to the u_2u_3-plane at $u_1 = 0.375$. We used an

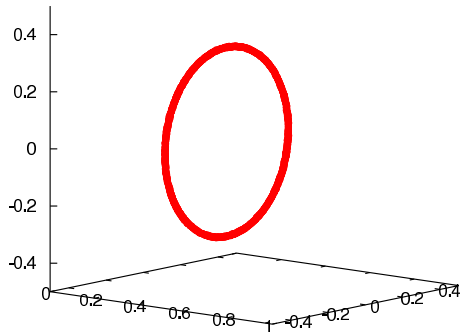

Fig. 5 Intersection of two spheres

EPS= 10^{-3} and obtained 1236 root simplices after 5255 iteration steps (with 2356 binary edge subdivisions).

6.6 Finding Complex Roots

As with the projected polyhedron method for tensor product Bernstein bases, complex roots can be found with the barycentric solver easily by substituting $\mathbf{a} + \mathbf{ib}$ for $\mathbf{u}$ in Eq. 8 and separating the real and imaginary parts, thus doubling the number of equations. So for N complex equations with n complex variables we get $2N$ real equations with $2n$ real variables. For example, the two complex equations

$$\begin{aligned} 0 &= z_1^2 + z_1 z_2 - z_2^2 + 0.51 - 1.1i \\ 0 &= z_2^2 - z_1 + 0.29 + 0.6i \end{aligned} \tag{53}$$

can be separated into

$$\begin{cases} 0 = u_1^2 - u_2^2 + u_1u_3 - u_2u_4 - u_3^2 + u_4^2 + 0.51 \\ 0 = 2u_1u_2 + u_1u_4 + u_2u_3 - 2u_3u_4 - 1.1 \\ 0 = u_3^2 - u_4^2 - u_1 + 0.29 \\ 0 = 2u_3u_4 - u_2 + 0.6 \end{cases} \tag{54}$$

with $z_1 = u_1 + u_2i$ and $z_2 = u_3 + u_4i$. Over the initial bounding simplex with one vertex at the origin and the others on the four axes at $u_i = 3$ we find the single root at $(0.5, 0.8, 0.5, 0.2)$ (up to 7 digits exact with $EPS = 10^{-7}$). So the common zero is at $z_1 = 0.5 + 0.8i$ and $z_2 = 0.5 + 0.2i$.

7 Conclusions and Recommendations

We have shown how systems of nonlinear polynomials in the barycentric Bernstein basis can be solved using the projected-polyhedron method developed by Sherbrooke and Patrikalakis [18]. Our implementation of the method in rounded interval arithmetic [1] is numerically robust and works independent of the coordinate axes.

There are a number of topics that need to be expanded upon if we wish to better quantify the capabilities of this solver method by itself:

Local convergence properties of the method need to be analyzed in order to get a better idea of the number of iterations the solver will use in general.

Tighter bound generation than the orthogonal projection can achieve would benefit the solver. We obtain both upper and lower bounds on the parameters in our projection method, but are only able to use the lower bound efficiently. It is worth analyzing alternative approaches to this problem in order to identify possible improvements. It is e.g., possible to obtain the solution in already converged dimensions (where the lower bound is close enough to the upper bound) and reduce the remaining problem to operate on a lower dimensional simplex.

It is possible that more intelligent simplex subdivision based on the pattern of bounding simplex generation could speed up the algorithm when convergence slows down. In the current implementation the simplex is always split at the midpoint of the longest edge.

Furthermore, methods to identify regions with exactly one solution (e.g., [11]) should be used to ensure that we do not prematurely terminate the subdivision process (if two roots are within EPS). Also, if a subregion contains exactly one solution, a Newton-Raphson

iteration can for example be used to quickly converge to that root.

Representation of control points and simplices in rounded interval arithmetic lead to overlapping search regions which is likely to cause reduced performance at high accuracy. It might be worth looking into methods for reducing this problem, or implementing methods for adaptively selecting the appropriate precision for a given problem.

Acknowledgements This work was partially funded by NSF grant number DMI-062933 and by a von Humboldt Foundation postdoctoral fellowship to the first author. The authors would like to thank Dr. W. Cho and the unknown reviewers for their insightful comments.

References

1. Abrams, S.L., Cho, W., Hu, C.Y., Maekawa, T., Patrikalakis, N.M., Sherbrooke, E.C., Ye, X.: Efficient and reliable methods for rounded-interval arithmetic. Computer-Aided Design **30**(8), 657–665 (1998)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge, MA (1990)
3. Elber, G., Kim, M.S.: Geometric constraint solver using multivariate rational spline functions. In: SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications, pp. 1–10. ACM Press, New York, NY, USA (2001). DOI http://doi.acm.org/10.1145/376957.376958
4. Farin, G.: Triangular Bernstein-Bézier patches. Computer Aided Geometric Design **3**(2), 83–128 (1986)
5. Farin, G.: Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide, 4th edn. Academic Press, San Diego (1997)
6. Farin, G.E.: Geometric Modeling: Algorithms and New Trends. SIAM, Philadelphia (1987)
7. Goldman, R.N.: Subdivision algorithms for Bézier triangles. Computer-Aided Design **15**(3), 159–166 (1983)
8. Goldman, R.N.: Blossoming and knot insertion algorithms for B-spline curves. Computer Aided Geometric Design **7**, 69–81 (1990)
9. Goldman, R.N., Lyche, T. (eds.): Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces. SIAM, Philadelphia (1993)
10. Graham, R.L., Knuth, D.E., Patashnik, O.: Concrete Mathematics: A Foundation for Computer Science, 2nd edn. Addison-Wesley, Boston (1994)
11. Hanniel, I., Elber, G.: Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. Comput. Aided Des. **39**(5), 369–378 (2007). DOI http://dx.doi.org/10.1016/j.cad.2007.02.004
12. Hoschek, J., Lasser, D.: Fundamentals of Computer Aided Geometric Design. A. K. Peters, Wellesley, MA (1993). Translated by L. L. Schumaker
13. Hu, C.Y., Maekawa, T., Patrikalakis, N.M., Ye, X.: Robust interval algorithm for surface intersections. Computer-Aided Design **29**(9), 617–627 (1997)
14. Lane, J.M., Riesenfeld, R.F.: A theoretical development for the computer generation and display of piecewise polynomial surfaces. IEEE Trans. Pattern Anal. Mach. Intell. **2**, 35–46 (1980)
15. Mourrain, B., Pavone, J.P.: Subdivision methods for solving polynomial equations. Tech. Rep. 5658, INRIA Sophia-Antipolis (2005)
16. Sablonnière, P.: Spline and bézier polygons associated with a polynomial spline curve. Computer-Aided Design **10**(4), 257–261 (1978)
17. Sederberg, T.W.: Algorithm for algebraic curve intersection. Computer-Aided Design **21**(9), 547–554 (1989)
18. Sherbrooke, E.C., Patrikalakis, N.M.: Computation of the solutions of nonlinear polynomial systems. Computer Aided Geometric Design **10**(5), 379–405 (1993)
19. Waggenspack, W.N., Anderson, C.D.: Converting standard bivariate polynomials to Bernstein form over arbitrary triangular regions. Computer-Aided Design **18**(10), 529–532 (1986)
20. Wilkinson, J.H.: Rounding Errors in Algebraic Processs. Prentice-Hall, Englewood Cliffs (1963)

Martin Reuter Dr. Reuter is a Feodor-Lynen Fellow of the Humboldt Foundation and works as a postdoc at MIT, Department of Mechanical Engineering since 2006. He has been awarded a prize for outstanding scientific accomplishments in 2006 of the Leibniz University Hannover where he obtained his Ph.D. in 2005 in the area of shape recognition from the faculty of electrical engineering and computer science. Before that he obtained his Diplom (M.Sc.) in mathematics with a second major in computer science and a minor in business information technology from the Leibniz University of Hannover in 2001. His research interests include computational geometry and topology, computer aided design, geometric modeling and computer graphics.

Tarjei S. Mikkelsen Tarjei Mikkelsen is a Medical Engineering and Medical Physics PhD candidate in the Harvard-MIT Division of Health Sciences and Technology. He holds an S.B. in Mathematics and an M.Eng. in Biomedical Engineering from MIT. His current research interests include genomics, medical informatics and software engineering.

Evan C. Sherbrooke Dr. Sherbrooke received the SB degree in Mathematics from the Massachusetts Institute of Technology in 1990, SM degrees in naval architecture and in electrical engineering and computer science from MIT, in 1993, and a PhD degree in computer aided design and manufacturing from MIT in 1995. Research interests include computational and differential geometry, alternative representations, and the problems of accuracy and robustness in numerical computation and data exchange.

Takashi Maekawa Dr. Maekawa is Professor of Mechanical Engineering at Yokohama National University (Japan) since 2003. He directs the Digital Engineering Laboratory at YNU. Before joining YNU he was a Principal Research Scientist at MIT and a Design and Manufacturing Engineer at Bridgestone Corp.. His research activities address a wide range of problems related to geometric modeling, computational and differential geometry, and CAD/CAM. He received SB and SM degrees from Waseda University and a PhD degree from MIT.

Nicholas M. Patrikalakis Dr. Patrikalakis is the Kawasaki Professor of Engineering and Associate Head of the Department of Mechanical Engineering at MIT. His current research interests include: computer-aided design, geometric modeling, robotics and distributed information systems for multidisciplinary data-driven forecasting. He is a member of ACM, ASME, CGS, IEEE, ISOPE, SIAM, SNAME and TCG, and he is editor, co-editor, or member of the editorial board of six international journals (CAD, JCISE, IJSM, TVC, GMOD, IJAMM). He has served as program chair of CGI '91, program co-chair of CGI '98, Pacific Graphics '98, ACM Solid Modeling Symposium 2002, co-chair of the ACM Solid Modeling Symposium 2004, and Chair of the 2005 Convention on Shapes and Solids (ACM SPM'05 and SMI'05).