Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 543–552
DOI: 10.15439/2014F183
ACSIS, Vol. 2

Solving Systems of Polynomial Equations: a Novel End Condition and Root Computation Method

Maciej Bartoszuk
Interdisciplinary PhD Studies Program,
Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
Email: m.bartoszuk@phd.ipipan.waw.pl

***Abstract*—In this paper we present an improvement of the algorithm based on recursive de Casteljau subdivision over an n-dimensional bounded domain (simplex or box). The modification consists of a novel end condition and a way of calculation the root in subdomain. Both innovations are based on linear approximation of polynomials in a system. This improvement results in that our approach takes almost half of the time of the standard approach: it can be stopped much earlier than using standard diameter condition and getting midpoint of a subdomain as a root.**

I. Introduction

SOLVERS of systems of algebraic equations are a very important part of today's CAD/CAM systems. Issues such as the numerical representation of curves and surfaces [1], physical contacts between objects, collision detection, representations of Voronoi diagrams of set-theoretic models [2] or formulation of configuration space of a robot in motion planning applications [3] are reduced to finding solutions of the systems of polynomial equations.

Finding roots of polynomials (one equation) is well researched. Unfortunately, the problem of solving systems of polynomial equations goes back to the ancient Greeks and Chinese and still there is no one good solving method. There are a number of algorithms invented over the years based on different approaches. For details, see section II. It is worth noticing that in many of applications, especially CAD/CAM, like collision detection or curves intersection, there is no need to find all solutions in $\mathbb{R}^n$ and problem is narrowed to a bounded domain, which can be approximated (bounded) by n-dimensional box or simplex. That is why algorithms which can be applied only to a specific domain are so desirable.

The method described in this paper is a multidimensional bisection algorithm that uses multivariate Bernstein representation of polynomials, de Casteljau subdivision and convex hull property. Brief review of the algorithm is presented in next sections, but for further details we refer readers to [4], [5].

First of all, the method can be applied to an n-dimensional box or simplex domain. Such domains are used in curves and surfaces representation, so the algorithm can be used naturally in CAD/CAM applications. The further advantages of the method are a numerical stability, finding all zeros in the domain and no need for entering a starting point. A major drawback is the exponential complexity $O(2^n)$ where n is a number of equations. However, a number of publications in recent years have proved that this algorithm is effective for surfaces or curves intersections, where a number of equations in the system is equal to one or two [6].

In addition, research on the use of graphics cards (CUDA technology) were conducted to improve the execution time of the algorithm [7].

Our contributions are twofold:

1) We show novel end condition in multidimensional bisection algorithm which is based on linear approximation of equations
2) We show a novel way of computing the root in multidimensional bisection algorithm which is based on solving a linear system of the linear approximations

In this paper we consider a unit n-simplex as a domain of a system of polynomial equations.

The paper is organized as follows: in Section II we discuss related work, in Section III we briefly introduce the theoretical background for a multidimensional bisection algorithm. After that we describe two geometrical interpretations in Section IV. In the Section V we discuss the details of our method (especially end condition and calculation of the root in subdomain). In the Section VI reader can find numerical results obtained for various sets of equations. Section VII provides conclusions drawn from the presented analysis.

II. Related Work

Current methods of solving systems of polynomial equations can be divided into three groups: symbolical, geometrical and numerical (in particular subdivision) solvers.

A. Symbolical/Algebraic solvers

Symbolical methods are based on resultants and Gröbner bases. Those methods eliminate variables and reduce problem to finding roots for univariate polynomial using rational univariate representation. Those methods, however, are efficient only for systems of three up to four polynomials of low degree, such as 2 or 3. After reducing those polynomials, we obtain one univariate polynomial of degree close to 15. As it was shown by Wilkinson [8], computing roots of polynomial of degree greater than 15 can be ill-conditioned. What is more, these methods are difficult to implement for computers that

978-83-60810-58-3/$25.00

use finite precision arithmetic and that also slows down the resulting algorithm. Symbolical methods should be considered a success in theoretical area, but practical impact is unclear. An example of such an approach is [9], [10].

B. Geometrical solvers

For some specific applications, particular methods have been developed. Those methods use geometrical formulation of a problem. For example, for curves and surface intersection or ray tracing the algorithms are based on subdivision. However, these methods have limited applications in the general case [11].

C. Numerical/Analytic solvers

Numerical methods are probably the most known. They can be classified into iterative methods and homotopy methods. The most popular iterative method is Newton's method and it works well locally and only if initial point is a good guess, which is difficult in solving systems of polynomial equations. Other methods are Newton like methods, minimization methods or Weierstrass method [12], [13].

Homotopic methods have a well-explored theoretical background. They are based on proceeding path in a complex space. Theoretically, every path should converge to an isolated solution. In practice, however, there are many issues as the paths are not geometrically isolated, which causes problems with robustness of the approach. Moreover, those methods are rather computationally demanding. Example of this approach is [14], [15], [16].

Subdivision methods use an exclusion criterion to remove a domain if it does not contain a root. These solvers are often used to isolate the real roots. Exclusion criteria are based on Taylor exclusion function [17], interval arithmetic [18], Turan test [19], Sturm method [20], Descartes rule [21]. In this paper we propose an improvement to subdivision algorithm, where excluding is based on properties of Bernstein representation of polynomials. More information about that method can be found in [4], [5], [7].

Interestingly, computing on GPU becomes more and more popular, recently there are attempts to calculate many numerical problems on GPU, in particular solving polynomial equations systems [7], [15], [22], [23].

III. Problem formulation and algorithm idea

Consider a set of n polynomial equations in n independent variables

$$\mathbf{p}(\mathbf{x}) = \mathbf{0} \tag{1}$$

where $\mathbf{p} = (p_1, p_2, \ldots, p_n)\colon S_1^n \to \mathbb{R}^n$ (S_1^n is a unit n-simplex). The problem is to calculate numerically, with a given accuracy ϵ, all real roots $\{\mathbf{x}_0\}$ of the system (1). The method can also be used when the number of equations is not equal to the number of variables, but it is not in the scope of this paper.

This algorithm uses a simplex (also known as barycentric) Bernstein representation of multivariate polynomials. Vectors of this basis are as follows (N - degree of Bernstein polynomial):

$$B^N_{(j_1,\ldots,j_n)}(x_1,\ldots,x_n) = \frac{N!}{j_0!\cdot j_1!\cdot\ldots\cdot j_n!}\cdot x_0^{j_0}\cdot x_1^{j_1}\cdot\ldots\cdot x_n^{j_n} \tag{2}$$

where

$$j_k \in \mathbb{N}\cup\{0\}, x_k \in [0,1]\wedge x_0+\cdots+x_n \le 1\wedge k\in\{1,\ldots,n\}$$

and

$$j_1+\cdots+j_n \le N, j_0 = N-(j_1+\cdots+j_n), x_0 = 1-(x_1+\cdots+x_n)$$

so each polynomial p_k, after conversion to Bernstein representation b_k is of the form

$$b_k(\mathbf{x}) = \sum_{(j_1,\ldots,j_n)\in\{(j_1,\ldots,j_n):j_1+\cdots+j_n\le N\}} b^{(k)}_{(j_1,\ldots,j_n)} B^N_{(j_1,\ldots,j_n)}(\mathbf{x}) \tag{3}$$

and $b^{(k)}_{(j_1,\ldots,j_n)}$ are called Bernstein coefficients. We will also use coefficients of the system:

$$\mathbf{b}_{(j_1,\ldots,j_n)} = \begin{bmatrix} b^{(1)}_{(j_1,\ldots,j_n)} \\ b^{(2)}_{(j_1,\ldots,j_n)} \\ \vdots \\ b^{(n)}_{(j_1,\ldots,j_n)} \end{bmatrix}_{n\times 1}$$

From that point we will refer to a system of polynomial equations in Bernstein basis as $\mathbf{b}(\mathbf{x})$, so we can rewrite the main problem as

$$\mathbf{b}(\mathbf{x}) = \mathbf{0} \tag{4}$$

Definition III.1 (Extreme coefficients of polynomial in Bernstein form)**.** Extreme coefficients of polynomial in Bernstein form are

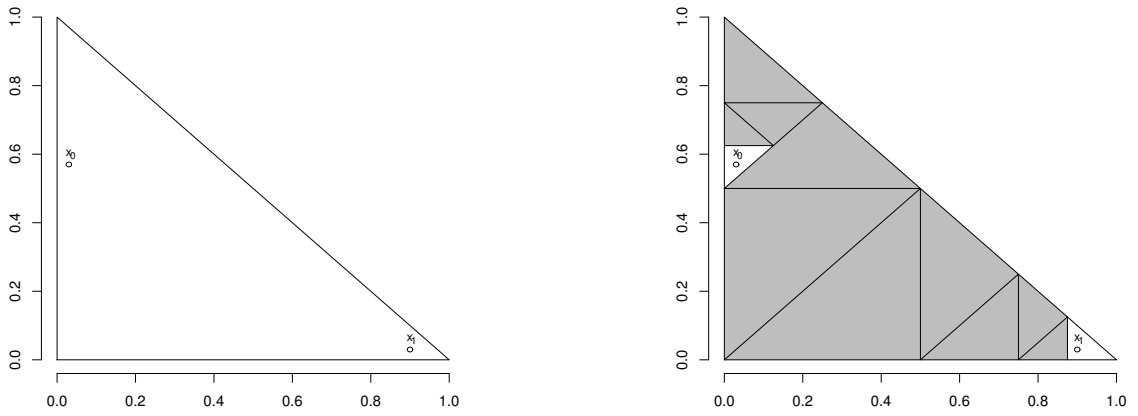
$$I_e = \{(j_1,\ldots,j_n)\colon j_1+\cdots+j_n \le N$$
$$\wedge(\forall_{k\in\{1,\ldots,n\}} j_k = 0 \vee \exists_{k\in\{1,\ldots,n\}} j_k = N)\wedge j_k\in\mathbb{N}\cup\{0\}\}$$

In other words, extreme coefficients of polynomial b_k are $b^{(k)}_{0,\ldots,0}$, $b^{(k)}_{N,0,\ldots,0}$, …, $b^{(k)}_{0,0,\ldots,N}$.

Example III.1. *For polynomial b_k of three variables and second degree extreme coefficients are $b^{(k)}_{000}$,$b^{(k)}_{200}$,$b^{(k)}_{020}$,$b^{(k)}_{002}$.*

A pseudocode is listed in Algorithm 1. Firstly, we have to convert input polynomials from $p_1,\ldots,p_n$ to $b_1,\ldots,b_n$. An efficient, robust algorithm is presented e.g. in [24]. After that, we create two queues S and Q, the first with root of the system and second with areas (simplexes) to be processed.

Routine is that: get one subsimplex (in the very beginning it is the initial simplex, where our system is defined), which means get coefficients of polynomials in the system defined on that subsimplex. After that, perform root exclusion tests. The tests can exclude subdomain, where there is no root with 100% certainty. Therefore, all of those tests have 100% true negative ratio. However, tests differ in false positive ratio. The fastest



(a) Domain before applying the algorithm

(b) Divided domain after applying the algorithm

Fig. 1: Algorithm idea, division of domain

Algorithm 1 Bisection algorithm to solve a system of polynomial equations

Require: Polynomials $b_1 \dots b_n$ in a system
Ensure: List of roots of the system
Initialize empty queue Q of n-tuples of polynomials;
Add $b_1 \dots b_n$ to Q
Initialize empty queue S of solutions;
while Q is not empty **do**
  Get n-tuple of $b_1 \dots b_n$ polynomials from Q
  Perform end condition test
  if End condition is true **then**
    Calculate root and add it to S queue
    CONTINUE;
  end if
  Divide $b_1 \dots b_n$ polynomials using de Casteljau method into $b_1^a \dots b_n^a$ and $b_1^b \dots b_n^b$ polynomials
  Perform tests (from fastest to slowest) and determine if every $b_1^a \dots b_k^a$ is suspected to have a root.
  if Every $b_1^a \dots b_n^a$ is suspected to have a root **then**
    Add $b_1^a \dots b_n^a$ into Q;
  end if
  Perform tests (from fastest to slowest) and determine if every $b_1^b \dots b_n^b$ is suspected to have a root.
  if Every $b_1^b \dots b_n^b$ is suspected to have a root **then**
    Add $b_1^b \dots b_n^b$ into Q;
  end if
end while
return S

test has the highest false positive ratio (the most subdomains are not excluded from consideration even though they should be). The slowest test has the lowest false positive ratio and is used only for those subdomains, which are not excluded by earlier tests. Details about the tests can be found in section V.

It should be stressed here that in general, de Casteljau division produces $n+1$ smaller simplexes from one input simplex. However, for simplicity and universality of the algorithm regardless of the number of equations, we make subdivision along one consecutive variable x_i, producing 2 smaller simplexes. Subdivision along the consecutive variable x_i has one more important advantage: we are assured that subdomains (diameter of subdomain) will be decreasing. More information about the diameter can be found in the section V. In other words, input simplex is a domain, and two output simplexes are subdomains, which added (in a set theory sense) are equal to input domain. Intersection of interiors of the subdomains is empty. Obtained polynomials, $b_1^a, \dots, b_n^a$ and $b_1^b, \dots, b_n^b$, are the same as input polynomials, but specified to new, smaller (scaled) unit simplex subdomain.

Subdomains not excluded from consideration are enqueued to Q.

Every subdomain dequeued from Q is tested for the end condition. If it is true, the root is calculated and enqueued in S. Thus the subdomain is excluded from consideration. The end condition and the root calculation are the centerpiece of this paper and will be discussed in section V.

When queue Q is empty, algorithm returns queue of solutions S.

In the Fig. 1 we can see an exemplary use of the algorithm. We have a two-dimensional domain (it means $n = 2$, so there are two polynomial equations in a system) and two roots x_0 and x_1 in the domain. The grey area represents a subdomain excluded by tests. We can see that the de Casteljau division is

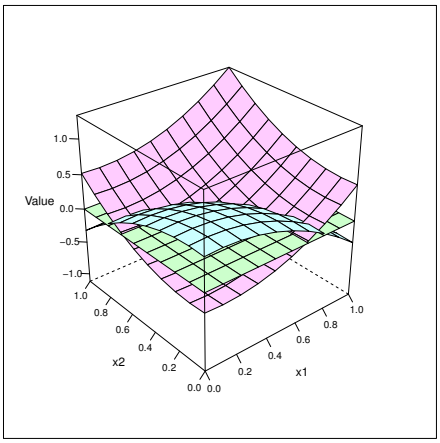


Fig. 2: Extended space representation for 2-dimensional box

performed along the longest edge of subsimplex, so diameters of the consecutive subsimplexes are getting smaller.

IV. Geometrical interpretations

Note that the problem can be geometrically interpreted in two ways.

A. Extended space representation

The extended space representation is used in the end condition and the root calculation. Consider a coordinate system with $n+1$ axes. n of them represent a domain. If we consider polynomial $p_k(\mathbf{x})$, the vector $\mathbf{x}$ has coordinates in those n axes. The $n+1$ axis is the value of the polynomial, so it is the value of $p_k(\mathbf{x})$. It implies that the polynomial $p_k(\mathbf{x})$ creates a surface in the coordinate system, where every point of that surface consists of n coordinates from domain (simplex) and last one which is value of the polynomial in that point.

Solving a system of polynomial equations (and equations in general) is finding points, where all of the surfaces have a zero value in the last, $n+1$, coordinate. The root is a point that consist of first n coordinates of such a point.

In the Fig. 2 we can see an example for n-dimensional box, where $n = 2$. One polynomial in the domain is pink, the second is blue, and value of zero is represented by a green plane.

B. n-dimensional mapping representation

The n-dimensional mapping representation can be applied to any system of functions. We can see the system as mapping $\mathbf{B}\colon S_1^n \to \mathbb{R}^n$. This function maps simplex S_1^n to $\mathbf{B}(S_1^n)$, so coordinate system has n axes.

This geometrical interpretation is used to understand tests that exclude subdomains from consideration. Those tests base on a Bernstein polynomial convex hull property, where coefficients $\mathbf{b}_{(j_1,\ldots,j_n)}$ of the system are the corners. More details can be found in [4].

Please note that:

$$(\mathbf{0} \in \mathbf{B}(S_1^n)) \Leftrightarrow ((\exists \mathbf{x}_0 \in S_1^n)\colon \mathbf{B}(\mathbf{x}_0) = \mathbf{0}) \tag{5}$$

Of course, even though this logical relationship is true, it is not very practical. It would be computationally demanding to check $\mathbf{0} \in \mathbf{B}(S_1^n)$. However, we can use the convex hull property of Bernstein polynomial:

$$(\exists \mathbf{x}_0 \in S_1^n\colon \mathbf{w}(\mathbf{x}_0) = \mathbf{0}) \Rightarrow (\mathbf{0} \in \operatorname{conv}(\{\mathbf{b}_{(j_1,\ldots,j_n)}\colon j_1 + \cdots + j_n \leq N\}) \tag{6}$$

Contraposition, however, of the above may be more convenient in practice:

$$(\mathbf{0} \notin \operatorname{conv}(\{\mathbf{b}_{(j_1,\ldots,j_n)}\colon j_1 + \cdots + j_n \leq N\}) \Rightarrow (\forall \mathbf{x}_0 \in S_1^n\colon \mathbf{w}(\mathbf{x}_0) \neq \mathbf{0}) \tag{7}$$

After obtaining certainty that the convex hull of the system of polynomial equations does not contain $\mathbf{0}$ point, we can see that we are able to exclude that system as it has no root. And if it contains $\mathbf{0}$, we can only suspect that a root is present in the system (because it is only in a convex hull).

We can see an example in the Fig. 3. It is a system of two polynomial equations of second degree. In Fig. 3a we can see a domain of the system and in the 3b there is the same system after mapping $\mathbf{B}(S_1^n)$. Coefficients of the system are marked as circles. Convex hull of the system is a green polygon. Vertices of red polygon are extreme coefficients of the system and they are marked as red circles. It is worth noticing that this system has two roots in the domain and both of them map to $\mathbf{0}$ in $\mathbf{B}(S_1^n)$.

Definition IV.1 (Hyperplane)**.** A hyperplane passing through the point h_0 and defined by a vector $\mathbf{p}$ is a set of points:

$$H^m(\mathbf{p}, \mathbf{h}_0) = \{\mathbf{h} \in \mathbb{R}^m\colon \mathbf{p}^T \cdot (\mathbf{h} - \mathbf{h}_0) = 0, \mathbf{h_0}, \mathbf{p} \in \mathbb{R}^m, \mathbf{p} \neq \mathbf{0}\} \tag{8}$$

where $\cdot$ is a dot product.

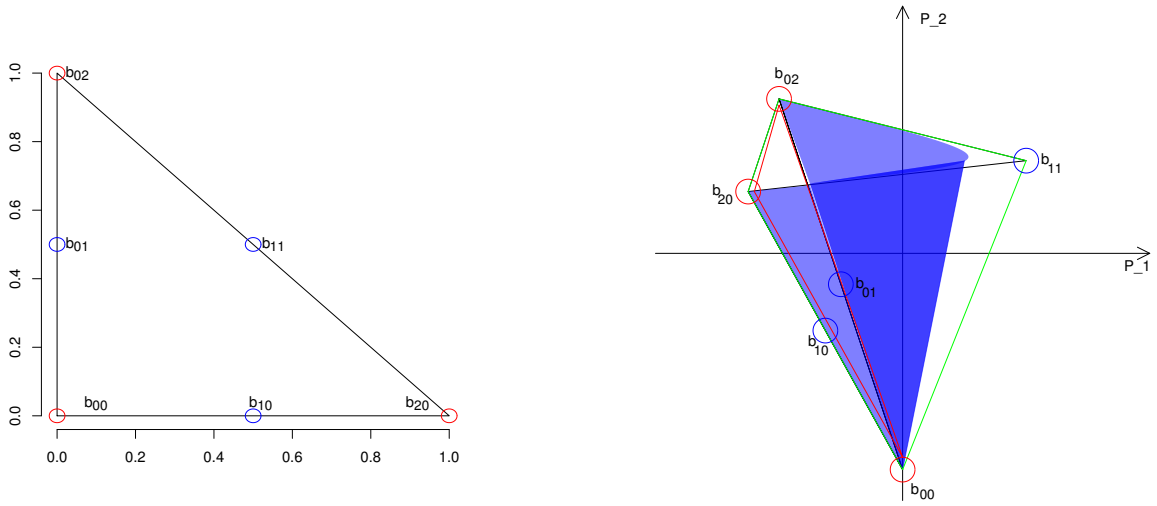
V. Algorithm details

All three described tests are based on hyperplanes. It can be proved that if we can find a hyperplane which separates the convex hull and $\mathbf{0}$, then convex hull does not include $\mathbf{0}$ point. More complicated tests use more interesting (and computationally damanding) hyperplanes.

As we wrote above, all three tests are based on finding a hyperplane that separates the convex hull and $\mathbf{0}$ point. Testing if all vertices of the convex hull are on one side of a hyperplane can be computed by checking a sign of dot product. More details about excluding tests can be found in [4].

A. Test of signs

This test checks hyperplanes $H^n(\mathbf{e}_i, \mathbf{0})$, where $\mathbf{e}_i$ is a base unit vector, for $i = 1, \ldots, n$. It can be performed as checking, if every polynomial in a system has coefficients of both signs, positive and negative. If at least one polynomial has all coefficients positive only or negative only, this test rejects that system as it does not have a root.



(a) S_1^n (b) $\mathbf{B}(S_1^n)$

Fig. 3: n-dimensional mapping representation

B. Test of midpoint

Let $M = \binom{n+N}{N}$ be a number of polynomial's coefficients. This equality can be proved by *Stars and bars* theorem, which can be found in [25].

Midpoint test (which should be performed as a second one) calculates a vector which is an average of Bernstein coefficients of set:

$$\mathbf{p_{mid}} = \frac{1}{M} \sum_{(j_1,\ldots,j_n)\in\{(j_1,\ldots,j_n):j_1+\cdots+j_n\leq N\}} \mathbf{b}_{(j_1,\ldots,j_n)}$$

Test verifies if all coefficients (points) are on one side of hyperplane $H^n(\mathbf{p_{mid}}, 0)$.

C. Test of hyperplanes

The last, most complex test is a side test. Hyperplanes in this test are determined by Bernstein coefficients of the system. We get every n linearly independent points from Bernstein coefficients $\mathbf{b}_{(j_1,\ldots,j_n)}$ and create a hyperplane H^n. Again, coefficients of the system are tested if they are on one side of the hyperplane (exclude) or not (further subdivision is needed).

There are two variants of this test. The first is that we get n linear independent points from extreme coefficients only. The second takes all coefficients into account. We have decided to implement the second variant in our work, as it tests more hyperplanes and can exclude more subdomains.

D. End condition and a root computation

The end condition used in literature is checking a diameter of the subdomain. A diameter, by definition, is the longest side of a simplex. If it is less than tolerance ϵ, algorithm assumes this area has a root and returns the midpoint of the simplex (adds it to the list of solutions). Such an approach can be found in [4], [5], [7].

We have found better approach, where a lower recursion level is needed. Using the de Casteljau division gives us smaller domains and we have observed that polynomials in these smaller domains are getting closer to linear functions.

We decided to check if polynomials on the subdomain are close enough to linear functions. If they are, then we change the problem to a system of linear equations (all polynomials on the subdomain are substituted to linear approximations) and solve it (find intersections of those approximations). Details are discussed below. In this part of the paper we should think of it according to the extended space representation, where every polynomial makes its own surface over the domain.

Definition V.1 (Control points of polynomial in Bernstein form)**.** Control point $\mathbf{b}^{(\mathbf{k})}_{(\mathbf{j_1},\ldots,\mathbf{j_n})}$ corresponding to a coefficient $b^{(k)}_{(j_1,\ldots,j_n)}$ of a polynomial b_k is a point (according to extended space representation) which first n coordinates are coordinates of the corresponding point of the domain (see Fig. 3a) and last $n+1$ coordinate is the value of coefficient $b^{(k)}_{(j_1,\ldots,j_n)}$.

Definition V.2 (Extreme control points of polynomial in Bernstein form)**.** Extreme control point $\mathbf{b}^{(\mathbf{k})}_{\mathbf{i_e}}$ $(i_e \in I_e)$ is a control point corresponding to an extreme coefficient $b^{(k)}_{i_e}$ of a polynomial b_k.

We can see an example in the Fig. 4. It is a surface created by polynomial according to extended space representation. Extreme control points are marked as red and the rest of control points are marked as blue.

After defining extreme control points, we can say that, after concluding that a polynomial is sufficiently linear on a subdomain, we can create a hyperplane passing through the extreme control points of the polynomial and the hyperplane will be an aformentioned linear approximiation of the equation.

The last thing we have to specify is a method to determine

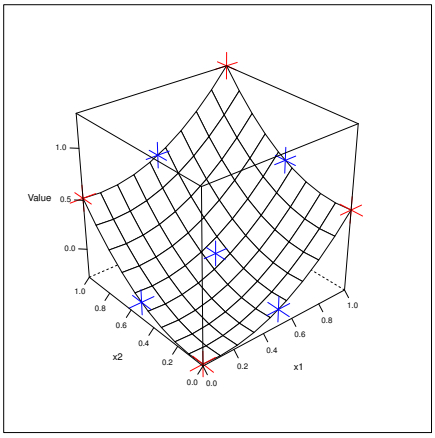


Fig. 4: Control points. Extreme control points are red.

when a polynomial is so similar to a linear function, that we can approximate the polynomial with the linear function with given error ϵ.

It should be stressed that there are $n+1$ extreme control points and we would denote them as ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{0}}^{(\mathbf{k})}$, ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{1}}^{(\mathbf{k})}$, ..., ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{n}}^{(\mathbf{k})}$ for polynomial b_k.

Definition V.3 (Thickness of a polynomial in Bernstein form)**.** Consider extreme control points ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{0}}^{(\mathbf{k})}$, ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{1}}^{(\mathbf{k})}$, ..., ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{n}}^{(\mathbf{k})}$ of polynomial b_k. Vectors ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{1}}^{(\mathbf{k})} - {}^{\mathbf{e}}\mathbf{b}_{\mathbf{0}}^{(\mathbf{k})}$, ..., ${}^{\mathbf{e}}\mathbf{b}_{\mathbf{n}}^{(\mathbf{k})} - {}^{\mathbf{e}}\mathbf{b}_{\mathbf{0}}^{(\mathbf{k})}$ span an n-dimensional subspace. For every control point $\mathbf{b}_{(\mathbf{j_1},\ldots,\mathbf{j_n})}^{(\mathbf{k})}$ (not only *extreme* control points) we can calculate the distance from the subspace by projecting $\mathbf{b}_{(\mathbf{j_1},\ldots,\mathbf{j_n})}^{(\mathbf{k})}$ on this subspace and computing the distance $d_{(j_1,\ldots,j_n)}^{(k)}$ from this point $\mathbf{b}_{(\mathbf{j_1},\ldots,\mathbf{j_n})}^{(\mathbf{k})}$ to its projection. *Thickness* of a polynomial b_k is $\max\{d_{(j_1,\ldots,j_n)}^{(k)} : j_1 + \cdots + j_n \leq N\}$.

E. Calculating thickness

Assume that we have vector $\mathbf{p} \in \mathbb{R}^{n+1}$ and subspace of n-dimensions U. We want to project $\mathbf{p}$ on U. It means $\mathbb{R}^{n+1} = U \oplus V$ where $\dim V = 1$. We say that V is the complementary (orthogonal) subspace to U. That means that $\mathbf{p}$ breaks up into:

$$\mathbf{p} = \mathrm{proj}(\mathbf{p}, U) + \mathrm{proj}(\mathbf{p}, V) \tag{9}$$

So to find $\mathrm{proj}(\mathbf{p}, U)$, we can simply find $\mathrm{proj}(\mathbf{p}, V)$, which is a projection onto a 1-dimensional subspace.

Assume that vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ span U. We want to find vector $\mathbf{v}$ which is orthogonal to those vectors:

$$\begin{aligned} \mathbf{v} \cdot \mathbf{v}_1 &= 0 \\ \mathbf{v} \cdot \mathbf{v}_2 &= 0 \\ &\vdots \\ \mathbf{v} \cdot \mathbf{v}_n &= 0 \end{aligned}$$

We obtain linear system with n equations and $n+1$ unknowns. We have to add one constraint, e.g. $\mathbf{v}$ is unit vector or one of its coordinates is equal to 1. When we have $\mathbf{v}$, we get:

$$\mathrm{proj}(\mathbf{p}, V) = \frac{\mathbf{p} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v} \tag{10}$$

So

$$\mathrm{proj}(\mathbf{p}, U) = \mathbf{p} - \mathrm{proj}(\mathbf{p}, V) = \mathbf{p} - \frac{\mathbf{p} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v} \tag{11}$$

End condition is checking if thicknesses of all polynomials in system are less than ϵ. If so, linear approximations l_k of polynomials b_k are created and system of linear equations is solved. Solution of the linear system is taken as a root of the system of polynomial equations. If not, further subdivision is needed.

VI. Experimental results

The numerical experiments were performed on a PC with Intel Core i5-2500K CPU 3.30GHz and 8 GB of RAM. The goal of our research was to compare time and recursion level for different end conditions and root computation methods.

The input systems of equations differed in the number n of variables and equations (2 or 3) and the degree N (ranging from 1 to 20). Intersection of two or three curves or surfaces is a common case in CAD/CAM applications and that is why we limited our research to 2 or 3 equations.

Four cases were tested:

1) end condition was thickness of polynomials less than ϵ and root computation was solving a linear system of equations (our approach),
2) end condition was diameter of subdomain less than ϵ and root computation was returning midpoint of subdomain (standard approach),
3) end condition was diameter of subdomain less than ϵ and root computation was solving a linear system of equations,
4) end condition was thickness of polynomials less than ϵ and root computation was returning midpoint of subdomain.

Last two scenarios, which are mixture of our and standard approach, did not yield better results. In third case too many divisions were performed for given error ϵ, so time results were unsatisfactory. In fourth case too few divisions were performed, so accuracy of solutions was too small. Therefore results of genuine approaches are presented only.

We present results on Fig. 5–7. The tolerance ϵ for all tests was equal to 10^{-6}. It should be noted that all polynomials in a system were normalized so all coefficients were in set $[-1; 1]$. All polynomials in one tested system are of the same degree.

In Fig. 5 we can see maximal recursion level (number of de Casteljau divisions) needed to compute roots depending on the polynomials' degree. For standard approach it is constant number, because there is always the same number of divisions needed to obtain given diameter equal to ϵ of subdomain. Unlike standard approach, recursion level in our approach varies. Number of divisions depends on degree of polynomials. When polynomial is of higher degree, more recursions are required to get desirable linear approximation. When we use standard end condition, linear system is treated as always and many unnecessary subdivisions are performed.

It can be seen that obtaining the same thickness of hyperplane as diameter of subdomain (equal to ϵ) needs about half the number of recursions. What is more, a number of necessary recursions grows very slowly in degree of polynomials.

Nice feature of our approach is that, when a system of polynomial equations is in fact linear (all polynomials are polynomials of first degree), this case is detected in the first recursion level and system is solved by method dedicated for those systems.

In Fig. 6 we can see how much time it takes to compute roots depending on the polynomials' degree. As we can see, smaller number of recursion results in shorter time of computation. On average, our approach takes 60% time of the standard approach.

It can be seen that the method is good for polynomials of low degree. Detailed time of computation for those polynomials can be seen in Fig. 7. Unfortunately, for higher polynomials' degree time of computation becomes unacceptable: over 0.015 seconds for two equations (up to 0.05 for 20 degree) and over 0.5 seconds for three equations (up to 2.5 seconds for 20 degree).

We compared our approach with well-known application for computations, Mathematica. In Mathematica we used the function NSolve which is designed for numerical computation (Mathematica can perform symbol computations as well). We chose arguments "Reals" and "6", which means we are interested in real roots only and results should have 6 significant digits. What is more, in addition to the polynomial equations, we added inequalities such as $x \geq 0$ or $x + y \leq 1$.

Results of this comparison can be seen in Fig. 8. As we can see, the computation time of Mathematica is incomparable to our method so much we decided to plot it on a logarithmic scale. For example, for polynomials of fifth degree time of our method's computation is 0.0011 seconds, while Mathematica needs 0.0590 seconds. The time difference is even greater as a degree gets bigger and for degree of 16 execution times are 0.0190 and 4505 seconds, respectively.

Mathematica's function NSolve finds all roots in whole domain. We did not find method other than adding inequalities to bound domain. Adding inequalities means that NSolve finds all roots and after that it excludes those, which are not in domain. This is reflected at computation time. It is much worse than our approach. There is no sense using NSolve for polynomials of degree higher than five if we can bound domain in simplex.

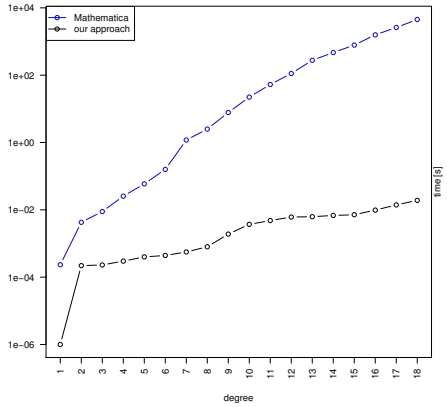


Fig. 8: Comparison of Mathematica and our approach computation time

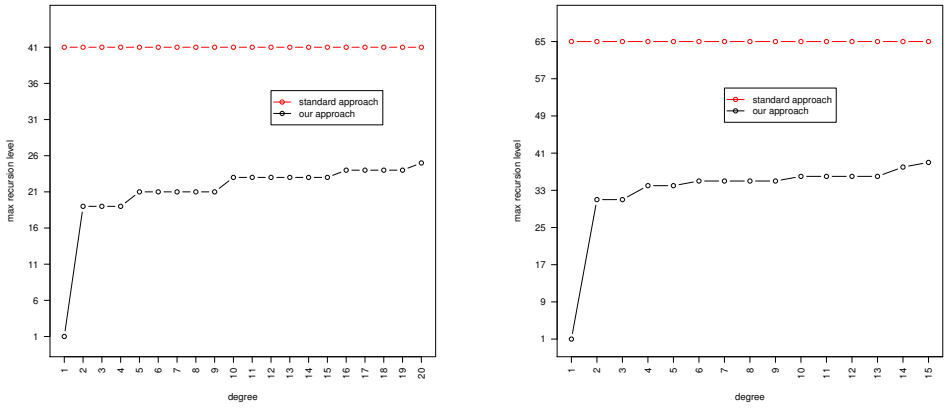
We tried to use NSolve for a system of three polynomial equations, but a difference of computation time was even bigger. For example, for three polynomials of fifth degree time of execution NSolve was 976 seconds.

VII. Conclusion

In this paper a new algorithm isolating the roots of systems of polynomial equations has been introduced. An algorithm based on the general multivariate Bernstein representation and the systematic search of a given domain has been implemented.

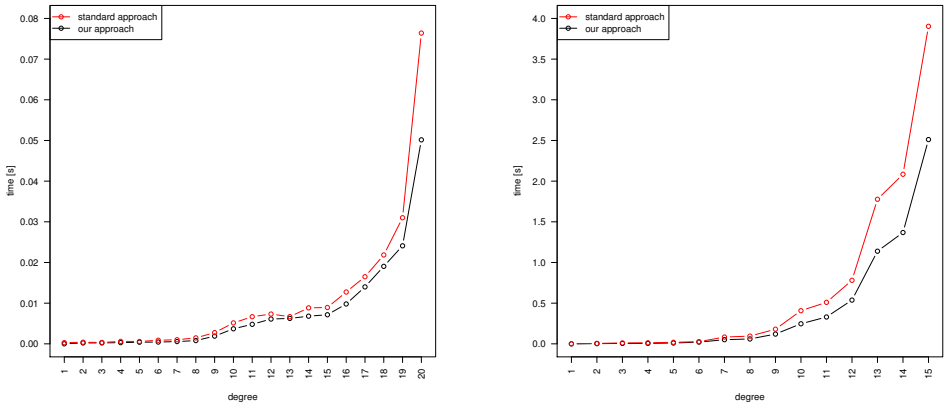
The main objective of this research was verification of applicability of novel end condition and computation of root by linear approximations. It was shown that this novel approach is nearly two times faster than standard methods.

Main advantage of this method is finding all roots in the given domain. Many known methods base on Newton's method which can converge to the solution in another simplex [12], [13]. This implies that some solutions may be missed and some may be found several times. Other methods, e.g. symbolic or homotopic, find all roots in $\mathbb{R}^n$, from which we can choose those in a domain (n-dimensional box or simplex). Unfortunately, this approach unnecessarily increase computation time, especially when most roots are outside of the domain, because many solutions are calculated and after that they are excluded from final set of roots. What is more, our algorithm is one of the fastest methods when there are no roots, because it is detected in the very first step. The next case, for which this method is doing exceptionally well is system of linear equations. In first step *thickness* is equal to zero and system is solved (approximations l_k and polynomials p_k from the system are the same).



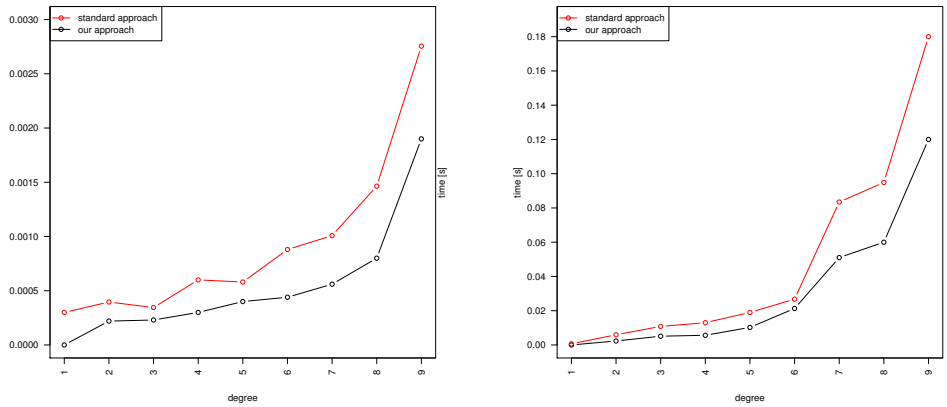
(a) Recursion level for two equations (b) Recursion level for three equations

Fig. 5: Recursion level depending on the polynomials' degree



(a) Computation time for two equations (b) Computation time for three equations

Fig. 6: Computation time depending on the polynomials' degree



(a) Computation time for two equations (b) Computation time for three equations

Fig. 7: Computation time depending on the polynomials' degree for 1–9 degrees

The algorithm has one interesting feature, when system has infinitely many solutions. If given ϵ is not too big, it can calculate some points which are roots of a system, and after connecting those points, we obtain a polyline, which approximates an infinite family of roots. A method for detecting this case in the beginning is needed.

The algorithm is at a disadvantage in the case where a root is in a corner or on a side of a subdomain. That causes many subdivisions and many simplexes to be considered because in many areas tests are passed. Detecting this case and evaluating only one simplex should be another area of exploration. However, it should be noted that a little perturbation of coefficients solves the problem.

A way for further development can be indicated. Some of the equations in the system can be close enough to linear function in earlier iterations of the algorithm than others. So we can substitute those equations to linear approximations. After that, one variable in the system (for one linear approximation) can be ousted. It would be very effective, because it would decrease the dimension of the problem. The algorithm has exponential computational complexity according to number of variables. That is why decreasing the number of variables is so important. One of the way to eliminate variables in a system of polynomial equations is using Gröbner basis.

Summarizing, this is an excellent method for surface intersections or for the visualization of curves and surfaces, because of finding all roots in a given domain. This is a well-conditioned numerical algorithm which can be useful for three to four equations [4]. It has some disadvantages like exponential computational complexity according to number of unknowns or vulnerability to roots in the corners or sides of a domain, but when we exclude those cases, it can be a powerful tool in computer graphics and computer-aided design.

ACKNOWLEDGMENT

Maciej Bartoszuk would like to acknowledge the support by the European Union from resources of the European Social Fund, Project PO KL "Information technologies: Research and their interdisciplinary applications", agreement UDA-POKL.04.01.01-00-051/10-00 via the Interdisciplinary PhD Studies Program.

REFERENCES

[1] T. Sederberg, D. Anderson, and R. Goldman, "Implicit representation of parametric curves and surfaces," *Computer Vision, Graphics, and Image Processing*, vol. 28, no. 1, 1984. doi: 10.1016/0734-189X(84)90140-3. [Online]. Available: http://dx.doi.org/10.1016/0734-189X(84)90140-3

[2] D. Lavender, A. Bowyer, J. Davenport, A. Wallis, and J. Woodwark, "Voronoi diagrams of set-theoretic solid models," *IEEE Computer Graphics and Applications*, pp. 69–77, September 1992. doi: 10.1109/38.156016. [Online]. Available: http://dx.doi.org/10.1109/38.156016

[3] J. Canny, "The Complexity of Robot Motion Planning," *MIT Press*, 1988. doi: 10.1017/S0263574700000151 ACM Doctoral Dissertation Award. [Online]. Available: http://dx.doi.org/10.1017/S0263574700000151

[4] J. Porter-Sobieraj, "Application of simplexes to the solving systems of algebraic equations," *Proc. of the IInd International Conference on Advances in Production Engineering 2001, Oficyna Wydawnicza PW, part II*, pp. 23–32, 2001.

[5] K. Marciniak, E. Pawelec, and J. Porter-Sobieraj, "Method for finding all solutions of systems of polynomial equations," *Proc. 7th IEEE Int. Conf. Methods and Models in Automation and Robotics*, vol. 1, pp. 155–158, 2001.

[6] J. Seland and T. Dokken, "Real-time algebraic surface visualization," *Geometrical Modeling, Numerical Simulation, and Optimization, Springer, Heidelberg*, pp. 163–183, 2007. doi: 10.1007/978-3-540-68783-2_6. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-68783-2_6

[7] J. Porter-Sobieraj and R. Klopotek, "Solving systems of polynomial equations on a GPU," *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on, Series IEEE Computer Society Press(2012)*, pp. 539–544, 2012.

[8] J. Wilkinson, "The evaluation of the zeros of ill-conditioned polynomials. parts i and ii," *Numer. Math., 1:150-166 and 167-180*, 1959. doi: 10.1007/BF01386381. [Online]. Available: http://dx.doi.org/10.1007/BF01386381

[9] J. Chen, S. Lazard, L. Peñaranda, M. Pouget, F. Rouillier, and E. P. Tsigaridas, "On the topology of planar algebraic curves," *Mathematics for Computer Science. Special issue on Computational Geometry and Computer Aided Geometric Design*, vol. 4, no. 1, pp. 113–137, 2010. doi: 10.1145/1542362.1542424. [Online]. Available: http://dx.doi.org/10.1145/1542362.1542424

[10] S. Gao, F. V. IV, and M. Wang, "A new algorithm for computing grobner bases," 2010.

[11] L. Buse, M. Elkadi, and B. Mourrain, "Generalized resultants over unirational algebraic varieties," *J. of Symbolic Computation*, vol. 29, pp. 515–526, 2000. doi: 10.1006/jsco.1999.0304. [Online]. Available: http://dx.doi.org/10.1006/jsco.1999.0304

[12] D. Bini, "Numerical computation of polynomial zeros by means of aberth's method," *Numerical Algorithms*, vol. 13, no. 2, pp. 179–200, 1996. doi: 10.1007/BF02207694. [Online]. Available: http://dx.doi.org/10.1007/BF02207694

[13] B. Mourrain and O. Ruatta, "Relation between roots and coefficients, interpolation and application to system solving," *JSC*, vol. 33, pp. 679–699, 2002. doi: 10.1006/jsco.2002.0530. [Online]. Available: http://dx.doi.org/10.1006/jsco.2002.0530

[14] H.-J. Su, J. McCarthy, M. Sosonkina, and L. Watson, "Algorithm 857: POLSYS GLP: A parallel general linear product homotopy code for solving polynomial systems of equations," *ACM Trans. Math. Softw*, vol. 32, no. 4, pp. 561–579, 2006. doi: 10.1145/1186785.1186789. [Online]. Available: http://dx.doi.org/10.1145/1186785.1186789

[15] J. Verschelde and G. Yoffe, "Polynomial homotopies on multicore workstations," *Proceedings of the 4th International Workshop on Parallel Symbolic Computation (PASCO 2010), ACM*, pp. 131–140, 2010. doi: 10.1145/1837210.1837230. [Online]. Available: http://dx.doi.org/10.1145/1837210.1837230

[16] J. Verschelde, "Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation," *ACM Transactions on Mathematical Software (TOMS)*, vol. 25, no. 2, pp. 251–276, 1999. doi: 10.1145/317275.317286. [Online]. Available: http://dx.doi.org/10.1145/317275.317286

[17] J.-P. Dedieu and J.-C. Yakoubsohn, "Computing the real roots of a polynomial by the exclusion algorithm," *Numerical Algorithms*, vol. 4, no. 1, pp. 1–24, 1993. doi: 10.1007/BF02142738. [Online]. Available: http://dx.doi.org/10.1007/BF02142738

[18] R. B. Kearfott, "Interval arithmetic techniques in the computational solution of nonlinear systems of equations: Introduction, examples and comparisons," *Lectures in Applied Mathemetics. AMS Press*, pp. 337–357, 1990.

[19] V. Pan, "Optimal and nearly optimal algorithms for approximating polynomial zeros," *Computers & Mathematics with Applications*, vol. 31, no. 12, pp. 97–138, 1996. doi: 10.1016/0898-1221(96)00080-6. [Online]. Available: http://dx.doi.org/10.1016/0898-1221(96)00080-6

[20] M. Roy, "Basic algorithms in real algebraic geometry: from Sturm theorem to the existential theory of reals," *Exposition in Mathematics*, vol. 23, pp. 1–67, 1996.

[21] F. Rouillier and P. Zimmermann, "Efficient isolation of a polynomial real roots," *J. Comput. Appl. Math.*, 2003. doi: 10.1016/j.cam.2003.08.015. [Online]. Available: http://dx.doi.org/10.1016/j.cam.2003.08.015

[22] J. Verschelde and G. Yoffe, "Evaluating polynomials in several variables and their derivatives on a GPU computing processor," *IEEE Computer Society*, pp. 1391–1399, 2012. doi: 10.1109/IPDPSW.2012.177. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2012.177

[23] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops (IPDSW 2010) IEEE Computer Society*, pp. 1–8, 2010. doi: 10.1109/IPDPSW.2010.5470941. [Online]. Available: http://dx.doi.org/10.1109/IPDPSW.2010.5470941

[24] W. N. Waggenspack and D. C. Anderson, "Converting standard bivariate polynomials to Bernstein form over arbitrary triangular regions," *Comput. Aided Des.*, vol. 18, no. 10, pp. 529–532, Dec. 1986. doi: 10.1016/0010-4485(86)90040-0. [Online]. Available: http://dx.doi.org/10.1016/0010-4485(86)90040-0

[25] J. Pitman, *Probability*, ser. Springer Texts in Statistics. Springer, 1993. ISBN 9780387979748. [Online]. Available: http://books.google.pl/books?id=L6IWgaCuilwC