Computer-Aided Design 46 (2014) 101–109

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Solving multivariate polynomial systems using hyperplane arithmetic and linear programming

Iddo Hanniel

Technion, Israel Institute of Technology, Haifa, Israel

HIGHLIGHTS

- A new scalable algorithm for solving systems of multivariate polynomials.
- The concept of hyperplane arithmetic, which is used in our algorithm.
- A benchmark of example systems that are scalable in the number of variables.
- Implementation and comparison with previous algorithms.

ARTICLE INFO

Keywords:
Subdivision solver
Multivariate solver
Geometric constraints

ABSTRACT

Solving polynomial systems of equations is an important problem in many fields such as computer-aided design, manufacturing and robotics. In recent years, subdivision-based solvers, which typically make use of the properties of the Bézier/*B*-spline representation, have proven successful in solving such systems of polynomial constraints. A major drawback in using subdivision solvers is their lack of scalability. When the given constraint is represented as a tensor product of its variables, it grows exponentially in size as a function of the number of variables. In this paper, we present a new method for solving systems of polynomial constraints, which scales nicely for systems with a large number of variables and relatively low degree. Such systems appear in many application domains. The method is based on the concept of *bounding hyperplane arithmetic*, which can be viewed as a generalization of interval arithmetic. We construct bounding hyperplanes, which are then passed to a linear programming solver in order to reduce the root domain. We have implemented our method and present experimental results. The method is compared to previous methods and its advantages are discussed.

1. Introduction

Solving polynomial systems of equations is a crucial problem in many fields such as robotics [1], computer aided design and manufacturing [2,3], and many others [4]. This problem, namely finding the roots of a set of multivariate polynomial equations, is a difficult one and various approaches have been proposed for it. Symbolical approaches, such as Gröbner bases and similar elimination-based techniques [5], map the original system to a simpler one, which preserves the solution set. Polynomial continuation methods (also known as homotopy methods [4]) start at roots of a simpler system and trace a continuous transformation of the roots to the desired solution. These methods handle the system in a purely algebraic manner, find all complex and real roots, and give general information about the solution set. Such methods are typically not well-suited if only real roots are required.

1.1. Subdivision methods

In recent years a family of solvers that focuses only on real roots in a given domain has been introduced. These methods are based on subdividing the domain and purging away subdomains that cannot contain a root. Thus, they are known as subdivision methods (sometimes such methods are referred to as exclusion or generalized bisection methods). Given n implicit algebraic equations in n variables,

$$\mathcal{F}_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1, \ldots, n, \tag{1}$$

we seek all $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ that simultaneously satisfy Eq. (1).

A typical frame of a subdivision algorithm for finding the roots of a polynomial system $\boldsymbol{\mathcal{F}}(x_1, \ldots, x_n) = \mathbf{0}$ over an n-dimensional domain box b within some predefined tolerance ϵ goes as follows:

```
**Algorithm**: root_isolation_in_box
**Input**: $\boldsymbol{\mathcal{F}}(x_1, \ldots, x_n)$, Box $b[x_1^{\min}, x_1^{\max}] \times \cdots \times [x_n^{\min}, x_n^{\max}]$
**Output**: list⟨Box⟩ boxes.
```

E-mail address: ihanniel@technion.ac.il.

0010-4485/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cad.2013.08.022

```
(1) If $(\max(x_i^{\max} - x_i^{\min}) < \epsilon)$ append $b$ to output boxes and return.
(2) Evaluate bound interval on $\mathcal{F}$ in $b$.
(3) If bound does not contain 0, return (there is no solution in $b$).
(4) Otherwise: Split $b$ into subdomains, $b_1, b_2$.
(5) root_isolation_in_box ($\mathcal{F}$, $b_1$, boxes).
(6) root_isolation_in_box ($\mathcal{F}$, $b_2$, boxes).
```

There are various modifications and enhancements to this general framework. The no-solution test in step (3) can be enhanced with single solution tests [6,7], which enable stopping the subdivision process earlier. Then, the algorithm can switch to faster numeric methods such as the Newton–Raphson iteration, which converge to a single root. Another common modification performs a more sophisticated domain reduction [8,9] in step (4), which enables to find tighter subdomains that contain roots and therefore accelerates the convergence of the algorithm.

A common approach for subdivision solvers, popular for its simplicity and wide generality, is interval arithmetic [10–12]. In interval arithmetic a value x is represented by a bounding interval $X = [x^{\min}, x^{\max}]$. Let $\mathcal{F}_i(x_1, x_2, \dots, x_n)$ be a scalar function in n unknowns, defined in a box $b = [x_1^{\min}, x_1^{\max}] \times \cdots \times [x_n^{\min}, x_n^{\max}]$. An interval evaluation of $\mathcal{F}_i$ in b is an interval $[F^{\min}, F^{\max}]$ such that $F^{\min} \le F_i \le F^{\max}$ for any value of $(x_1, \dots, x_n) \in b$. That is, an interval evaluation of a function for the box gives an interval that contains all possible values of the function evaluated on points in the box. Therefore, if the interval evaluation of $\mathcal{F}_i \in b$ does not contain zero, then no root can exist in b. This makes it suitable for use in the root isolation algorithm described above. There are many implementations of interval arithmetic software packages [13,14] and in particular the ALIAS library [15] implements interval methods for the determination of real roots of system of equations and inequalities. The main drawback of interval arithmetic is that the bounds given by the interval evaluation are not tight and with every arithmetic operation the looseness may accumulate. Thus, the interval evaluation may give bounds that are too loose to be useful.

1.2. Bézier/B-spline subdivision methods

Subdivision methods that are based on the tensorial Bézier*B*-spline representation [16,2,8,9] give tight bounds for an exclusion test, based on the convex hull property of the basis. They have been implemented in recent years and applied successfully to a wide variety of problem domains [2,3].

In *B*-spline/Bézier subdivision solvers (e.g., [2]), the $\mathcal{F}_i$, $i = 1, \dots, n$, from Eq. (1) are usually represented as *B*-spline or Bézier multivariate scalar functions, i.e.,

$$\mathcal{F}_i = \sum_{i_1} \cdots \sum_{i_n} P_{i_1,\dots,i_n} B_{i_1,k_{i_1}}(x_1) \cdots B_{i_n,k_{i_n}}(x_n), \tag{2}$$

where $B_{i_j,k_{i_j}}$ are the i_j'th k_{i_j}-degree Bézier/*B*-spline basis functions.

Patrikalakis and Sherbrooke [9] exploited the special properties of the Bézier representation for efficient reduction of the subdomain where roots can exist. In their Projected Polyhedron (PP) algorithm the points of the control polyhedron are projected onto two-dimensional planes and the convex hull of their projection is computed. The intersections of the convex hulls are then used to reduce the domain. To achieve more robustness, Maekawa and Patrikalakis [17,18,3] extended the PP algorithm to operate in rounded interval arithmetic. This resulted in the Interval Projected Polyhedron (IPP) algorithm. Mourrain and Pavone [8] proposed a modification of the IPP algorithm so that instead of using the convex hull of the projected control points, the upper and lower envelopes of the projections would be used as control polygons of two new Bézier forms. These Bézier forms still bound the original function from above and below, and therefore can be used as a tighter bound. They use a univariate root solver to find the roots of these Bézier forms, and use them to construct the bounding intervals. Mourrain and Pavone also suggest a preconditioning step that uses an orthogonalization approach, which makes the domain reduction more efficient.

A single-solution test for *B*-spline/Bézier based solvers was proposed in [19]. This termination criterion was based on computing the normal cones of the function using the Bézier or *B*-spline representation. The single solution test is then implemented using a dual hyperplane representation of the normal cones (see [6] for details).

1.3. Limitations of B-spline/Bézier subdivision methods

As noted above, *B*-spline/Bézier subdivision solvers have been used successfully in recent years. However, the usage of the tensor form has a scalability limitation [20,21], which makes it impractical for systems with a large number of variables. It can be seen from Eq. (2), that the *B*-spline/Bézier representation grows exponentially with the number of variables n. Given a multivariate polynomial in $\mathbb{R}^n$ (i.e., of dimension n, with n unknowns $x_1, \dots, x_n$) and degree d, it is typically represented with $O(n^d)$ coefficients using the standard monomial form. However, it will be represented with $O((d+1)^n)$ coefficients using the tensorial *B*-spline/Bézier representation. Thus, the *B*-spline/Bézier representation grows exponentially with n, whereas the monomial representation only grows polynomially in n. Therefore, when the degree d is much smaller than n, the *B*-spline/Bézier representation is not efficient.

Furthermore, in many cases in practice, the actual monomial representation in standard form is sparse and consists of fewer coefficients. For example, representing the constant 1 in monomial form requires just one coefficient, compared to $O((d+1)^n)$ in the dense Bernstein-Bézier representation. Similarly, representing a linear polynomial $a_0 + a_1x_1 + a_2x_2 + \cdots + a_nx_n$ requires n coefficients in the power basis and $O((d+1)^n)$ in the Bernstein-Bézier basis. Due to its exponential growth and its dense representation, using *B*-spline/Bézier subdivision methods is especially problematic for many engineering problems that are characterized by (or can be transformed to) systems of high-dimension (i.e., a large number of variables n) and relatively low degree d. For example, computing the forward kinematics of a parallel robot [1,8] can be transformed to a system of quadratic constraints, but the number of variables grows with each joint of the mechanism.

Little work has addressed the explosion of the *B*-spline/Bézier form for high-dimensional polynomials. Elber and Grandine [20] represent multivariates as expression trees and compute bounds on the expressions using interval arithmetic, to overcome this problem. Their approach is natural for symbolic manipulations of free-form curves and surfaces. It is thus suited to handle problems arising from manipulations of splines with a large number of control points (see [20]). However, the bounds given by the interval arithmetic over the expression tree are not tight. Furthermore, the expression tree structure is not well suited for more advanced domain reduction algorithms such as the Projected Polyhedron algorithm. Fünfzig et al. [21] proposed a method based on linear programming (LP [22]) to address the high-dimensionality problem for quadratic polynomials. They use a linearization of the terms in the polynomials, representing each term of type X_iX_j as a separate variable of an LP problem. Tight bounds on these variables are constructed using Bernstein polyhedra (see [21] for details) and these inequalities are solved using an LP solver, resulting in a domain reduction. While their method is successful in handling relatively high-dimensional systems of quadratic multivariate polynomials, it is not easily extended to higher degrees. Furthermore, the LP problem constructed by this method is relatively large since the number of variables depends on the number of terms in the problem, which is quadratic in the general case.

In this paper, we will present a new method for solving systems of multivariate polynomials, which scales nicely for systems with a large number of variables and relatively low degree. In Section 2,