Global Solutions of Well-constrained Transcendental Systems using Expression Trees and a Single Solution Test

Maxim Aizenshtein[a], Michael Bartoň[a], Gershon Elber[a]

[a] *Technion – Israel Institute of Technology, Haifa 32000, Israel*

Abstract

We present an algorithm which is capable of globally solving a well-constrained transcendental system over some sub-domain $D \subset \mathbb{R}^n$, isolating all roots. Such a system consists of n unknowns and n regular functions, where each may contain non-algebraic (transcendental) functions like sin, exp or log. Every equation is considered as a hyper-surface in $\mathbb{R}^n$ and thus a bounding cone of its normal (gradient) field can be defined over a small enough sub-domain of D. A simple test that checks the mutual configuration of these bounding cones is used that, if satisfied, guarantees at most one zero exists within the given domain. Numerical methods are then used to trace the zero. If the test fails, the domain is subdivided. Every equation is handled as an expression tree, with polynomial functions at the leaves, prescribing the domain. The tree is processed from its leaves, for which simple bounding cones are constructed, to its root, which allows to efficiently build a final bounding cone of the normal field of the whole expression. The algorithm is demonstrated on curve-curve intersection, curve-surface intersection, ray-trap and geometric constraint problems and is compared to interval arithmetic.

Keywords: non-algebraic equation system, root solver, single root criteria, expression tree, bounding cone

1. Introduction and Previous Work

Solving nonlinear algebraic and/or transcendental systems of equations is a crucial problem in many fields such as computer-aided design, manufacturing, robotics, kinematics and many others. Robust and efficient algorithms that solve such systems are in strong demand. For instance, the problem of intersecting a parametric space curve with a parametric surface leads to a system consisting of three equations and three unknowns. Similarly, the problem of computing the closest point(s) on a curve or surface to a given point leads to a *well-constrained*

polynomial/transcendental system (see, e.g., [19, 20]). In these and similar applications, all solutions of a system of equations within a certain domain D, which is typically a box in $\mathbb{R}^n$, are sought for.

For *polynomial* systems, various methods exist. The symbolically oriented approaches like *Gröbner bases* and similar elimination-based techniques [5] map the original system to a simpler one, preserving the solution set. *Polynomial continuation methods* start at roots of a suitable simple system and transform it continuously to the desired one [17]. These methods handle the system in a purely algebraic manner and give a general information about the solution set. These methods are typically not well-suited if only *real* roots are required.

Contrary to this, a family of solvers which focuses only on real roots has been introduced. These subdivision based schemes handle the polynomials as hyper-surfaces in $\mathbb{R}^n$ and exploit the convex hull property of its Bernstein-Bézier representations [6, 10, 12, 13, 15, 16]. The domain is subdivided, sub-domains which can not contain a root are clipped away and (a set of) sub-domain(s) which may contain roots are returned. The subdivision is usually stopped if some numerical threshold is reached. In [10], a termination criterion which guarantees *at most one* root within a sub-domain has been proposed for isolating roots. Many others polynomial root-finding techniques exist. One survey can be found in [11].

In contrast, for the case of *transcendental* solvers, schemes which also support trigonometric and transcendental terms, the literature is not so extensive. Local root tracing techniques, such as Newton-Raphson iterations, are clearly employed once a close-to-a-root guess is available. One family of solvers is based on the reduction of the original n-dimensional system to one-dimensional nonlinear equations, see [8] and related work cited therein. Every function of the system is evaluated at $n-1$ variables and solved with respect to the remaining one. The root of the univariate function and the partial derivatives of f_i, $i = 1, \ldots, n-1$ in the root are then used as the coefficients of the linear system which computes the improvement of the first $n-1$ values. The process is iteratively repeated, converging quadratically to the root. Even though these methods use a reduction to a single equation, a good initial guess of the root is again needed and the detection of all the roots is not guaranteed.

Another iterative method was proposed in [9]. Every function of the system is considered as an *objective* function. The goal is to minimize these functions and the problem is essentially reduced to a multiobjective optimization problem. An evolutionary algorithm is proposed and a sequence of candidates that approximate the root is created. Again, similarly to [8], the process is iterative and a good initial guess is required to reach the root.

A different subdivision based approach that handles *some* transcendental systems was presented in [7]. The method presented therein is capable of solving an Extended Chebyshev (EC) systems and is well suited for systems consisting of exp function(s). In contrast, "if sin or cos functions are involved, its difficult to decide whether the system is EC or not." [7], (p. 94, section 4.3 and p. 82, 1st paragraph of Section 4.2). Making this decision fully automatic is an even more complex task. In contrast, our approach requires to subdivide only

polynomial leaves that contain the variable which needs to be subdivided (and can be therefore efficiently achieved by using deCasteljeau algorithm). The numerical subdivision in [7] is based on multiplying a subdivision matrix, which is expensive and the numerical stability is guaranteed only for low-dimensional systems, see [7], (p. 82).

Another family of subdivision based solvers that support also transcendental functions relies on *interval arithmetic* [14]. These methods typically construct an interval bound on values that given function may attain over given domain. If zero is outside of the bound of some function of the system, the particular domain is discarded. Again, these schemes are difficult to guarantee numerical stability during subdivision and no root isolations are offered.

A major drawback of many subdivision solvers stems from its exponential dependency on the dimension of the problem. In [6], an alternative representation of the equations, in a form of expression trees, is shown to present only polynomial dependency. Herein, we exploit another advantage of expression trees [1], and show how they can be used to find the roots of sets of transcendental functions.

In this paper, an extension of [4], we present a "divide and conquer" algorithm which is capable of solving *transcendental* (non-algebraic), *well-constrained* system over some domain $D \subset \mathbb{R}^n$. Such a system consists of n unknowns and n regular functions, where each may contain transcendental functions like sin, exp or log. Every equation is considered as a hyper-surface in $\mathbb{R}^n$ and thus a bounding cone of its normal (gradient) field can be defined over a small enough sub-domain of D. The termination criterion of [10] is exploited to check the mutual configuration of these bounding cones which, if satisfied, guarantees at most one zero within the given sub-domain, and hence offers a robust scheme to isolate all roots, globally. A multivariate Newton-Raphson method is then used to converge to the zero. Moreover, such a condition guarantees that the subdivision is not terminated until *all* roots are isolated, with the possibility of terminating at the permissible subdivision tolerance, in cases such as multiple roots.

The rest of the paper is organized as follows. Section 2 briefly recalls some notions, including transcendental systems, single solution criterion, expression trees and interval arithmetic. In section 3, the introduced solver that is capable of handling transcendental systems is presented. The construction of bounding cones of differential spaces of transcendental functions is explained and its arithmetic is introduced. In section 4, the application of the proposed solver is demonstrated on curve-curve and curve-surface intersection problems, the ray-trap problem [1] and the geometric constraint problem and is also compared to interval arithmetic. Finally, Section 5 identifies some possible future improvements of the presented method and concludes.

[1] A ray is considered trapped if it bounces forever between k entities in a loop, bouncing from the ith object toward the $i+1$st object, in a cycle.

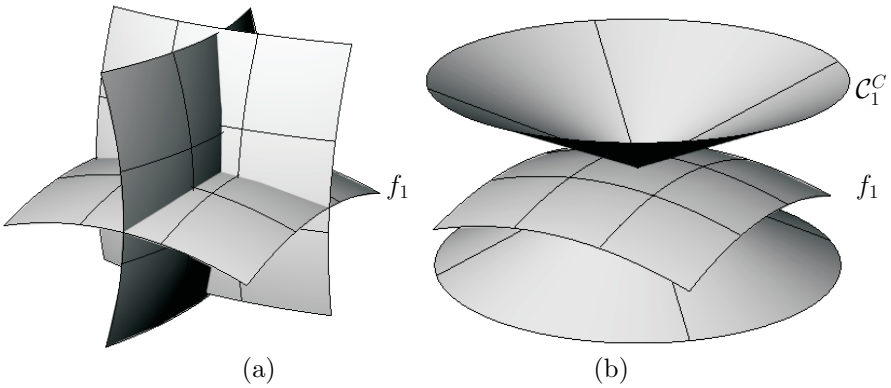


Figure 1: (a) System (1) for $n = 3$, with single solution over some domain $D \subset \mathbb{R}^n$. (b) Complementary (tangent) circular bounding hyper-cone $\mathcal{C}_1^C$ of hyper-surface $f_1 = 0$.

2. Preliminaries

The presented solver exploits both the expression trees representation [6] and the single solution termination test of [10]. A brief survey on these topics will be given. In 2.1, a non-algebraic system is defined and the single root termination criterion for such a system is formulated in 2.2. The notion of expression trees is recalled in 2.3, and in 3.2 the no-root exclusion test and interval arithmetic are introduced.

2.1. Well-constrained Transcendental Systems

Definition 1. *Function* $f : \mathbb{R} \to \mathbb{R}$ *is* algebraic *over* $\mathbb{Q}$ *if there exist a polynomial* $p(x, y)$ *with integer coefficients* y *such that* $p(x, f(x)) = 0$. *Functions which are non-algebraic are called* transcendental.

Definition 2. *Consider the mapping* $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$, *such that at least one component* f_i, $i = 1, \ldots, n$ *of* $\mathcal{F}(\mathbf{x}) = \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})\}$ *is a transcendental function in variables* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. *Then, every solution* $\mathbf{x}$ *of the system,*

$$\mathcal{F}(\mathbf{x}) = 0, \tag{1}$$

is called a root *of* $\mathcal{F}$ *and the set of all roots is known as the* zero set *of the* transcendental mapping $\mathcal{F}$. *The determinant of Jacobian matrix,* $(\frac{\partial f_i}{\partial x_j}(\mathbf{x}))$, *is referred to as a* Jacobian *of System (1) at* $\mathbf{x}$.

In general, System (1) has a zero set of dimension zero. In such a case, we say that the system is *well-constrained*. We assume System (1) is well-constrained in some sub-domain $D \subseteq \mathbb{R}^n$ and $\mathbf{a} = (a_1, a_2, \ldots, a_n) \in D$ is a root. Additionally, we assume that Jacobian of System (1) never vanishes in (the vicinity of) any of its root (the root is regular). In the singular case, the proposed solver will only subdivide to as close as prescribed neighborhood of the root.

If there is a guarantee that $\mathbf{a}$ is the *only root* of (1) in D, some numerical technique, like the multivariate Newton–Raphson method [18], can be used to try to numerically improve the root. This stage, though, may not always converge, a case which will entail further subdivisions.

2.2. Single Solution Termination Criterion for Transcendental Systems

In [10], a single solution criterion was formulated for (piecewise) *polynomial* systems. Considering every $f_i(\mathbf{x})$, $i = 1, \ldots, n$, from System (1) as hyper-surface in $\mathbb{R}^n$, its bounding hyper-cone of the normal field, and subsequently bounding hyper-cone of the complementary (tangential) field, was created. From the mutual position of all n complementary hyper-cones, an existence of at most one root can be determined. See Fig. 1 and [10] for more.

Since this idea is general, regardless of the type of the system (polynomial or transcendental), we adopt this approach and, in a similar manner, test the mutual position of all corresponding tangent bounding hyper-cones. Obviously, the construction of these hyper-cones, unlike the polynomial case, can not be accomplished from the control points of hyper-surfaces $f_i(\mathbf{x}) = 0$ (there are no control points anymore) and it will be explained later, in Section 3. Since the proposed technique is based on the bound of normal fields (gradients), all hyper-surfaces are required to be regular and C^1 continuous over the domain of interest. We start by formulating two definitions:

Definition 3. *Consider implicit function $f_i(\mathbf{x}) = 0$, $\mathbf{x} \in \mathbb{R}^n$ $i = 1, \ldots, n$ over some (rectangular) sub-domain $D \subset \mathbb{R}^n$. We define the* normal field *of the implicit function $f_i(\mathbf{x}) = 0$ over sub-domain D by*

$$\mathcal{N}_i = \{\nabla f_i(\mathbf{x}), \mathbf{x} \in D\}, \tag{2}$$

where $\nabla f_i(\mathbf{x}) = (\frac{\partial f_i}{\partial x_1}, \frac{\partial f_i}{\partial x_2}, \ldots, \frac{\partial f_i}{\partial x_n})$ is the gradient of f_i.

Definition 4. *Consider circular hyper-cone in $\mathbb{R}^n$ with the axis in the direction of unit vector $\vec{v}_i$ and an opening angle α_i as*

$$\mathcal{C}_i^N(\vec{v}_i, \alpha_i) = \{\vec{u} | \langle \vec{u}, \vec{v}_i \rangle = \|\vec{u}\| \cos \alpha_i\}, \tag{3}$$

We say that $\mathcal{C}_i^N$ is a bounding normal hyper-cone *of function f_i if*

$$\langle \vec{u}, \vec{v}_i \rangle \geq \|\vec{u}\| \cos \alpha_i, \quad \forall \vec{u} \in \mathcal{N}_i. \tag{4}$$

By a complementary (or tangent) bounding hyper-cone*, $\mathcal{C}_i^C$, we denote*

$$\mathcal{C}_i^C(\vec{v}_i, \alpha_i) = \mathcal{C}_i^N(\vec{v}_i, 90^{\mathrm{o}} - \alpha_i). \tag{5}$$

The tangent bounding hyper-cone, Eq. (5), bounds the complements of the normal vectors, i.e. the tangent hyperplanes. Therefore, it contains all vectors perpendicular to those bounded by the normal hyper-cone.

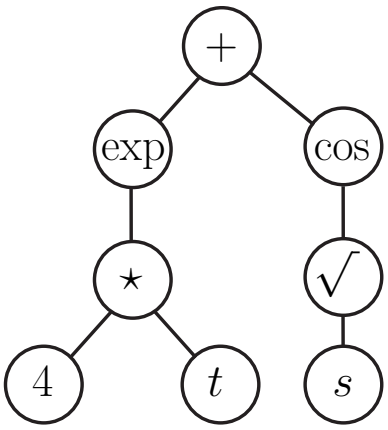


Figure 2: Binary tree for $f(s,t) = e^{4t} + \cos(\sqrt{s})$. The bounding normal cone of the whole expression f is constructed by parsing the tree from the leaves (lower row) to the upper node, the root, applying the bounding cones' arithmetic.

Remark 1. In the remainder of the paper, if no misunderstanding can occur, we call the circular bounding normal hyper-cone as *bounding normal cone* and the circular complementary bounding hyper-cone as *bounding tangent cone.*

Theorem 1. *[10] Given n implicit hyper-surfaces $f_i(\mathbf{x}) = 0$ together with bounding tangent cones $\mathcal{C}_i^C$, $i = 1, \ldots n$ in $\mathbb{R}^n$, there exist at most one common solution of System (1) if*

$$\bigcap_{i=1}^{n} \mathcal{C}_i^C = \{\mathbf{0}\}, \tag{6}$$

where $\mathbf{0}$ *is the origin of the coordinate system.*

2.3. Expression Trees

We recall the notion of a *binary tree* as a structure that uniquely corresponds to some, not necessarily algebraic, expression. The leaf *nodes* of the tree are, for instance constants and variables, expressed as polynomial parametric forms and the interior nodes are unary/binary operators, including, in this case, transcendental functions. In the case of a binary operator, its two sub-nodes are the two operands whereas an unary operator is descended by only one sub-node, see Fig. 2.

In the context of solving transcendental System (1), we have n functions (represented as expression trees) $f_i(\mathbf{x})$ and our aim is to construct a bounding normal cone of every $f_i(\mathbf{x})$ in order to decide whether there is a single zero inside some sub-domain D. For every particular tree, $f_i(\mathbf{x})$, we start to construct bounding normal cones bottom-up from every leaf and, using the bounding cones' arithmetic described in Section 3, the tree is parsed all the way up to the root of the tree, resulting with the bounding cone of the whole expression of $f_i(\mathbf{x})$.

2.4. Interval Arithmetic's Solver and No-Root Exclusion Test

Since interval arithmetic [14] is one alternative to be used for solving non-polynomial systems (and the proposed method will be compared to it in Section 4), we briefly introduce this method.

In order to eliminate domains which contain no roots (no root exclusion test), the sign variation of every function $f_i(\mathbf{x})$, $i = 1, \ldots, n$, is tested over a given domain D. If $f_i(\mathbf{x}) > 0$, (or $f_i(\mathbf{x}) < 0$) for some i, for all $\mathbf{x} \in D$, D is discarded. For any polynomial leaf, this test is easily achieved by checking the signs of the corresponding coefficients. If all are positive (negative), the convex hull property guarantees that no roots exist. For every f_i, all its polynomial leaves are tested and the *minimum* and *maximum* of the coefficients define a bounding interval of values that the (polynomial) leaves may attain. Since f_i is treated as an expression tree, an interval arithmetic is applied at every interior node of the tree, giving a new bound on the merged expression. The tree is parsed from its leaves to the root, only to provide a bound on f_i itself.

Interval arithmetic-based solver works as follows. Given the System (1) over some domain D, subdivide D until either

- some f_i is completely positive (negative) and the subdomain is discarded, or
- the subdivision tolerance is reached and the domain is proclaimed as a *root-containing*.

Then, the numerical improvement stage, see Section 3.3, is invoked.

Observe that the answer in the later case is "may-be a root" which means, unlike the proposed method that exploits the single solution test, that there may exist several roots inside the domain.

Also observe that the subdivision scheme converges only linearly to a single root. If a high subdivision accuracy is required, a large number of subdivisions is expected. On the other hand, the approach based on the single solution test can isolate single roots much earlier whenever possible and thus allows an earlier robust invocation of some numeric improvement scheme.

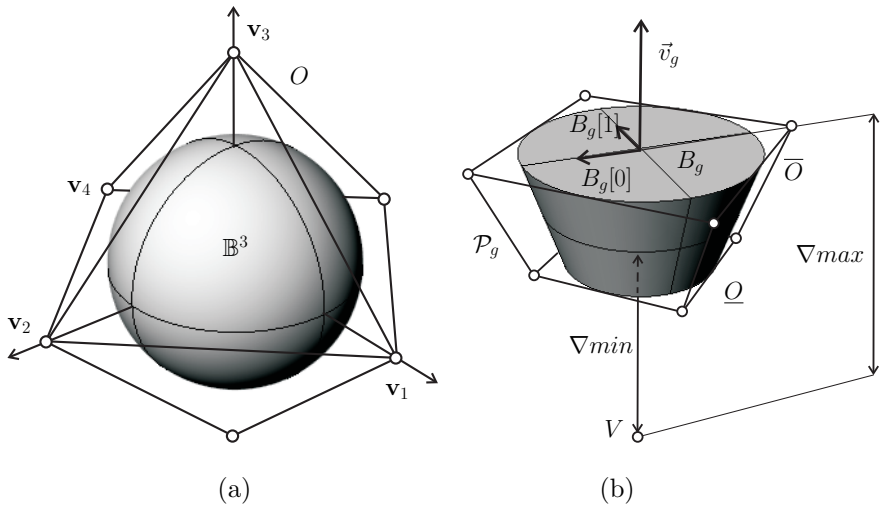
3. Bounding Cones' Construction and Arithmetic

In this section, we explain how the bounding normal cone of an expression – an interior node of an expression tree – is constructed, and introduce the bounding cones' arithmetic which is used when two sub-trees are merged together at some binary (or even unary) node.

3.1. Truncated Bounding Cones and their Bounding Polytopes

Definition 5. *Consider bounding normal cone, $\mathcal{C}_g^N(\vec{v}, \alpha)$, of g, over some sub-domain $D \subset \mathbb{R}^n$. Similarly, consider two positive numbers ∇min and ∇max, such that for all $\mathbf{x} \in D$,*

$$\nabla min \leq \|\nabla g(\mathbf{x})\| \leq \nabla max, \tag{7}$$



(a) (b)

Figure 3: (a) $n = 4$, a cap of a truncated cone in $\mathbb{R}^4$, ball $\mathbb{B}^3$, with its wire-frame bounding orthoplex (an octahedron for $\mathbb{B}^3$) consisting of $2(n-1) = 6$ axis-aligned vertices. (b) $n = 3$, truncated normal cone in $\mathbb{R}^3$ with axis $\vec{v}_g$ and apex V at the origin. The orthogonal complement of $\vec{v}_g$, B_g, and its orthonormal basis $\{B_g[0], B_g[1]\}$ is computed to construct a pair of bounding orthopleces $\underline{O}$, $\overline{O}$ and subsequently the bounding polytope $\mathcal{P}_g$.

holds and

$$\langle \nabla g, \vec{v} \rangle > \alpha \|\nabla g\|, \tag{8}$$

where α is the spanning angle of $\mathcal{C}_g^N$. We further denote by $\overline{\mathcal{C}_g^N} = (\mathcal{C}_g^N, \nabla min, \nabla max)$ the truncated bounding normal cone *of function g.*

Observe that the bounding normal cone, as defined in Def. 4, always contains the origin of the coordinate system (the apex of the cone). As will be seen from the definition of arithmetic operations on bounding cones, a tighter bound which does not contain the origin, is needed. The truncated cone defined in Def. 5 bounds both the direction and the magnitude of the gradients of g, see Fig. 3(b).

The *upper* and *lower caps* of $\overline{\mathcal{C}_g^N}$ (see Fig. 3) are $n-1$ dimensional balls, since their boundaries are $n-2$ dimensional spheres, the result of intersections of the hyper-cone with two hyper-planes perpendicular to its axis.

In order to perform operations with truncated normal cones, we now introduce polygonal bounding regions to these cones, which are referred to as *bounding polytopes*. Such a polytope follows the shape of the truncated cone, conservatively bounds it, and is easy to construct once a polygonal bound on both caps of the cone are given. Direct operations on (exact) truncated cones would be very difficult to handle, whereas these polytopes discretize the problem to treating only a finite number of points (the vertices of the polytope).

Definition 6. *Consider an* $(n-1)$*-dimensional ball,* $\mathbb{B}^{n-1}$*, of radius* r*. An* orthoplex *[21]* O *of ball* $\mathbb{B}^{n-1}$ *is the set*

$$O = \{\mathbf{x} \in \mathbb{R}^{n-1}, \|\mathbf{x}\|_1 \leq r\sqrt{n-1}\}, \tag{9}$$

where $\|.\|_1$ *is the* L^1 *norm.*

Lemma 1. *The orthoplex* O *from Def. 6 bounds* $\mathbb{B}^{n-1}$*.*

Proof 1. *By definition, all the points of* $\mathbb{B}^{n-1}$ *satisfy* $\|\mathbf{x}\|_2 \leq r$*. Due to the equivalence of norms in a finite dimensional space,* $\|\mathbf{x}\|_1 \leq \lambda\|\mathbf{x}\|_2$*, for some* $\lambda \in \mathbb{R}^+$*. Hölder inequality*

$$\sum_{i=1}^{n-1} |x_i y_i| \leq (\sum_{i=1}^{n-1} x_i^2)^{\frac{1}{2}} \cdot (\sum_{i=1}^{n-1} y_i^2)^{\frac{1}{2}}, \tag{10}$$

for $y_i = 1$*,* $i = 1, \ldots, n-1$ *gives* $\lambda = \sqrt{n-1}$*, which yields in* $\|\mathbf{x}\|_1 \leq r\sqrt{n-1}$*.* □

Obviously, the bounding orthoplex O was defined in a way that it bounds $\mathbb{B}^{n-1}$. Note the advantage that O possesses only $2(n-1)$ vertices, compared to another natural bounding region – the $(n-1)$-dimensional cube, which consists of 2^{n-1} vertices. Since the operations on truncated cones are reduced to the vertices of bounding polytopes, the *linear growth* with respect to the dimension is definitely beneficial.

The construction of an orthoplex is straightforward. Consider ball $\mathbb{B}^{n-1}$ with its center at the origin of the Cartesian system. Then, all the vertices $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_{2n-2}$ of the bounding orthoplex are located on the $n-1$ axes, at a distance of $\pm r\sqrt{n-1}$ from the origin, see Fig. 3(a), so they can be expressed as all permutations over $(\pm r\sqrt{n-1}, 0, \ldots, 0)$.

Then, ball $\mathbb{B}^{n-1}$, along with all the vertices $\mathbf{v}_i$ of its bounding orthoplex, is transformed (rotated and translated) from the origin-related position to the proper location such that it caps the truncated cone. In order to achieve this transformation, the destination position of the system is needed. Since the ball's basis is the orthogonal complement of the cone's axis, see Fig. 3, the construction of the basis is achieved by a Gramm-Schmidt process.

Definition 7. *Let* $\overline{\mathcal{C}_g^N}$ *be the truncated bounding normal cone of* g *and let* $\underline{O}$*,* $\overline{O}$ *be the bounding orthopleces of the lower and upper caps, respectively. The convex hull of* $\{\underline{O}, \overline{O}\}$ *is referred to as the* bounding polytope *of* $\overline{\mathcal{C}_g^N}$*, denoted by* $\mathcal{P}_g$*, see Fig. 3.*

Apparently, the closed polyhedron $\mathcal{P}_g$ that bounds $\overline{\mathcal{C}_g^N}$ consists of (at most) $4(n-1)$ vertices, as the convex hull of $\underline{O}$ and $\overline{O}$, each of which has $2(n-1)$ vertices. Therefore, the arithmetic operations on the bounding truncated cones can be efficiently accomplished on their bounding polytopes.

3.2. Bounding Cone's Arithmetic

We introduce the arithmetic rules for the computation of the resulting bounding cone at a node of an expression tree. In this work, we consider the following binary operations: summation, subtraction, product, inner-product and transcendental functions: sin, cos, exp, log and square-root, that can act at any node. The idea is general and can be applied to arbitrary trigonometric function. However, current implementation handles only the above mentioned functions.

Let f and g be two expressions, two neighboring sub-trees that are being merged at some node of an expression tree, and let $\overline{\mathcal{C}_f^N}(\vec{v}_f, \alpha_f)$ and $\overline{\mathcal{C}_g^N}(\vec{v}_g, \alpha_g)$ be their truncated bounding normal cones. Sections 3.2.1 to 3.2.5 explain the action taken for the different operators, as part of the execution of the bounding cones' arithmetic.

3.2.1. Addition

Let $h = f + g$ and let $\overline{\mathcal{C}_h^N}$ be the sought truncated normal cone of h. As we already mentioned, the exact construction of $\overline{\mathcal{C}_h^N}$ from $\overline{\mathcal{C}_f^N}$ and $\overline{\mathcal{C}_g^N}$ would be complicated. Instead, we use their bounding polytopes $\mathcal{P}_f$ and $\mathcal{P}_g$, as the following holds

$$\overline{\mathcal{C}_h^N} \subseteq \overline{\mathcal{C}_f^N} \oplus \overline{\mathcal{C}_g^N} \subseteq \mathcal{P}_f \oplus \mathcal{P}_g \subseteq \mathcal{C}^*, \tag{11}$$

where the $\oplus$ denotes the Minkowski sum, and $\mathcal{C}^*$ is a cone that contains the Minkowski sum of both polytopes. This construction is reduced to only adding all possible pairs of vertices of both polytopes. Algorithm 1 summarizes this process. An explanation of some of its steps follows in more detail:

- Since each of the bounding polytopes consists of at most $4(n-1)$ vertices, their Minkowski sum is obtained by processing all possible pairs, $16(n-1)^2$ in all. See line 1.1.
- $\nabla minmax$ is a vector of size 2 holding $(\nabla min, \nabla max)$.
- The radii of the orthopleces is set by $\sqrt{n-1}\tan(\alpha)$ and $\overline{\mathcal{C}^N}.\nabla minmax[i]$ in lines 1.7 and 1.9 and lines 1.10 and 1.11, respectively.
- In lines 1.2 and 1.3, the orthogonal complements of both axis vectors are constructed. These orthonormal bases are used to build vertices $\vec{\mathbf{v}}_f$, $\vec{\mathbf{v}}_g$ of the bounding polytopes, in lines 1.10 and 1.11.
- Bounding cone $\mathcal{C}^*$ that contains the Minkowski sum of both polytopes is computed, possibly using the method of [2], and returned in line 1.13.

3.2.2. Subtraction

Since

$$\nabla(f - g) = \nabla f + \nabla(-g) \tag{12}$$

and the bounding cone of $-g$ is achieved simply by flipping $\mathcal{C}_g$, the problem is easily reduced to addition.

Algorithm 1: AddNormalCone($\overline{\mathcal{C}_f^N}, \overline{\mathcal{C}_g^N}, n$)

input : $\overline{\mathcal{C}_f^N}(\vec{v}_f, \alpha_f)$, $\overline{\mathcal{C}_g^N}(\vec{v}_g, \alpha_g)$, truncated normal cones of f and g; n, dimension;
output: $\mathcal{C}^*$, bounding normal cone of $f + g$, see Eq. (11);

PointsList $\leftarrow$ List to hold $16(n-1)^2$ elements;
OrthoSystem$B_f \leftarrow$ `HyperplaneOrthoSystem`($\vec{v}_f$, n);
OrthoSystem$B_g \leftarrow$ `HyperplaneOrthoSystem`($\vec{v}_g$, n);
for $i \leftarrow 0$ **to** 1 **do**
  for $j \leftarrow 0$ **to** 1 **do**
    for k $\leftarrow 1$ **to** $n-1$ **do**
      $\vec{u}_f \leftarrow \sqrt{n-1}\tan(\alpha_f) \cdot$ OrthoSystemB_f[k];
      for m $\leftarrow 1$ **to** $n-1$ **do**
        $\vec{u}_g \leftarrow \sqrt{n-1}\tan(\alpha_g) \cdot$ OrthoSystemB_g[m];
        $\vec{\mathbf{v}}_f \leftarrow \overline{\mathcal{C}_f^N}.\nabla minmax[i] \cdot (\vec{v}_f \pm \vec{u}_f)$;
        $\vec{\mathbf{v}}_g \leftarrow \overline{\mathcal{C}_g^N}.\nabla minmax[j] \cdot (\vec{v}_g \pm \vec{u}_g)$;
        `AppendToList`(PointsList, $\vec{\mathbf{v}}_f + \vec{\mathbf{v}}_g$);
$\mathcal{C}^* \leftarrow$ `BoundingConeOfVectors`(PointsList);
return $\mathcal{C}^*$;

3.2.3. Scaling

Scaling (scalar multiplication) by a non-zero coefficient $\lambda \in \mathbb{R}$ is achieved by scaling all polytope's vertices.

3.2.4. Multiplication

Since the gradient of a product is

$$\nabla(f \star g) = g \star \nabla f + f \star \nabla g, \tag{13}$$

the bounding cone is obtained by applying the above discussed operations, where $g \cdot \nabla f$ is accomplished by computing the minimum and maximum of g, in the sub-domain D. Note that a constant sign of both f and g over D is required. If not, the bounding cone would span whole $\mathbb{R}^n$ and the solver subdivides in that case.

3.2.5. Transcendental Functions

exp: Since the exponential function attains only positive values and

$$\nabla(e^f) = e^f \cdot \nabla f, \tag{14}$$

this case is similar to scaling by $\min_{\mathbf{x} \in D} e^{f(\mathbf{x})}$ and $\max_{\mathbf{x} \in D} e^{f(\mathbf{x})}$.

log: We obtain

$$\nabla(\log f) = \frac{1}{f} \cdot \nabla f, \tag{15}$$

the problem is again reduced to scaling and f is required to have a strictly monotone sign on D.

sin & **cos:** Analogously,

$$\nabla(\sin f) = \cos f \cdot \nabla f, \tag{16}$$

and scaling by constants $\max_{\mathbf{x}\in D} \cos f(\mathbf{x})$ and $\min_{\mathbf{x}\in D} \cos f(\mathbf{x})$ gives the result under the assumption that $\cos f$ does not change sign on D.

In the case of a polynomial leaf, $g(\mathbf{x})$, the bounds of $\min_{\mathbf{x}\in D} g(\mathbf{x})$ and $\max_{\mathbf{x}\in D} g(\mathbf{x})$ are directly obtained from its control points exploiting the convex hull property. Min/max bounds of interior nodes are computed following the same rules of interval arithmetic for simple arithmetic and Equations (14) to (16), in case of transendental functions. In the latter case, f is required to be monotone over D and $f(g(\mathbf{x}))$ is evaluated at $\min g(\mathbf{x})$ and $\max g(\mathbf{x})$.

3.3. Numerical Improvements Stage

Once a domain, which contains at most one root is isolated by a single solution test discussed in Section 2.2, a numerical improvement stage is commenced, and techniques, such as the ones presented in [8] or [9] could be clearly employed. Herein, we use simple Newton Raphson iterations, starting with an initial solution guess of x_0 at the mid point of the obtained domain. The expression tree structure of the equations also allows for an efficient computation of $\frac{\partial f_i}{\partial x_j}$, necessary for the Newton Raphson iterations, by using the derivative rules (such as the addition and multiplication rules, in Section 3.2).

If the iterations do not converge or go outside of the domain, we declare that there is no root in the domain. One can, in that case, continue the subdivision steps in the hope that a closer initial guess will be more successful. This, until some prescribed subdivision tolerance is met. The last case typically hints on non simple roots, which are prevented by passing the single solution test, or in some cases, on roots on the boundary of the domain.

3.4. Algorithm – Summary

Every function f_i of System (1) is represented by an expression tree. The solver parses the tree, starts with the simple bounding cones of polynomial functions at the leaves of the expression tree, and ends up at the root of the tree with a bounding normal cone of f_i. If in some node, the merging process fails to produce valid bounding cone (i.e. the cone spans the whole $\mathbb{R}^n$), the solver simply subdivides. Recall from [6] that the advantage of subdividing using expression trees stems from the fact that only the leaves in the direction of the subdivision are required to be subdivided.

Once the bounding cones of all the f_i functions are built, the complementary bounding cones are constructed (recall Section 2.2) and the single solution test

of [10] is executed. If this test succeeds, guaranteeing at most one isolated root within the domain, a multivariate Newton-Rapshon method is applied. Otherwise, the solver subdivides further, up to the permissible tolerance.

3.5. Analysis of the Bounding Cone's Tightness

In this section, we discuss the quality of the bound which was introduced in Section 3.1. The polyhedral bound, the bounding polytope, of the truncated normal cone is based on bounds of the cap(s), the $(n-1)$ dimensional ball(s) $\mathbb{B}^{n-1}$. For convenience, we shift the index to n, in this section.

Since the tightness of the polytope with respect to the truncated cone follows the quality of the bound of the orthoplex with respect to $\mathbb{B}^n$, we discuss the quality of this bound. Several criteria of the bound can be considered:

(1) Number of vertices (complexity) of the bounding polyhedron,
(2) distance of the farthest vertex from the center of the ball,
(3) the ratio between the volumes of the bound and the original ball.

As a natural alternative to an orthoplex, an n-dimensional cube comes up in mind. A comparison of these two bounds follows.
(1) As already mentioned in Section 3.1, an orthoplex consists of only $2n$ vertices in contrast to 2^n vertices of the cube.
(2) Any vertex of both bounding objects is at the distance of $r\sqrt{n}$ from the center of $\mathbb{B}^n$.
(3) The volume of a cube is $V(C^n) = (2r)^n$ and the volume of the inscribed ball [21], $\mathbb{B}^n$, is given by

$$V(\mathbb{B}^n) = \frac{(r\sqrt{\pi})^n}{\Gamma(\frac{n}{2}+1)}, \tag{17}$$

which can be rewritten using Stirling's approximation as

$$\frac{(r\sqrt{\pi})^n}{\Gamma\left(\frac{n}{2}+1\right)} \approx \left(\frac{2\pi e}{n}\right)^{\frac{n}{2}} \frac{r^n}{\sqrt{n\pi}}. \tag{18}$$

The direct computation of the volume [21] of the orthoplex gives

$$V(O^n) = \int_O d\mathbf{x} = (r\sqrt{n})^n \int_{\|\mathbf{x}\|_1 \leq 1} d\mathbf{x} = \frac{(2r\sqrt{n})^n}{n!}. \tag{19}$$

Table 1 displays the comparison of the criteria for various n.

Observe also the asymptotic behavior of $\frac{V(\mathbb{B}^n)}{V(O^n)}$ and $\frac{V(\mathbb{B}^n)}{V(C^n)}$, in Fig. 4. Using Stirling's approximation again, we get

$$\frac{V(\mathbb{B}^n)}{V(O^n)} = \frac{\frac{(r\sqrt{\pi})^n}{\Gamma(\frac{n}{2}+1)}}{\frac{(2r\sqrt{n})^n}{n!}} \approx \frac{(\frac{2\pi e}{n})^{\frac{n}{2}} \frac{r^n}{\sqrt{n\pi}}}{(\frac{2e}{\sqrt{n}})^n \frac{r^n}{\sqrt{2n\pi}}} = \sqrt{2}(\frac{\pi}{2e})^{\frac{n}{2}}, \tag{20}$$

whereas

$$\frac{V(\mathbb{B}^n)}{V(C^n)} \approx (\frac{e\pi}{2n})^{\frac{n}{2}} \frac{1}{\sqrt{n\pi}}, \tag{21}$$

Table 1: Cube vs. Orthoplex as a bound on unit ball $\mathbb{B}^n$ with respect to the dimension n. The numbers of vertices, volumes and relative volumes with respect to the ball $\mathbb{B}^n$ are shown.

n	$NumOfVert$		$Volume$		$Volumetric\ Ratio$	
	$Orth.$	$Cube$	$Orth.$	$Cube$	$\frac{Sphere}{Orth.}$	$\frac{Sphere}{Cube}$
2	4	4	4	4	$\frac{\pi}{4}$	$\frac{\pi}{4}$
4	8	16	$\frac{32}{3}$	16	$\frac{3\pi^2}{64}$	$\frac{\pi^2}{32}$
6	12	64	$\frac{96}{5}$	64	$\frac{5\pi^3}{576}$	$\frac{\pi^3}{384}$
8	16	256	$\frac{8192}{315}$	256	$\frac{105\pi^4}{65536}$	$\frac{\pi^4}{6144}$
10	20	1024	$\frac{16000}{567}$	1024	$\frac{189\pi^5}{640000}$	$\frac{\pi^5}{122880}$

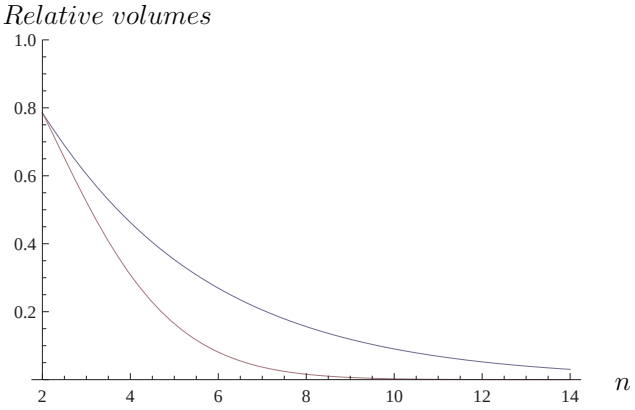


Figure 4: Comparison of the bounding tightness: Relative volumes $\frac{V(\mathbb{B}^n)}{V(C^n)}$ (red) and $\frac{V(\mathbb{B}^n)}{V(O^n)}$ (blue), as a function of dimension n, are depicted.

which converges to zero much faster.

As observed from the above analysis, the bounding orthoplex is not only more efficient to process but it also offers a satisfactory bound on $\mathbb{B}^n$, that is better than the bounding cube. Hence, the use of the orthoplex as a bounding volume results in a tight bound of the truncated cone.

4. Examples

In this section, the proposed solver is applied to curve-curve intersection, curve-surface intersection, ray-trap and geometric constraint problems, and is also compared to interval arithmetic.

In the first example, an Archimedean spiral is intersected with a parabola, resulting in four single roots. Grey rectangles at Fig. 5(b) show, when the single solution test guarantees at most one root. In the next example, see Fig. 6(a), circle $C_1(t) = [10\cos(t), 10\sin(t)]$, $t \in [0, 2\pi]$ is intersected with cycloid

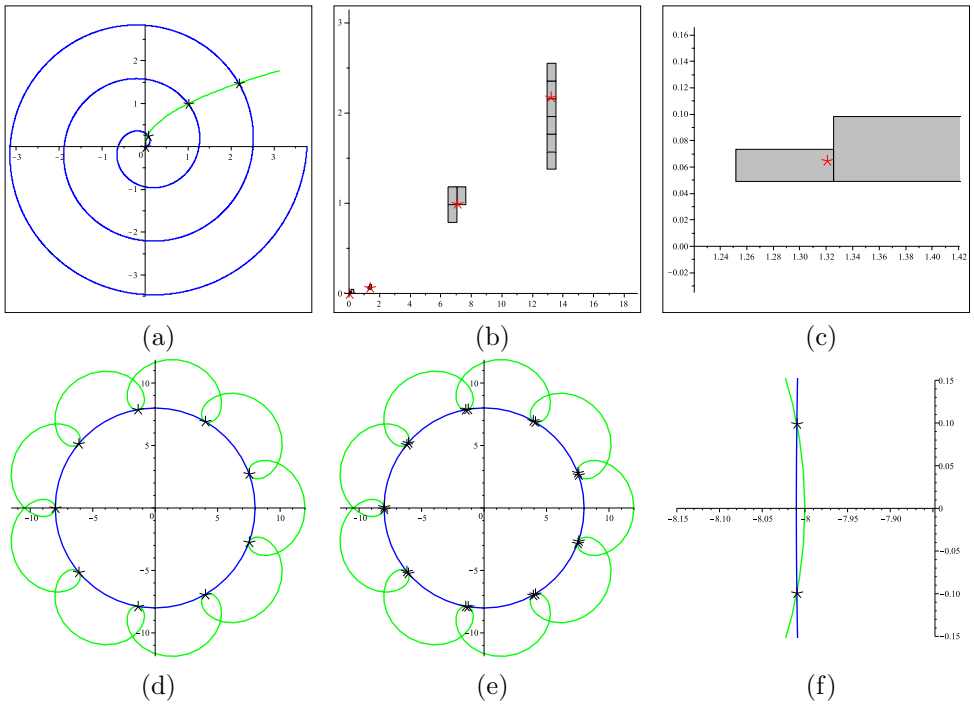


Figure 5: (a) Archimedean spiral $C_1(t) = [\frac{1}{5}t\cos(t), \frac{1}{5}t\sin(t)]$, $t \in [0, 6\pi]$ (blue) vs. a segment of parabola $C_2(s) = [s, \sqrt{s}]$, $s \in [0, \pi]$, (green) gives 4 intersection points. (b) Corresponding points (red asterisks) in the parametric st-space $[0, 6\pi] \times [0, \pi]$. The grey domains, rectangles with black polylines as boundaries, report when the subdivision was stopped with a guarantee of at most one root within the domain. (c) A zoom-in on the second root. In the case of the root at the origin $[0, 0]$, the subdivision tolerance of $\varepsilon_{sub} = 10^{-3}$ was reached. Once the subdivision is stopped, a Newton-Raphson scheme is applied to numerically reach the root. (d) A cycloid $C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ (green) and a circle (blue) of radius $r = 8$ possess a tangent contact along 9 points (black asterisks). (e) An almost-tangent configuration for $r = 8.01$ with 18 pairwise grouped intersection points and a zoom on one such a pair (f).

$C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ yielding the system

$$\begin{aligned} 10\cos(t) - 10\cos(s) - 2\cos(10s) &= 0, \\ 10\sin(t) - 10\sin(s) - 2\sin(10s) &= 0, \end{aligned} \tag{22}$$

over the Cartesian product of their parametric domains, $[0, 2\pi] \times [0, 2\pi]$. A modification of the circle's radius gives the tangent and almost-tangent configuration as shown at Fig. 5(e,f). Whereas the first case forces the solver to subdivide until the subdivision tolerance is reached, and the centers of the may-be-root domains are returned, in the latter case, the roots are isolated by the single solution test [10]. Observe the relatively small number of subdivisions in Ta-

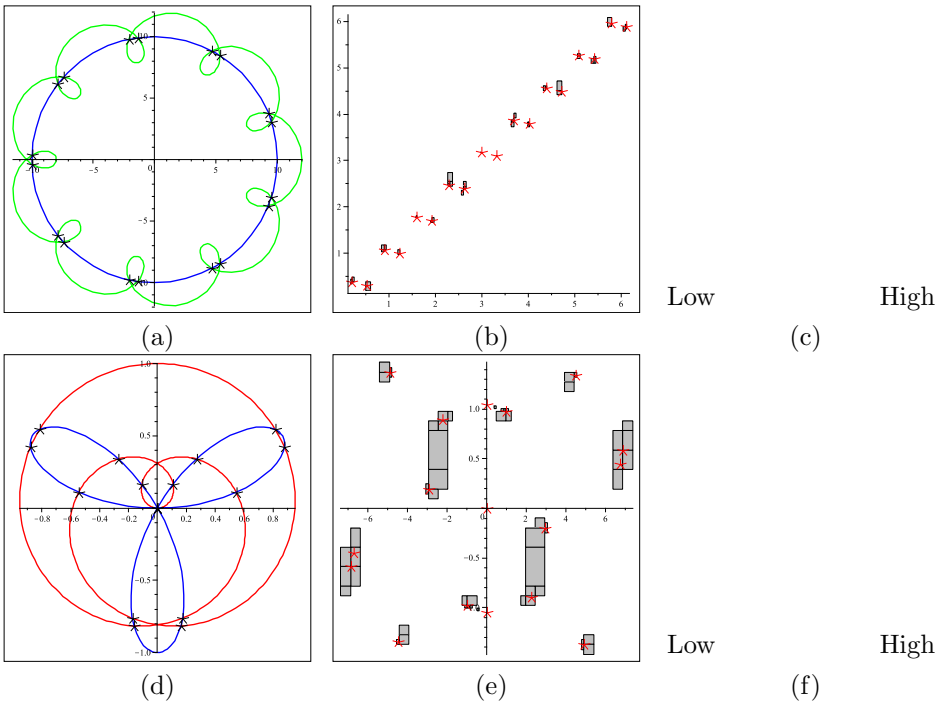


Figure 6: (a) A cycloid $C_2(s) = [10\cos(s) + 2\cos(10s), 10\sin(s) + 2\sin(10s)]$, $s \in [0, 2\pi]$ (green) is intersected with a circle (blue), giving 17 intersection points. (b) Corresponding points (red asterisks) in the solution (preimage) st-space $[0, 2\pi] \times [0, 2\pi]$ and sub-domains, that contain at most one root (grey). (c) Red isocurves indicate locations where the Jacobian of System (22) is zero. Color coding depicts the L^2 norm of the system, dark color corresponds to low values. Green dots are the roots. Bottom row, (d-f), an analogy for two cycloidal curves defined in (23).

ble 2, when compared to interval arithmetic, especially when higher accuracy is required.

A more complex example is shown in Fig. 6(d). Two cycloidal curves

$$\begin{array}{rcll} C_1(t) & = & [\sin(\frac{t}{5})\cos(t), \sin(\frac{t}{5})\sin(t)], & t \in [-\frac{5\pi}{2}, \frac{5\pi}{2}], \\ C_2(s) & = & [\sin(3s)\cos(s), \sin(3s)\sin(s)], & s \in [-\frac{\pi}{2}, \frac{\pi}{2}], \end{array} \tag{23}$$

are intersected, having fourteen single roots and a triple root at point $[0, 0]$. For a very fine subdivision tolerance $\varepsilon_{sub} = 10^{-6}$, interval arithmetic requires almost three times more subdivisions than the algorithm exploiting single solution test, see Table 2.

In the next example, two parametric curves

$$\begin{array}{rcl} C_1(t) & = & [t, \sin(\frac{1}{t})] \\ C_2(s) & = & [\sin(\frac{1}{s}), s]. \end{array} \tag{24}$$

are intersected, giving infinitely many solutions over domain $D = [-1, 1] \times$

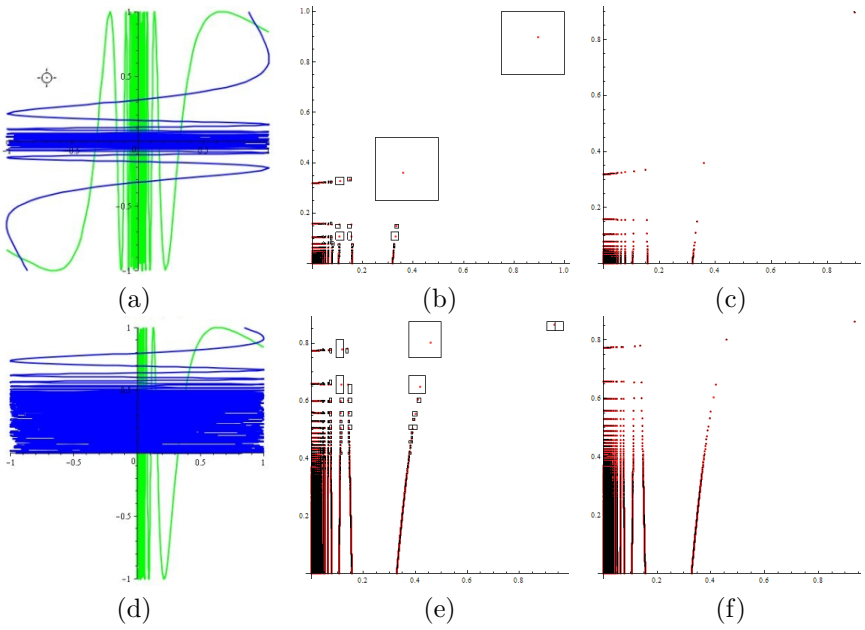
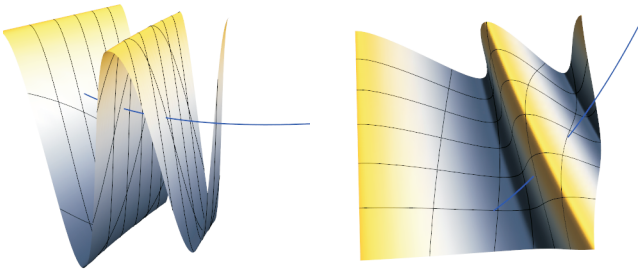


Figure 7: (a) Sinusoidal shaped curves $C_1(t) = [t, \sin(\frac{1}{t})]$, $t \in [-1, 1]$ (green) and $C_2(s) = [\sin(\frac{1}{s}), s]$, $s \in [-1, 1]$, (blue) with infinitely many intersection points in the vicinity of x- and y-axes. (b) The pre-images of intersection points (red) in the parameter st-subspace $[0.001, 1] \times [0.001, 1]$. The rectangles correspond to the stage of the algorithm when subdivision is terminated. Once the subdivision is stopped, a Newton-Raphson scheme is applied to numerically improve the root. (c) In contrast, interval arithmetic subdivides until subdivision tolerance of $\varepsilon_{sub} = 10^{-3}$ is reached. Bottom row, (d–f), an analogy for curves $C_1(t) = [t, \sin(\frac{1}{t})]$, $t \in [0, 1]$ (green) and $C_2(s) = [\sin(\frac{1}{s^5}), s]$, over domain $[0.001, 1] \times [0.1, 1]$.

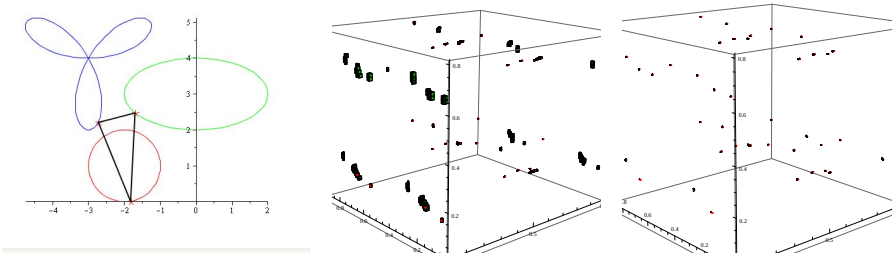
$[-1, 1]$, see Fig. 7. The intersection were sought for over sub-domain $D' = [0.001, 1] \times [0.001, 1]$ in order to test the ability of the single solution test to isolate roots. Black-bounded rectangles indicate when at most one root was detected. In this stage of the algorithm, the subdivision is stopped and the numerical improvement stage (in our implementation, a multivariate Newton-Raphson method) is invoked. Fig. 7(b) shows the root isolation in the pre-image space. In contrast, by its definition, interval arithmetic subdivides until the subdivision tolerance is reached. This fact may result in an incorrect number of roots if the subdivision tolerance is too coarse, or, in a huge number of subdivisions for fine tolerances. Similar behavior is also observed in the example at Fig. 7(d-f) for a slightly modified curve $C_2(s) = [\sin(\frac{1}{s^5}), s]$ and domain $D' = [0.001, 1] \times [0.1, 1]$.

As a next example, space curve $C(t) = [e^t, e^{-2t}, \frac{t}{2}]$, $t \in [0, 1]$ is intersected with surface $S(u, v) = [\ln(1+u), \ln(1+v), \cos(2\pi(uv+v))$, $[u, v] \in [0, 1] \times [0, 1]$, giving a system of dimension three, see Fig. 8.



(a) (b) (c)

Figure 8: (a) A curve-surface intersection with 2 solutions. (b) The same arrangement seen from the top. (c) A zoom on a root-containing segment of the solution, $uv \times t$, space. Colored voxels are reported when the solver stops subdivision, while the two thick black dots are the roots of the system.



(a) (b) (c)

Figure 9: (a) A set of three bouncing rays are trapped between three parametric planar curves. One solution is shown (black). (b) A zoom on the three curves' parameter *stu*-space with the sub-domains (green voxels) where at most a single solution is isolated. (c) In contrast, interval arithmetic subdivides until the given subdivision tolerance $\varepsilon_{sub} = 10^{-3}$ is reached.

The ray traps example is shown in Fig. 9. The ray-trap problem examines an arrangement of k objects for the existence of ray loops in which a ray is bounced off one object to the next in an infinite loop. Herein, the $k = 3$ objects are planar parametric curves parameterized by transcendental functions.

As a last example, a problem that corresponds to a 4×4 system is shown at Fig. 10. Vertices $V_1, \ldots, V_4$ of a square (of an unknown size) are constrained to lie in turn on four algebraic curves $\alpha_1 : (x-3)^2 + (y-0.4)^2 = 1$, $\alpha_2 : (x-3)^2 + (x-3)(y-3) + (y-3)^2 = 1$, $\alpha_3 : x^2 - x(y-3) + (y-3)^2 = 1$ and $\alpha_4 : x^2 + xy + y^2 = 1$. Consider V_i, $i = 1, \ldots, 4$ as $V_i = [c_1 + k\cos(\phi + \frac{(i-1)\pi}{2}), c_2 + k\sin(\phi + \frac{(i-1)\pi}{2})]$, where $C = [c_1, c_2]$ is the center of the square, k is the scaling factor, and ϕ is the angle between $V_1 - C$ and the positive x-axis. Substituting the coordinates of V_i into α_i gives the system. Solving for

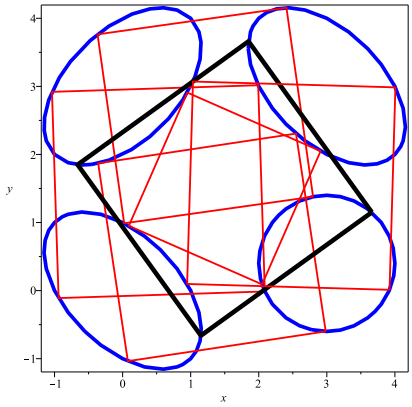


Figure 10: A solution of a 4×4 system: A square of an unknown edge's length is sought such that each its vertex lies on one of four particular implicit curves (blue). Six solutions were found (red). One solution is highlighted (bold black).

$[c_1, c_2, k, \phi]$ over domain $[0, 3] \times [0, 3] \times [0, 2.5] \times [-\pi/2, \pi/2]$ gives 6 solutions, see Fig. 10.

Table 2 gives a timing summary of all examples and shows a comparison of the proposed approach exploiting the single solution test with interval arithmetic. These timings are obtained from a 2.67 MHz IBM PC running Windows XP. Every example was executed 60 times and the timings were averaged. For low accuracies, the timings of both methods are fairly comparable since the benefit of the lower number of subdivisions is eliminated by the computational cost of the single solution test. On the contrary, in the case of higher subdivision tolerances, the single solution test shows a substantial reduction of subdivisions that is also reflected in the timings.

As a post-process, all roots were validated. An existence of false positive answers (invalid roots) was revealed in example at Fig. 9, see Table 2 where the number of invalid roots is displayed in italics. This phenomenon appeared in both methods and is exclusively related to the numerical improvement stage, see Section 3.3, and goes beyond the scope of this paper. We emphasize that after the subdivision stage, the single solution method returns a set of domains which contain "at most one root" whereas interval arithmetic answers "may be a root".

The solver has been implemented as a library of the Irit, solid modeling system [2], written in the C programming language and no third party libraries were used.

[2] www.cs.technion.ac.il/∼irit

Table 2: Statistics on examples computed by the presented algorithm and its comparison to interval arithmetic for two sets of tolerances: subdivision tolerance $\varepsilon_{sub} = 10^{-3}$ and numerical tolerance $\varepsilon_{num} = 10^{-6}$ and $\varepsilon_{sub} = 10^{-6}$, $\varepsilon_{num} = 10^{-8}$ (marked by $*$) over normalized parameter spaces. S indicates the number of domains returned after subdivision stage. Further, the number of total roots after the numerical stage, the number of invalid roots (italic), the number of subdivisions and timings (in seconds) of both algorithms are displayed.

Example	Single solution test				Interval arithmetic			
	S	roots	subd	time	S	roots	subd	time
Fig. 5(a)	15	4	57	0.005	11	4	191	0.004
Fig. 5(a)*	15	4	57	0.005	13	4	506	0.008
Fig. 5(d)	176	9	879	0.19	234	9	1449	0.17
Fig. 5(d)*	551	9	1994	0.63	7422	9	46260	5.3
Fig. 5(e)	187	18	901	0.16	259	18	1523	0.12
Fig. 5(e)*	200	18	1098	0.22	511	18	13518	0.57
Fig. 6(a)	52	18	640	0.056	72	18	990	0.035
Fig. 6(a)*	52	18	720	0.06	70	18	2334	0.07
Fig. 6(d)*	138	17	1305	0.26	97	17	3587	0.22
Fig. 7(a)	1238	1074	2419	0.25	1241	1074	2941	0.17
Fig. 7(d)	11927	5688	18709	2.33	11930	5688	19917	1.84
Fig. 8	9	2	42	0.02	8	2	237	0.008
Fig. 8*	9	2	42	0.02	8	2	506	0.014
Fig. 9	844	*6*+40	14934	44.6	912	*2*+40	18659	13.8
Fig. 10*	926	6	15817	27.9	996	6	62198	4.8

5. Conclusion and Future Work

In this work, we have presented a solver that robustly solves well-constrained $n \times n$ transcendental systems. Exploiting the expression trees to construct bounding normal cones of n transcendental constraints, the subdivision based solver detects all sub-domains, where at most one root can exist. The root is then numerically improved by a multivariate Newton-Raphson scheme.

The presented solver guarantees to isolate and return *all* single roots of the system within a given domain. Other roots are only isolated. This, in contrast of commercial software such as Maple[3], Mathematica[4], or Matlab[5], which – to our best knowledge – can locally isolate and provide only one root or a few of them.

As a future work, an improved algorithm is intended, which better handles

[3]http://www.maplesoft.com/
[4]http://www.wolfram.com/products/mathematica
[5]http://www.mathworks.com/

multiple roots. As of now, the solver only subdivides to such points, up to the permissible tolerance. In addition, the numerical stage of the algorithm deserves some further research. For once, under the current scheme, there is no guarantee that a multivariate Newton-Raphson scheme will converge to the root. Also, if the system is underconstrained and (at least) one-dimensional solution space is expected like in [3], a special treatment of the system is more favorable.

Despite the use of expression trees, which are highly efficient during the subdivision stage, the growth in the number of subdivisions is required to be minimal with respect to the dimension n. Hence, the handling of higher-dimensional systems is within the scope of our interest.

6. Acknowledgments

This research was partly supported by the Israel Science Foundation (grant No. 346/07), in part by the Israeli Ministry of Science Grant No. 3-8273, and in part by the New York metropolitan research fund, Technion.

[1] A. V. Aho, J. D. Ullman and J. E. Hopcroft. *Data structures and algorithms*. Addison Wesley, 1983.

[2] G. Barequet and G. Elber. Optimal bounding cones of vectors in three and higher dimensions. *Information Processing Letters*, 93:83–89, 2005.

[3] M. Bartoň, I. Hanniel and G. Elber. Topologically guaranteed univariate solutions of underconstrained polynomial systems via no–loop and single–component tests. *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, 207–212, Haifa, 2010.

[4] M. Aizenshtein, M. Bartoň and G. Elber. Global Solutions of Well-constrained Transcendental Systems using Expression Trees and a Single Solution Test *Advances in Geometric Modeling and Processing*, Lecture Notes in Computer Science, vol. 6130/2010, 1-18, 2010.

[5] D. A. Cox, J. B. Little, and D. O'Shea. *Using algebraic geometry*. Springer, 2005.

[6] G. Elber and T. Grandine. Efficient solution to systems of multivariate polynomials using expression trees. In *Tenth SIAM Conference on Geometric Design and Computing*, 2007.

[7] J. Gaukel. Efficient solving of polynomial and nonpolynomial systems using subdivision (in German), PhD thesis, TU Darmstadt, 2003.

[8] T. N. Graspa and M. N. Vrahatis. Dimension reducing methods for systems of nonlinear equations and unconstrained optimization: A review. *Recent Adv. Mech. Related Fields*, 215–225, 2003.

[9] C. Grosan and A. Abraham. A new approach for solving nonlinear equations systems. In *IEEE Transactions on systems, man and cybernetics: Systems and humans*, (38), No. 3, 2008.

[10] I. Hanniel and G. Elber. Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. *Computer Aided Design*, (39):369–378, 2007.

[11] J. M. McNamee. Bibliographies on roots of polynomials. In *J. Comp. Appl. Math* (47): 391–394, (78):1–1, (110):305–306; (142):433–434, 1993–2002.

[12] B. Mourrain and J.-P. Pavone. Subdivision methods for solving polynomial equations. *J. of Symbolic Computation*, 44, 3, 292–306, 2009.

[13] A. Mantzaflaris, B. Mourrain and E. Tsigaridas. Continued fraction expansion of real roots of polynomial systems. In Proceedings of *Conference on Symbolic Numeric computation* (SNC '09), 85-94, 2009.

[14] A. Neumaier. Introduction to Numerical Analysis. Cambridge Univ. Press, Cambridge 2001.

[15] M. Reuter, T. S. Mikkelsen, E. C. Sherbrooke, T. Maekawa, and N. M. Patrikalakis. Solving nonlinear polynomial systems in the barycentric Bernstein basis. *Visual Computer*, (24):187–200, 2008.

[16] E. C. Sherbrooke and N. M. Patrikalakis. Computation of solution of nonlinear polynomial systems. *Computer Aided Geometric Design*, 5(10):379–405, 1993.

[17] A. J. Sommese and C. W. Wampler. *The numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.

[18] J. Stewart. *Multivariate Calculus*. 2002.

[19] H. Wang, J. Kearney, and K. Atkinson. Robust and efficient computation of the closest point on a spline curve. In: Lyche, T., et al. (Eds.), *Curve and Surface Design*. Saint Malo 2002. Nashboro Press, Brentwood, 397–405, 2002.

[20] J. Zhou, E. C. Sherbrooke and N. M. Patrikalakis. Computation of stationary points of distance functions *Engineering with Computers*. Vol. 9, No. 4, pp. 231–246, 1993.

[21] G. M. Ziegler. *Lectures on Polytopes (Graduate Texts in Mathematics)*. Springer, 1998.