Numerical Solving of Constraints of Multivariate Polynomial Strict Inequalities

Teimuraz Kutsia, Josef Schicho

Research Institute of Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria

October 31, 1999

Abstract

We consider the problem of safe numerical solving of constraints in a form of disjunctions and/or conjunctions of multivariate polynomial strict inequalities with real coefficients. We use a modified bisection method and present an algorithm that computes a solution of the problem.

1 Introduction

Solving of polynomial equations and inequalities is a known and important problem in scientific computing. Numerous methods have been developed to solve this problem. A symbolic method for finding a real solution of a system of integral polynomial (non-strict) inequalities is described by Grigor'ev and Vorobjov [3]. McCallum [4] uses another symbolic method - cylindrical algebraic decomposition to solve the problem of determining the consistency over the real numbers of a system of integral polynomial strict inequalities.

However, pure symbolic methods are expensive, e.g. because of the costly arithmetic of algebraic numbers. Fast numerical methods (and other approaches, like e.g. subdefinite calculations [1]) can be considered as an alternative way for solving the problem. But, because of non-safe approximation, not all algorithms guarantee correctness of the results which would be essential in some applications. A good compromise is to use safe numerical methods like, for example, the exclusion method for computing real roots of polynomials (Dedieu and Yakobsohn [2]).

In this paper we present a solution of the problem for polynomial inequalities in this spirit, using a safe numerical method. We consider constraints in a form of disjunctions and/or conjunctions of multivariate polynomial strict inequalities with real coefficients and present an algorithm that computes a solution of a constraint in a given parallelepiped. The algorithm is based on the bisection method. Moreover, we give a precise interpretation of the approximation error using the concept of ε-solution set. The solution computed by the algorithm contains ε-solution set of the constraint on a given parallelepiped and approximates the set of exact solutions from inside.

The paper is organized as follows: Section 2 contains some preliminaries and a statement of the problem. In section 3 the solving method is described. Section 4 contains a detailed description of the algorithm.

2 Definitions and Problem Statement

We consider constraints defined as follows:

```
constraint :: polynomial relation 0
              constraint ∨ constraint
              constraint ∧ constraint

relation :: >
            <
```

where 'polynomial' means a polynomial with real coefficients.

A constraint is called atomic with a polynomial $p(x_1, \dots, x_n)$ if it has a form $p(x_1, \dots, x_n) > 0$ or $p(x_1, \dots, x_n) < 0$.

Below by an n-dimensional parallelepiped we mean a subset $\mathbb{R}^n$

$$[a_1, b_1]\mathsf{x}[a_2, b_2]\mathsf{x} \cdots \mathsf{x}[a_n, b_n]$$

of n-dimensional real space.

The problem to be solved can be stated as follows:

General Problem:

Given:

- *n-variate constraint*
- *n-dimensional parallelepiped*
- *precision (a real number)*

Find:

A solution of the constraint in the parallelepiped by the given precision.

Statement of the problem is very general and needs to be specified. In order to specify what *a solution of a constraint in a parallelepiped by a given precision* means we need the following definitions:

Definition 1 *Given a constraint C and a precision ε an ε-solution set for C, denoted by $Sol(\varepsilon, C)$ is defined as follows:*

- *If C has a form $p(x_1, \dots, x_n) > 0$, where $p(x_1, \dots, x_n)$ is an n-variate polynomial, then*

  $$Sol(\varepsilon,C) = \{(r_1, \dots r_n) \,|\, (r_1, \dots r_n) \in \mathbb{R}^n,\ p(r_1, \dots, r_n) > \varepsilon\}$$

- *If C has a form $p(x_1, \dots, x_n) < 0$, where $p(x_1, \dots, x_n)$ is an n-variate polynomial, then*

  $$Sol(\varepsilon,C) = \{(r_1, \dots r_n) \,|\, (r_1, \dots r_n) \in \mathbb{R}^n,\ p(r_1, \dots, r_n) < -\varepsilon\}$$

- *If C has a form $C_1 \wedge C_2$, where C_1 and C_2 are n-variate constraints, then*

  $$Sol(\varepsilon, C) = \{(r_1, \dots r_n) \,|\ (r_1, \dots r_n) \in \mathbb{R}^n,\ (r_1, \dots r_n) \in Sol(\varepsilon, C_1) \cap Sol(\varepsilon, C_2)\}$$

- *If C has a form $C_1 \vee C_2$, where C_1 and C_2 are n-variate constraints, then*

$$Sol(\varepsilon, C) = \\ \{(r_1, \dots r_n) \mid (r_1, \dots r_n) \in \mathbb{R}^n,\ (r_1, \dots r_n) \in Sol(\varepsilon, C_1) \cup Sol(\varepsilon, C_2)\}$$

Definition 2 *By an exact solution of a constraint C we mean $Sol(0, C)$.*

Now the general problem stated above can be specified as follows:

Specified Problem:

Given:

- *n-variate constraint C*
- *n-dimensional parallelepiped D*
- *precision ε*

Find a subset S of D such that

$$Sol(\varepsilon, C) \cap D \subseteq S \subseteq Sol(0, C) \cap D$$

3 Solving Method

3.1 Solution Idea - Bisection

Below we use the letters C, D, I, ε to denote respectively a constraint, a parallelepiped, a sequence of parallelepipeds and a precision.

The idea of bisection method is the following: given C, I, ε compute a solution sequence of parallelepipeds S as follows:
S is empty from the beginning.
While I is not empty do:
Take the first parallelepiped D from I;

- if C is **true on** D then add D to S;
- otherwise, if C is **false on** D, then let I be $I - \{D\}$;
- otherwise, **bisect** D, add obtained parallelepipeds to the beginning of $I - \{D\}$ and denote the obtained sequence by I

Return S.

We will discuss the role of ε in this method a bit later. To carry out the idea of bisection one needs to know

- how to compute the truth value of a constraint and
- how to bisect a parallelepiped

These problems are subject of discussion in the subsections below.

3.2 Computing the Truth Value of a Constraint on a Parallelepiped

We consider three truth values - *true*, *false* and *undefined* and order them as $false < undefined < true$.

Idea of Computing of the Truth Value of a Constraint on a Parallelepiped:

Let C be an n-variate constraint and $D = [a_1, b_1]\mathsf{x}\cdots\mathsf{x}[a_n, b_n]$.

1. If C is an atomic constraint with a polynomial $p(x_1, \dots, x_n)$ then:
    - if $p(x_1, \dots, x_n)$ **has no roots in** D and the point $(a_1, \dots, a_n)$ satisfies C, then the truth value of C on D is *true*
    - if $p(x_1, \dots, x_n)$ **has no roots in** D and the point $(a_1, \dots, a_n)$ does not satisfy C, then the truth value of C on D is *false*
    - if $p(x_1, \dots, x_n)$ **has roots in** D then truth value of C on D is *undefined*
2. if C has a form $C_1 \wedge C_2$, then the truth value of C on D is a minimum of truth values of C_1 and C_2 on D
3. if C has a form $C_1 \vee C_2$, then the truth value of C on D is a maximum of truth values of C_1 and C_2 on D

Thus, the problem of computing the truth value of a constraint on a parallelepiped is reduced to a problem of existence of roots of a polynomial on the parallelepiped. We have the following sufficient condition:

Sufficient Condition for Non-existence of Roots of a Polynomial in a Parallelepiped: Given n-variate polynomial $p(x_1,\dots,x_n)$ and n-dimensional parallelepiped $D = [a_1,b_1]\text{x}\cdots\text{x}[a_n,b_n]$, if

$$|p(a_1,\dots,a_n)| > \sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)$$

where $U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)$ is an upper bound of i-th partial derivative of the polynomial $p(x_1,\dots,x_n)$ on D, then $p(x_1,\dots,x_n)$ has no roots in D.

Since this is only sufficient condition for non-existence of roots of a polynomial we are not always able to solve the problem of computing truth values. Instead, we introduce a weaker concept of ε-truth value of a constraint on a parallelepiped that can be computed using the sufficient condition: For C, $D = [a_1,b_1]\text{x}\cdots\text{x}[a_n,b_n]$ and ε

1. If C is an atomic constraint with a polynomial $p(x_1,\dots,x_n)$ then
   - if $p(x_1,\dots,x_n)$ satisfies the sufficient condition and the point $(a_1,\dots,a_n)$ satisfies C, then the ε-truth value of C on D is $true$
   - if $p(x_1,\dots,x_n)$ satisfies the sufficient condition and the point $(a_1,\dots,a_n)$ does not satisfy C, then the ε-truth value of C on D is $false$
   - if $p(x_1,\dots,x_n)$ does not satisfy the sufficient condition and

     $$\sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right) \le \varepsilon,$$

     then the ε-truth value of C on D is $false$
   - otherwise ε-truth value of C on D is $undefined$
2. if C has a form $C_1 \wedge C_2$, then ε-truth value of C on D is a minimum of ε-truth values of C_1 and C_2 on D
3. if C has a form $C_1 \vee C_2$, then ε-truth value of C on D is a maximum of ε-truth values of C_1 and C_2 on D

Thus, computing an ε-truth value of a constraint we overcome the first problem related with the bisection method. The other problem - bisection of a parallelepiped is considered in the next subsection.

3.3 Bisection of a Parallelepiped

The idea of bisecting of a parallelepiped is quite simple: Halve all faces of (n-dimensional) parallelepiped and obtain 2^n new parallelepipeds of less size hoping that ε-truth value of a constraint (which was *undefined* on the initial parallelepiped) will become *true* or *false* on new parallelepipeds. But it has disadvantages, namely:

- It is not guaranteed that the ε-truth value of the constraint is *true* or *false* on the first new parallelepiped and thus, one needs to continue the same procedure (computing-bisecting) until the ε-truth value of the constraint is not either *true* or *false* on the first parallelepiped;
- All faces will be bisected every time while in some cases it is enough to bisect only some of them to obtain desired result.

We refine the idea of bisection to overcome these difficulties. The refinement is based on the following ideas:

- Instead of halving a face divide it in such two parts that truth value of the constraint on the first new parallelepiped is *true* or *false*.
- Try to keep unbisected as much faces as possible.

To implement these ideas we need to extract additional information from the sufficient condition of non-existence of roots of a polynomial. Note that for a given precision ε an ε-truth value of an atomic constraint C with a polynomial $p(x_1, \ldots, x_n)$ is *undefined* on a parallelepiped $D = [a_1, b_1] \mathsf{x} \cdots \mathsf{x} [a_n, b_n]$ iff

$$|p(a_1, \ldots, a_n)| \leq \sum_{i=1}^{n} |b_i - a_i| U \left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i} \right)$$

and

$$\sum_{i=1}^{n} |b_i - a_i| U \left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i} \right) > \varepsilon.$$

Refinement:

1. Suppose $|p(a_1, \ldots, a_n)| \leq \varepsilon$.

Divide each face of the parallelepiped on the value

$$\frac{\sum_{i=1}^{n}\left\{|b_i - a_i| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)\right\}}{\varepsilon}$$

called division coefficient. Denote by D' the first parallelepiped obtained after the division. Let $l_1,\dots,l_n$ be lengths of faces of D' and $U'\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)$ be the upper bound of i-th partial derivative of the polynomial $p(x_1,\dots,x_n)$ on D'. Then for each i, $1 \le i \le n$

$$U'\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right) \le U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right),$$

$$l_i = \frac{|b_i - a_i|\varepsilon}{\sum_{j=1}^{n} |b_j - a_j| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_j}\right)}$$

and the sum

$$\sum_{i=1}^{n} l_i U'\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right) =$$

$$\frac{\varepsilon}{\sum_{j=1}^{n} |b_j - a_j| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_j}\right)} \sum_{i=1}^{n} |b_i - a_i| U'\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right) \le \varepsilon$$

Therefore,

- if $p(a_1,\dots,a_n) \le \sum_{i=1}^{n} l_i U'\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)$ then ε-truth value of C on D' is $false$
- otherwise,
  - if $(a_1,\dots,a_n)$ satisfies C, then ε-truth value of C on D' is $true$,
  - otherwise $false$.

2. Suppose $|p(a_1, \dots, a_n)| > \varepsilon$. Then represent the sum

$$\sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1, \dots, x_n)}{\partial x_i}\right)$$

as a sum of two summands $L + G$ where

$$L = \sum_{i=1}^{k} |b_{m_i} - a_{m_i}| U\left(\frac{\partial p(x_1, \dots, x_n)}{\partial x_{m_i}}\right) < |p(a_1, \dots, a_n)| - 1$$

and

$$G = \sum_{i=k+1}^{n} |b_{m_i} - a_{m_i}| U\left(\frac{\partial p(x_1, \dots, x_n)}{\partial x_{m_i}}\right),$$

$k \geq 0$, $\{m_1, \dots, m_n\} = \{1, \dots, n\}$. Then the sufficient condition can be written as

$$|p(a_1, \dots, a_n)| \leq L + G.$$

Let $M = |p(a_1, \dots, a_n)| - L$. Then $M \leq G$. Define division coefficient as

$$DivCoef := \frac{G + \varepsilon}{M}$$

Divide all faces $[a_{m_{k+1}}, b_{m_{k+1}}], \dots [a_{m_n}, b_{m_n}]$ on $DivCoef$. Let D' be the first parallelepiped obtained after the division and $U'\left(\frac{\partial p(x_1, \dots, x_n)}{\partial x_i}\right)$ be the upper bound of i-th partial derivative of the polynomial $p(x_1, \dots, x_n)$ on D'. Then we obtain

$$\begin{aligned}
&L + \frac{1}{DivCoef} \sum_{i=k+1}^{n} |b_{m_i} - a_{m_i}| U'\left(\frac{\partial p(x_1, \dots, x_n)}{\partial x_{m_i}}\right) \leq \\
&L + \frac{1}{DivCoef} G < \\
&L + M = \\
&|p(a_1, \dots, a_n)|
\end{aligned}$$

It means that the sufficient condition is satisfied on D'. Then, if $(a_1, \dots, a_n)$ satisfies C, then the ε-truth value of C on D' is $true$, otherwise $false$.

Thus, the refinement guarantees that the ε-truth value of the constraint is either $true$ or $false$ on the first parallelepiped obtained after bisection. Also, it allows to keep unbisected as much faces of the parallelepiped as possible.

4 Algorithm

In this section we describe an algorithm to solve the given problem. First we compute an ε-truth value of a constraint C on a parallelepiped D by a precision ε, a division coefficient and a set of faces of D to bisect by the coefficient.

```
Function $Value(C, D, \varepsilon)$
▷
  Input $n$-variate constraint $C$, $n$-dimensional parallelepiped
        $D = [a_1, b_1] \text{x} \cdots \text{x} [a_n, b_n]$, real number $\varepsilon$ - precision
  Output Triple $(T, F, DivCoef)$, where $T$ is a $\varepsilon$-truth value
         of $C$ on $D$, $F$ is a set of faces of $D$ that should be
         bisected, $DivCoef$ is a division coefficient
◁
  if $C$ is an atom with polynomial $p(x_1, \ldots, x_n)$ then
   if $|p(a_1, \ldots, a_n)| > \sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i}\right)$ then
     $DivCoef := 1$;
     $F := \emptyset$;
     $T :=$ True
    else if $\sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i}\right) \leq \varepsilon$ then
      $DivCoef := 1$;
      $F := \emptyset$;
      $T =$ False
    else if $|p(a_1, \ldots, a_n)| \leq \varepsilon$ then
      $DivCoef := \frac{\sum_{i=1}^{n} \left\{|b_i - a_i| U\left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i}\right)\right\}}{\varepsilon}$;
      $F$:=all faces of $D$;
      $T$:=Undefined;
    else if $|p(a_1, \ldots, a_n)| = \sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1, \ldots, x_n)}{\partial x_i}\right)$ then
      $DivCoef := 2$;
      $F$:=largest face of $D$;
      $T$:=Undefined;
      else
```

```
            Represent the sum $\sum_{i=1}^{n} |b_i - a_i| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_i}\right)$
                as a sum of two summands $L+G$ where
                $L = \sum_{i=1}^{k} |b_{m_i} - a_{m_i}| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_{m_i}}\right) <$
                    $< |p(a_1,\dots,a_n)| - 1,$
                $G = \sum_{i=k+1}^{n} |b_{m_i} - a_{m_i}| U\left(\frac{\partial p(x_1,\dots,x_n)}{\partial x_{m_i}}\right),$
                $k \geq 0$, $\{m_1,\dots,m_n\} = \{1,\dots,n\}$.
            $DivCoef := \frac{G+\varepsilon}{|p(a_1,\dots,a_n)| - L}$;
        $F$:=all faces of $D$ from $G$;
        $T$:=**Undefined**;
else if $C$ has a form $C_1 \wedge C_2$
    $(T_1, F_1, DivCoef_1) := Value(C_1, D, \varepsilon)$;
    if $T_1$ =**False** then
        $T$ :=**False**;
        $F = F_1$;
        $DivCoef := DivCoef_1$;
    else
        $(T_2, F_2, DivCoef_2) := Value(C_2, D, \varepsilon)$;
        $T := min(T_1, T_2)$;
    if $T$ =**Undefined**;
        $F := F_1 \cup F_2$;
        $DivCoef = max(DivCoef_1, DivCoef_2)$;
    else
        $DivCoef := 1$;
        $F = \emptyset$;
else if $C$ has a form $C_1 \vee C_2$
    $(T_1, F_1, DivCoef_1) := Value(C_1, D, \varepsilon)$;
    if  $T_1$ =**True** then
        $T$ :=**True**;
        $F := F_1$;
        $DivCoef := DivCoef_1$;
    else
        $(T_2, F_2, DivCoef_2) := Value(C_2, D, \varepsilon)$;
        $T := max(T_1, T_2)$;
        if $T$ =**Undefined**;
          $F := F_1 \cup F_2$;
          $DivCoef = max(DivCoef_1, DivCoef_2)$;
        else
            $DivCoef := 1$;
            $F := \emptyset$;
  Return $(T, F, DivCoef)$
```

This function playes a key role in the main algorithm below:

```
Algorithm $Solver(C,\ D,\ \varepsilon)$
$\triangleright$
  Input $n$-variate constraint $C$, $n$-dimensional parallelepiped
        $D = [a_1, b_1] \text{x} \cdots \text{x} [a_n, b_n]$, real number $\varepsilon$ - precision
  Output Subset $S$ of $D$ such that
        $Sol(\varepsilon, C) \cap D \subseteq S \subseteq Sol(0, C) \cap D$
$\triangleleft$
  Initialize $I :=< D >$;
  Initialize $S := \emptyset$;
  while $I$ is not empty do
     take the first parallelepiped $D$ from $I$;
     $(T, F, Divcoef) := Value(C, D, \varepsilon)$;
     if $T$ =**True**
       $S := S \cup D$;
       $I := I - \{D\}$;
     else if $T$ =**False**
         $I := I - \{D\}$;
     else
         Bisect each face $[a_i, b_i]$ of $F$ in two parts:
           $[a_i, a_i + \frac{|b_i - a_i|}{DivCoef}]$ and $[a_i + \frac{|b_i - a_i|}{DivCoef}, b_i]$;
         $I$ := sequence obtained by adding all new
           parallelepipeds in head of $I$;
  Return $S$;
```

5 Implementation

The algorithm is implemented on C. Polynomials, constraints, parallelepipeds and sequences of parallelepipeds are represented as structures. GNU multiple precision arithmetic library is used to represent and operate on floating point real numbers. Since this representation is quite costly, running some examples faced the following problem:

```
Cannot allocate gmp: resource temporary unavailable.
Abort
```

Since the algorithm is exponential, number of bisected parallelepipeds increases fast. Keeping them with the ends represented in multiple precision floats needs big amount of memory. Trying to reduce the number of parallelepipedss kept we put the following condition on a minimal border of a parallelepiped: if the length of a minimal border is less than ε^2 (provided that $\varepsilon < 1$) then discard the parallelepiped. Therefore, there are some parallelepipeds with a 'rather small' minimal border which should be in a solution set but because of the condition they have been discarded from there.

Example 1 Consider the following constraint

$$x^4 - 2xy + y^2 < 0 \ \wedge \ 2431xy - 3301y - 2431x + 2685 > 0.$$

To solve it in the parallelepiped $[-1,\ 1]\ [-1,\ 1]$ with precision $\varepsilon = 0.1$ the algorithm needs 2388 bisection steps and produces the solution set consisting of 1107 parallelepipeds. Part of this solution is given below:

```
Parallelepiped 1
[ -0.94e0 , -0.9101018560230351066e0 ]
[ -0.94e0 , -0.9297676401230769231e0 ]
Parallelepiped 2
[ -0.9101018560230351066e0 , -0.8863579028230337432e0 ]
[ -0.9398957245546097634e0 , -0.9297676401230769231e0 ]
Parallelepiped 3
[ -0.8863579028230337432e0 , -0.8585468715726248084e0 ]
[ -0.9397698251100384274e0 , -0.9396223604752369086e0 ]
Parallelepiped 4
[ -0.94e0 , -0.9297676401230769231e0 ]
[ -0.9297676401230769231e0 , -0.9038218619887260361e0 ]
```

```
Parallelepiped 5
[ -0.9398624256446021927e0 , -0.9397261992981452871e0 ]
[ -0.9038218619887260361e0 , -0.8781303110379416136e0 ]
Parallelepiped 6
[ -0.9297676401230769231e0 , -0.8958268036233072827e0 ]
[ -0.9297676401230769231e0 , -0.9181117231810328071e0 ]
Parallelepiped 7
[ -0.8958268036233072827e0 , -0.8693170774107198618e0 ]
[ -0.9296322198718549521e0 , -0.9181117231810328071e0 ]
Parallelepiped 8
[ -0.8693170774107198618e0 , -0.8388258042364510827e0 ]
[ -0.9294711264573494585e0 , -0.9181117231810328071e0 ]
Parallelepiped 9
[ -0.8388258042364510827e0 , -0.8050760130551304362e0 ]
[ -0.9292858381118994622e0 , -0.9181117231810328071e0 ]
Parallelepiped 10
[ -0.8050760130551304362e0 , -0.769254055260403876e0 ]
[ -0.929080748513589511e0 , -0.9181117231810328071e0 ]

...

Parallelepiped 1097
[ 0.7353320614290356468e0 , 0.7354662569350640448e0 ]
[ 0.5917564248543491003e0 , 0.5919634180150821817e0 ]
Parallelepiped 1098
[ 0.7354662569350640448e0 , 0.7355678792004077423e0 ]
[ 0.5919634180150821817e0 , 0.5921201677791159822e0 ]
Parallelepiped 1099
[ 0.7355678792004077423e0 , 0.7356448412773256948e0 ]
[ 0.5921201677791159822e0 , 0.592238879828642129e0 ]
Parallelepiped 1100
[ 0.7356448412773256948e0 , 0.7357031310967890102e0 ]
[ 0.592238879828642129e0 , 0.5923287903951458915e0 ]
Parallelepiped 1101
[ 0.7357031310967890102e0 , 0.7357472810109998242e0 ]
[ 0.5923287903951458915e0 , 0.5923968905168195404e0 ]
Parallelepiped 1102
[ 0.7357472810109998242e0 , 0.7357807223063301916e0 ]
[ 0.5923968905168195404e0 , 0.5924484728657692612e0 ]
Parallelepiped 1103
[ 0.7357807223063301916e0 , 0.7358060530822229943e0 ]
[ 0.5924484728657692612e0 , 0.5924875449444181766e0 ]
Parallelepiped 1104
[ 0.7358060530822229943e0 , 0.7358252407908614865e0 ]
[ 0.5924875449444181766e0 , 0.5925171414978039992e0 ]
Parallelepiped 1105
[ 0.7358252407908614865e0 , 0.7358397754456859496e0 ]
[ 0.5925171414978039992e0 , 0.592539560833825902e0 ]
Parallelepiped 1106
[ 0.7358397754456859496e0 , 0.7358507855547343995e0 ]
[ 0.592539560833825902e0 , 0.5925565436474789335e0 ]
Parallelepiped 1107
[ 0.7358507855547343995e0 , 0.7358591258711289268e0 ]
[ 0.5925565436474789335e0 , 0.5925694083737511516e0 ]
```

References

[1] Babichev A.B., Kadyrova O.B., Kashevarova T.P., Leshchenko A.S. and Semenov A.L. (1993). UniCalc, a Novel Approach to Solving Systems of Algebraic Equations. Proceedings of the International Conference on Numerical Analysis with Automatic Result Verifications, Lafayette, Louisiana, USA, 1993. *Interval Computations, 2,* 29-47.

[2] Dedieu, J.P. and Yakobsohn, J.C. (1993). Computing the Real Roots of a Polynomial by the Exclusion Algorithm. *Numerical Algorithms 4,* 1-24.

[3] Grigor'ev, D. Yu. and Vorobjov N. N. (Jr) (1988). Solving Systems of Polynomial Inequalities in Subexponential Time. *J. Symbolic Computation 5,* 37-64.

[4] McCallum, S. (1993). Solving Polynomial Strict Inequalities Using Cylindrical Algebraic Decomposition. *The Computer Journal, Vol.36, No.5*, 432-438.