Solving and Visualizing Nonlinear Set Inversion Problems*

Elif Garajová
Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
elif.garajova@matfyz.cz

Martin Mečiar
Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
martinmeciar@centrum.sk

Abstract

The notion of constraint satisfaction allows us to formalize various problems involving a set of variables with given properties. This paper presents a solver for nonlinear set inversion problems, a specific class of continuous constraint satisfaction problems, in which the properties are described by a system of nonlinear inequalities over interval domains. The solver is based on the branch-and-bound algorithm SIVIA combined with interval contractors and is accompanied by a visualization tool, which can provide a two-dimensional graphical representation of the solution sets. It also implements several methods for obtaining a finer approximation of a solution set using inner and outer approximations by interval boxes.

Keywords: nonlinear constraints, interval analysis, visualization
AMS subject classifications: 65-04, 65G40

1 Introduction

Many problems from various branches of natural and technical sciences can be modeled using the notion of constraint satisfaction. Formally, we can define a *constraint satisfaction problem* (CSP) as the triplet (X, D, C), where

- $X = \{x_1, \ldots, x_n\}$ is a set of variables,
- $D = \{D_1, \ldots, D_n\}$ is a set of the respective domains,
- $C = \{c_1, \ldots, c_m\}$ is a set of constraints to be satisfied (relations over X).

*Submitted: November 20, 2015; Accepted: January 25, 2016.

In this paper, we are interested in a special case of CSPs: we only allow bounded closed real intervals as the domains and both linear and nonlinear inequalities as the constraints. This case, the so-called *set inversion problem*, can be approached with interval methods by finding an inner and outer approximation of the solution set using a set inversion algorithm. The resulting approximations are then described by sets of interval boxes.

However, sometimes it might be difficult to extract useful information about the solution set only from lists of boxes generated by an interval solver. It can be convenient to be able to obtain a graphical representation of the data, which is often easier to analyze. Visualization can serve as a helpful tool in determining the shape and structure of the solution set or in comparing the quality of the results obtained using different algorithms. The aim of our solver is to merge the algorithmic core for solving nonlinear set inversion problems with a user-friendly visualization tool aiding in the interpretation of the results.

Inner and outer interval approximations of the solution set, if precise enough, can provide a good estimate of the real shape of the set. Obtaining an almost-precise approximation using interval methods may, however, require a long computation time. With this in mind, we have extended our basic visualization tool to include several algorithms for approximating the boundary of the solution set, based on the interval boxes from the guaranteed interval approximations.

The paper is organized as follows: In Section 2 we review the SIVIA algorithm, which forms the basis of our solver. We also briefly introduce interval contractors that can enhance the efficiency of the core algorithm. Section 3 is devoted to the visualization of the results obtained by the solver and the techniques for acquiring an approximation of the actual shape of the solution set. Then, in Section 4, we discuss some implementation details regarding the structure of the solver and its components. Section 5 shows the use of our solver on concrete examples and describes an application of the solver in the visualization of complex interval arithmetic. Finally, Section 6 summarizes the article and offers some ideas for future work.

2 Nonlinear Constraints and Interval Analysis

2.1 Set Inversion via Interval Analysis

Hereinafter, boldface lowercase letters refer to intervals and interval vectors. The set of all (closed) real intervals $[a, b]$ with $a \leq b$ will be denoted by the symbol $\mathbb{IR}$. Similarly, the symbol $\mathbb{IR}^n$ will denote the set of all n-dimensional interval vectors.

Consider a set inversion problem in the form

$$\begin{aligned} c_1(x_1, \ldots, x_n) &\leq 0 \\ &\vdots \\ c_m(x_1, \ldots, x_n) &\leq 0 \\ x_i &\in \boldsymbol{d}_i \qquad\qquad \forall i \in \{1, \ldots, n\} \end{aligned}$$

where the domains satisfy $\boldsymbol{d}_i \in \mathbb{IR}$, and let us by $\mathbb{X}$ denote the solution set of this problem. Using the well-known set inversion algorithm SIVIA [12], we can find an approximation of $\mathbb{X}$ by three unions $\mathcal{S}, \mathcal{N}, \mathcal{E}$ of n-dimensional interval boxes with the properties $\mathcal{S} \subseteq \mathbb{X} \subseteq \mathcal{S} \cup \mathcal{E}$ and $\mathbb{X} \cap \mathcal{N} = \emptyset$, which form a partition of the interval box $\boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$

(also called a *paving*). The maximal width of the boxes in the set $\mathcal{E}$ (and thus the precision of the approximation) is determined by a parameter $\varepsilon > 0$ as part of the input.

The algorithm SIVIA has a branch-and-bound nature. In the course of the algorithm, we maintain a list of undetermined interval boxes (i.e., the boxes we have not assigned to $\mathcal{S}, \mathcal{N}$ or $\mathcal{E}$ yet). Initially, the set of undetermined boxes consists of the box $D = \boldsymbol{d}_1 \times \cdots \times \boldsymbol{d}_n$. Then, given an undetermined interval box from the list, we either include it into one of the resulting sets $\mathcal{S}$ or $\mathcal{N}$, or, if the length of at least one side of the box is greater than or equal to ε, bisect the box into two (smaller) undetermined boxes and iterate the procedure. If the box is undetermined, but smaller than ε, we include it into the set $\mathcal{E}$. Let us by $[c] : \mathbb{IR}^n \to \mathbb{IR}$ denote the natural interval extension of a function $c : \mathbb{R}^n \to \mathbb{R}$ obtained by replacing all occurrences of a real variable by its interval counterpart. The conditions for including an interval box $\boldsymbol{x} \in \mathbb{IR}^n$ into one of the sets $\mathcal{S}$ or $\mathcal{N}$ are the following:

- if $[c](\boldsymbol{x}) \leq 0$ (equivalently $[c](\boldsymbol{x}) \subseteq (-\infty, 0]$), then include $\boldsymbol{x}$ in $\mathcal{S}$,
- if $[c](\boldsymbol{x}) > 0$ (equivalently $[c](\boldsymbol{x}) \cap (-\infty, 0] = \emptyset$), then include $\boldsymbol{x}$ in $\mathcal{N}$.

Figure 1 shows a two-dimensional visualization of a result obtained by the SIVIA algorithm with $\varepsilon = 0.25$. The solution set is defined by the constraints

$$x_1^2 + x_2^2 \leq 16,$$
$$x_1^2 + x_2^2 \geq 9$$

over the domain $[-5, 5] \times [-5, 5]$. The set $\mathcal{S}$ is represented by green interval boxes, the set $\mathcal{E}$ by yellow boxes and the set $\mathcal{N}$ by white boxes.

It is also possible to obtain a partial visualization of a higher-dimensional problem by projecting the results onto a plane defined by two of the variables. This approach may be desirable, if some of the variables are only artificial or if we need to capture the relation between a pair of variables. The solver includes a simple algorithm, which creates a two-dimensional paving based on the projection of the interval boxes in the original paving.

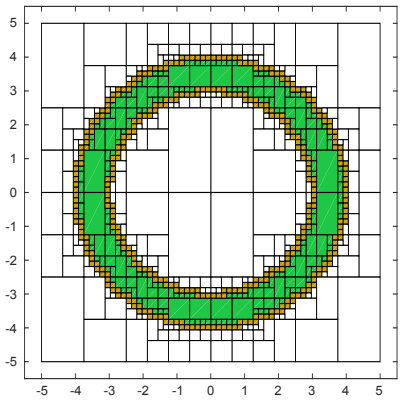

Figure 1: Visualization of a solution set obtained by the SIVIA algorithm

2.2 Contractors

The SIVIA algorithm can be improved by adding more sophisticated methods to remove points of the search space, which are not contained in the solution set of the given set inversion problem. Examples of such methods are various contractor functions that can reduce an interval box by removing some of its points inconsistent with the constraints. This may lower the number of bisections needed during the course of the algorithm and may also lead to a better approximation of the solution set.

A function $f_{\mathbb{X}}(\boldsymbol{x}) : \mathbb{IR}^n \to \mathbb{IR}^n$ is called a *contractor* for a set $\mathbb{X} \subseteq \mathbb{R}^n$ if it satisfies the following properties:

$$\forall \boldsymbol{x} \in \mathbb{IR}^n : f_{\mathbb{X}}(\boldsymbol{x}) \subseteq \boldsymbol{x}, \qquad \text{(contractance)}$$
$$\forall \boldsymbol{x} \in \mathbb{IR}^n : \boldsymbol{x} \cap \mathbb{X} \subseteq f_{\mathbb{X}}(\boldsymbol{x}). \qquad \text{(corectness)}$$

An example of a contractor, which is also implemented in our solver, is the forward-backward contractor [4, 11]. This contractor works as follows: First, the expression tree of a single constraint from the given set inversion problem is constructed. Then, the tree is traversed and the subexpression at each node is evaluated using interval arithmetic. Given an expression tree of a constraint, the forward phase uses the domains of the variables to obtain values for the intermediate nodes of the tree. These values can then be refined in the backward phase by isolating the nodes and using inverse operations, yielding reduced domains of the variables. Other supported contractors include a box-consistency based contractor [5] and a contractor exploiting the monotonicity of functions [3].

3 Visualization and Approximation

We will have a look at one of the ways how to make the approximation of a two-dimensional paving for the purposes of a visualization. We will solve approximation of the paving $(\mathcal{S}, \mathcal{N}, \mathcal{E})$ by partitioning the sets $\mathcal{S}, \mathcal{N}, \mathcal{E}$ into new sets $\mathcal{S}', \mathcal{N}', \mathcal{E}'$.

Before we start, we will introduce a few definitions. Hereinafter, we will by $\ell(X)$ denote the number of elements of a list X. The symbols $\mathcal{S}, \mathcal{N}, \mathcal{E}$, when used as input for the algorithms, will also stand for lists of interval boxes forming the corresponding unions.

We say that a box b_1 *is a neighbor to* a box b_2, if the intersection of b_1 and b_2 is a line segment or a vertex. The term *box path* will denote a list of two-dimensional boxes $(b_1, \ldots, b_k)$, where $\forall i \in \{1, \ldots, k-1\}$ it holds that b_i is a neighbor to b_{i+1} and $\forall i, j \in \{2, \ldots, k-1\} : b_i \neq b_j$ for $i \neq j$.

In the remaining definitions, elements of the set $\{s, n, e, d\}$ are used to denote the following:

- the symbols s, n, e correspond to the sets $\mathcal{S}, \mathcal{N}, \mathcal{E}$, respectively,
- the symbol d corresponds to the area out of the domain space.

A *border* is a triplet (V, L, R), where V is a list of two-dimensional points, L is a list of elements from the set $\{s, n, e, d\}$, R is a list of elements from $\{s, n, e, d\}$ and it holds that $\ell(V) = \ell(L) - 1 = \ell(R) - 1$. L_i and R_i hold indicators, which sets are to the left and the right side of the segment $V_i V_{i+1}$.

A *region* is a triplet (V, I, K), where V is a list of two-dimensional polygons, I is a list of values $\{0, 1\}$ such that $\ell(I) = \ell(V)$, K is a value from the set $\{s, n, e, d\}$. The

symbol I is an indicator, if the region lies inside, or outside of each polygon. The symbol K is an indicator, which set the region belongs to.

We will divide the complex task of the approximation into three simpler tasks:

- obtaining the borders,
- approximating the borders,
- partitioning the sets,

which will be explained in sections 3.1, 3.2 and 3.3.

3.1 Obtaining the Borders

In the first phase of the approximation process, we create a helper description of the sets $\mathcal{S}, \mathcal{N}, \mathcal{E}$: the borders. The construction of borders is formalized in Algorithm 1. A short description of all included functions follows.

Algorithm 1 Borders construction

```
1: function CreateBorders(S, N, E)
2:     D ← CreateEdgeBoxes(S, N, E)
3:     G ← GetSelectedBoxes(S, N, E, D)
4:     Graph ← CreateGraph(G)
5:     POIs ← GetPOIs(Graph, S, N, D)
6:     BoxPaths ← CreateBoxPaths(Graph, POIs)
7:     Sequences ← SampleSequences(BoxPaths)
8:     Borders ← CreateBorders(Sequences, S, N, E, D)
9:     return Borders
10: end function
```

Function CreateEdgeBoxes creates 4 one-dimensional boxes, which cover the edges of the domain.

Function GetSelectedBoxes selects one-dimensional boxes from the paving or creates new one-dimensional boxes in the domain of the paving. Control points for our approximation will be sampled from these boxes. There are several different methods that can be used for obtaining different results. For example, if we only want to get smoother sets $\mathcal{S}, \mathcal{N}, \mathcal{E}$, then an output of the function consists of:

- all one-dimensional intersections of boxes in $\mathcal{S}$ with boxes in $\mathcal{E}$,
- all one-dimensional intersections of boxes in $\mathcal{N}$ with boxes in $\mathcal{E}$,
- all one-dimensional intersections of boxes in $\mathcal{S}$ with boxes in $\mathcal{N}$,
- all one-dimensional intersections of boxes in D with boxes in $\mathcal{S}$, $\mathcal{N}$ or $\mathcal{E}$.

Function CreateGraph returns a graph, in which every box from G has a corresponding vertex. If two boxes from G are neighbors, then there exists an edge between their corresponding vertices in the graph. No additional vertices or edges exist in the graph.

Function GetPOIs creates a set of vertices of the *Graph* that were marked as points of interest (POI). A vertex is marked as POI, if it satisfies at least one of the conditions:

- it has more or less than two edges,

- its corresponding box has an intersection with a box of D,
- its corresponding box has zero volume and it has a one-dimensional intersection with at least one box from $\mathcal{S}$ and at least one box from $\mathcal{N}$.

Function CreateBoxPaths creates a set of box paths *BoxPaths* such that for each $P \in \mathit{BoxPaths}$ with $P = (b_1, \ldots, b_{\ell(P)})$ and for the corresponding vertices $(v_1, \ldots, v_{\ell(P)})$ of *Graph* the following holds:

$$\begin{array}{ll} v_i \notin \mathit{POIs}, & \forall i \in \{2, \ldots, \ell(P) - 1\} \\ v_1 \in \mathit{POIs}, & \\ v_{\ell(P)} \in \mathit{POIs}. & \end{array}$$

Function SampleSequences samples lists of points from box paths. For the box path $P = (b_1, \ldots, b_{\ell(P)})$ the list of points is $V = (p_1, \ldots, p_{2\ell(P)-1})$, where p_{2i-1} is the midpoint of b_i and p_{2i} is the intersection of b_i with b_{i+1}.

Function CreateBorders returns borders made from *Sequences*. To each segment of sequence it adds information about the sets on the sides of the segment. We can obtain this information for a segment s by constructing a perpendicular line p to s intersecting s in the middle of the segment and look for the boxes that are intersected by p to both sides from s.

3.2 Approximating the Borders

Approximation gathered in Algorithm 1 just keeps a track of box paths. In Algorithm 2 describes, how to make a smoother approximation of our set partitioning from the borders constructed in Algorithm 1. By approximation of the border (V, L, R), we mean the border (V', L, R), where V' is an approximation of V.

We can sequentially apply several different approximating or interpolating methods. The only conditions, which have to be satisfied, are that $v'_1 = v_1$, $v'_{\ell(V')} = v_{\ell(V)}$ and that the approximated borders cannot intersect themselves. To obtain the best results, the approximated sequences should not intersect each other, except in their first or last points.

During the testing, especially good results were obtained by various two-dimensional subdivision algorithms.

Algorithm 2 Borders approximation

```
1: function ApproximateBorders(Borders, DemandedApproximations)
2:     ApproxBorders ← Borders
3:     for ApproxMethod ∈ DemandedApproximations do
4:         ApproxBorders ← ApproxMethod(ApproxBorders)
5:     end for
6:     return ApproxBorders
7: end function
```

3.3 Partitioning the Sets

In this section, we use Algorithm 3 to part the domain space of the paving into regions constructed from the approximated borders. The resulting sets $\mathcal{S}'$, $\mathcal{N}'$, $\mathcal{E}'$ then consist of these constructed regions with corresponding indicators.

Algorithm 3 Regions construction

```
1: function CreateRegions(ApproxBorders)
2:     SplitBorders ← SplitOnIntersections(ApproxBorders)
3:     Regions ← GetRegions(SplitBorders)
4:     return Regions
5: end function
```

Function SplitOnIntersections finds for each two borders the intersections of all their segments. Borders having common intersection in a place other than a starting or an ending point of the sequence are split in the place of their intersection into new borders.

Function GetRegions creates regions by constructing the polygons and regions are constructed from these polygons. For construction of polygons we use the method described in [2], but we are using also counter-clockwise circled polygons. For construction of the region, we always select one clockwise circled polygon Q in function of bounding polygon, which has corresponding indicator in I equal to 1. We find all overlapping counter-clockwise circled polygons with Q that are not equivalent to Q and we include them to the region with corresponding indicator I equal to 0. Information about which set does the region belong to is obtained from the information stored for each segment.

3.4 Visualization of Regions

In this section, we present a method for obtaining a visualization based on the constructed regions, summarized in Algorithm 4. For a resulting visualization of two different approximation techniques applied to a paving generated by the SIVIA algorithm see Figure 2.

Algorithm 4 Regions visualization

```
1: function CreateVisualization(Regions, XPts, YPts, Domain)
2:     Canvas ← CreateCanvas(XPts, YPts)
3:     for Region ∈ Regions do
4:         DrawPolygons(Canvas, Region, XPts, YPts, Domain)
5:     end for
6:     for X ∈ {1, ..., XPts} do
7:         for Y ∈ {1, ..., YPts} do
8:             if IsEmpty(X, Y) then
9:                 (CX,CY) ← GetCoordinates(X, Y, XPts, YPts, Domain)
10:                Region ← FindRegion(Regions, CX, CY)
11:                FloodFill(Canvas, Region, X, Y)
12:            end if
13:        end for
14:    end for
15:    return Canvas
16: end function
```

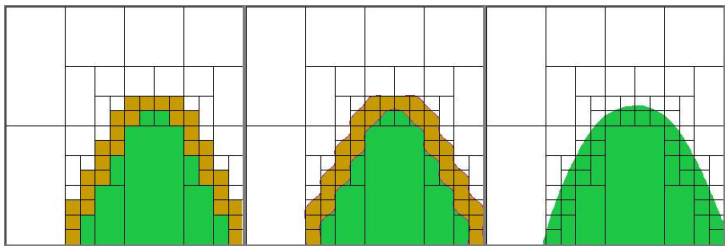

Figure 2: Two approximation techniques (middle, right) applied to a paving generated by SIVIA (left)

There are several auxiliary functions used in the visualization:

- CreateCanvas creates an empty canvas of size $XPts \times YPts$.
- DrawPolygons draws polygons of *Region* into *Canvas* with color based on the indicator K of the *Region*.
- IsEmpty checks, if a pixel *(X, Y)* in *Canvas* is empty.
- GetCoordinates returns real coordinates of the pixel *(X, Y)* in *Domain*.
- FloodFill initiates flood-fill algorithm from the pixel *(X, Y)* in *Canvas* with the color chosen according to indicator K of the *Region*.

Function FindRegion returns the region, which the coordinates *(CX, CY)* belong to. Whether the coordinates belong to a region (V, I, K) can be determined by using the Winding number algorithm [1]. If the returned values of algorithm used on polygons V agree with all corresponding indicators of I, then the coordinates belong to the region.

4 Implementation Details

The core of our solver is written mainly in MATLAB with the use of MEX (Matlab EXecutable) files for communication with the parts written in C++. Interval computations are handled by INTLAB [15] and the Boost interval arithmetic library [7]. The solver also includes an implementation of the contractors discussed in Section 2.2.

The visualization tool is written entirely in C++ and contains both the main function of C++ application and the gateway function for MEX, so it can also be used as a self-standing application outside of the MATLAB environment. The tool is accepting input generated by other solvers as well, if their solutions are pavings. The tool allows visualizing several pavings at once. The CImg Library [16] is used by the graphical user interface for image processing and interaction with the user.

Further information, as well as the source code of the solver, are available on the project website [14].

5 Examples

5.1 A Nonlinear Problem

This short example illustrates the use of our solver for visualization of the solution of a nonlinear set inversion problem. Let us consider the following problem:

$$\sin(x) + \cos(2x) \geq y,$$
$$(x, y) \in [-5, 5] \times [-5, 5].$$

We can input the constraint of a problem using the solver in the MATLAB environment as strings, with letters of the English alphabet as the names for the variables:

```
c1 = 'sin(x) + cos(2*x) >= y';
```

or as anonymous functions with declared variables:

```
c1 = @(x,y)(sin(x) + cos(2*x) >= y);
```

For more complex constraints, there is also the possibility to create a separate function, testing whether a given box satisfies the constraint, contains no values consistent with the constraint or whether it cannot be decided. The domains of the variables are contained in an interval vector, e.g. `D = [infsup(-5,5), infsup(-5,5)];`. The following lines show the basic syntax for a call of the SIVIA algorithm with a precision of 0.25 (minimal length of a side of the tested box, which can be further divided):

```
[S, N, E] = cspsivia({c1}, D, 0.25);
```

The user can further specify a division strategy (by default the longest side is chosen) or select a contractor to be used (the default setting uses no contractors).

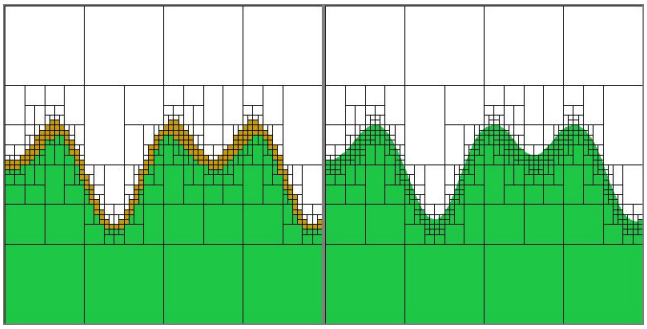

Figure 3: Visualization and approximation of the problem

The resulting sets of interval boxes describing the solution can now be passed as arguments to the visualization tool by the command

```
Runner(1, 600, 600, 'matlab', S, N, E);
```

The arguments of this command are: the number of output pavings to be visualized, the x-size and y-size of the canvas in pixels and blocks starting with the keyword 'matlab' (for sets stored in the MATLAB environment) or 'file' (for sets stored in

a text file). The visualization tool also allows various approximation techniques to be applied on the result (see Figure 3).

Smaller values of ε result in higher computation times (see Table 1 for details on the computation time using different ε).

ε (precision)	0.25	0.1	0.01
Number of boxes in $\mathcal{S}$	122	249	1989
Number of boxes in $\mathcal{N}$	119	247	1965
Number of boxes in $\mathcal{E}$	187	368	2914
SIVIA algorithm	5.320 s	11.254 s	97.775 s
Subdivision algorithm	0.190 s	0.428 s	20.421 s
Casteljau's algorithm	0.273 s	0.961 s	162.250 s

Table 1: Example computation time details

5.2 Complex Intervals

In analogy to the extension of real numbers to complex numbers, it may also be useful to define complex intervals and arithmetic operations on them. There are three main non-equivalent definition of complex intervals, which are based on different ways to express complex numbers – namely the rectangular [6], circular [9] and polar [13, 8] complex intervals. In this example, we will present a simple visualization of arithmetic operations on rectangular complex intervals, which are also supported by our solver.

Generalizing the basic definition of complex numbers, rectangular complex intervals are defined by ordered pairs of real intervals $(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{IR}^2$ as $\boldsymbol{x} = \boldsymbol{a} + \boldsymbol{b}i$, where $\boldsymbol{a}$ represents the real part of the complex interval $\boldsymbol{x}$ (denoted by $\operatorname{Re}\boldsymbol{x}$) and $\boldsymbol{b}$ represents its imaginary part (denoted by $\operatorname{Im}\boldsymbol{x}$). We will denote the set of all complex intervals by $\mathbb{IC}$. For two complex intervals $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IC}$, we define their sum or difference as the rectangular complex interval

$$\boldsymbol{x} \pm \boldsymbol{y} = (\operatorname{Re}\boldsymbol{x} \pm \operatorname{Re}\boldsymbol{y}) + (\operatorname{Im}\boldsymbol{x} \pm \operatorname{Im}\boldsymbol{y})i.$$

However, introducing complex interval multiplication (or division) in a similar manner may result in overestimation, since the set of all products (quotients) of all pairs of points in $\boldsymbol{x}$ and $\boldsymbol{y}$ is, in general, not a rectangular complex interval. The problem of describing the exact product $\{x \cdot y \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$, where $\cdot$ denotes the standard product defined on the set of complex numbers, can be formulated as a nonlinear set inversion problem.

First, we can rewrite the exact product as the set

$$\{(\operatorname{Re}x \cdot \operatorname{Re}y - \operatorname{Im}x \cdot \operatorname{Im}y) + (\operatorname{Re}x \cdot \operatorname{Im}y + \operatorname{Im}x \cdot \operatorname{Re}y)i \mid x \in \boldsymbol{x}, y \in \boldsymbol{y}\}.$$

Let us also define the rectangular product of complex intervals $\boldsymbol{x}, \boldsymbol{y}$ as the complex interval $\boldsymbol{x} \cdot \boldsymbol{y} = (\operatorname{Re}\boldsymbol{x} \cdot \operatorname{Re}\boldsymbol{y} - \operatorname{Im}\boldsymbol{x} \cdot \operatorname{Im}\boldsymbol{y}) + (\operatorname{Re}\boldsymbol{x} \cdot \operatorname{Im}\boldsymbol{y} + \operatorname{Im}\boldsymbol{x} \cdot \operatorname{Re}\boldsymbol{y})i$. The following set inversion problem can then be used to find the exact product of $\boldsymbol{x}$ and $\boldsymbol{y}$:

- variables: x_r, x_i, y_r, y_i, p_r, p_i,
- domains: $\operatorname{Re}\boldsymbol{x}$, $\operatorname{Im}\boldsymbol{x}$, $\operatorname{Re}\boldsymbol{y}$, $\operatorname{Im}\boldsymbol{y}$, $\operatorname{Re}\boldsymbol{x} \cdot \boldsymbol{y}$, $\operatorname{Im}\boldsymbol{x} \cdot \boldsymbol{y}$,
- constraints: $x_r y_r - x_i y_i = p_r$, $x_r y_i + x_i y_r = p_i$.

Our solver implements the class `rectcintval`, which can be used to work with rectangular complex intervals. It also provides the support for basic rectangular arithmetic operations (operators `+,-,*,/`), as well as the visualization of the exact complex interval products and quotients (operators `.*, ./`). The following example shows a simple use of this class with a graphical representation of the result (see Figure 4):

```
x = rectcintval(infsup(-1,5), infsup(2,3));
y = rectcintval(infsup(-6,6), infsup(1,1));
z = x .* y;
```

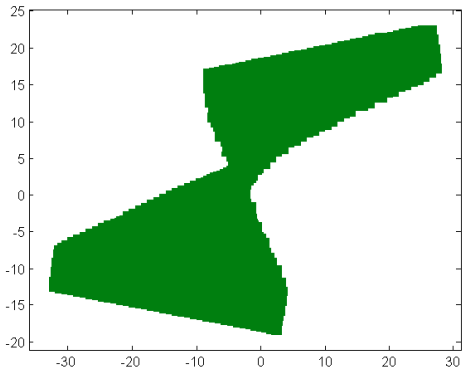

Figure 4: Visualization of complex multiplication

6 Conclusion

We have presented the algorithmic background of our tool for solving and visualizing nonlinear set inversion problems using interval methods, as well as some examples showing the simple use of the solver in the MATLAB environment. The main goal of the project is to provide the user with a piece of software, which can not only find the solution to a problem, but also aid in the interpretation of the results by providing a human-understandable visual representation of the solution set.

The possibilities for future improvements include utilizing the solver to solve subproblems in the interactive visualization. Another interesting extension might be the visualization of three-dimensional problems. The solver together with the visualization tool will be available after revision as part of the Lime toolbox [10].

Acknowledgements

We would like to thank Milan Hladík, Jaroslav Horáček and Miroslav Rada for their valuable advice and consultations. We also express our gratitude to the two anonymous referees for their helpful suggestions.

The authors were supported by the Czech Science Foundation Grant P402/13-10660S.

References

[1] David G. Alciatore and Rick Miranda. A winding number and point-in-polygon algorithm. Glaxo virtual anatomy project research report, Department of Mechanical Engineering, Colorado State University, 1995.

[2] Algorithm 101 : Finding all polygons in an undirected graph. `http://blog.reactoweb.com/2012/04/algorithm-101-finding-all-polygons-in-an-undirected-graph/`. Accessed: 2015-10-25.

[3] Ignacio Araya, Bertrand Neveu, and Gilles Trombettoni. An interval constraint propagation algorithm exploiting monotonicity. In *International workshop IntCP*, pages 65–83, 2009.

[4] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244. MIT press, 1999.

[5] Frédéric Benhamou, David A. McAllester, and Pascal Van Hentenryck. Clp(intervals) revisited. In Maurice Bruynooghe, editor, *ILPS*, pages 124–138. MIT Press, 1994.

[6] Ray E. Boche. Complex interval arithmetic with some applications. Technical Report LMSC4-22-66-1, Lockheed Missiles and Space Co., Sunnyvale, CA, USA, 1966.

[7] Hervé Brönnimann, Guillaume Melquiond, and Sylvain Pion. The design of the Boost interval arithmetic library. *Theoretical Computer Science*, 351(1):111 – 118, 2006.

[8] Yves Candau, Tarek Raissi, Nacim Ramdani, and Laurent Ibos. Complex interval arithmetic using polar form. *Reliable Computing*, 12(1):1–20, 2006.

[9] Irene Gargantini and Peter Henrici. Circular arithmetic and the determination of polynomial zeros. *Numerische Mathematik*, 18(4):305–320, 1971.

[10] Jaroslav Horáček and Milan Hladík. LIME – Library of Interval MEthods. `http://kam.mff.cuni.cz/~horacek/projekty/lime/`. Accessed: 2015-10-25.

[11] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Eric Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, London, 2001.

[12] Luc Jaulin and Eric Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, vol. 29(no. 4):1053–1064, 1993.

[13] Rudi Klatte and Christian P. Ullrich. Complex sector arithmetic. *Computing*, 24(2-3):139–148, 1980.

[14] Martin Mečiar and Elif Garajová. Interval data visualisation. `http://www.ms.mff.cuni.cz/~meciarm/iviz/`.

[15] Siegfried M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tuhh.de/rump/`.

[16] David Tschumperlé. The CImg Library. In *IPOL 2012 Meeting on Image Processing Libraries*, 2012. `http://www.cimg.eu/`.