Extended quantified set inversion algorithm with applications to control

P. Herrero[1,*], M. A. Sainz[2]

[1]Imperial College, London, SW7 2AZ, United Kingdom

[2]University of Girona, Girona, 17004, Spain

[*]Corresponding author: Herrero, Pau, e-mail: p.herrero-vinias@imperial.ac.uk

In memoriam to Irene A. Sharaya

The Quantified Set Inversion (QSI) algorithm is a set inversion algorithm based on Modal Interval Analysis and designed for estimation of AE-solution sets to parametric non-linear systems, i. e., for the solution of quantified real constraint (QRC) problems. However, the original QSI algorithm is limited to the QRC problems where existentially quantified variables are not shared between equality constraints. This paper presents an extended version of the QSI algorithm that overcomes some of these limitations. In addition, we introduce a user-friendly Matlab toolbox including a modal interval arithmetic, an efficient implementation of an algorithm for performing modal interval computations (f^*-algorithm) and the QSI algorithm. Due to the high popularity of Matlab in the scientific and engineering communities, the presented toolbox is expected to promote the use of Modal Interval Analysis. Finally, several examples of using the Matlab toolbox and applications to control engineering are presented.

Keywords: constraint satisfaction problem, modal interval analysis, quantified solutions, AE-solutions, set inversion, control systems.

Introduction

Solving Quantified Real Constraints over the reals

A Quantified Real Constraints (QRC) is a mathematical formalism used for modelling many real-life problems that involve systems of nonlinear equations linking real variables, some of them affected by logical quantifiers. QRCs appear in numerous contexts, such as Control Engineering, Electrical Engineering, Mechanical Engineering, and Biology [1]. Solving QRC is an active research area in which two radically different approaches have been proposed: the symbolic quantifier elimination [2, 3] and approximate methods [4 − 11]. However, no matter which approach is actually used, the solution of large problems within a reasonable computational time as well as processing the general case still remain the challenge to our capabilities.

Modal Interval Analysis

Modal Interval Analysis (shortly MIA, see [12]) is an extension of the classical Interval Analysis (see, e. g., [13, 14]) in which any interval $[a, b]$ is assigned a *modality* to represent its quantification by logical quantifiers "$\forall$" and "$\exists$". The interval is *proper* if $a \le b$ or *improper* if $a \ge b$. Modal Interval Analysis uses the so-called Kaucher arithmetic [15], also referred to as complete interval arithmetic, to perform interval computations. An important operator of this interval arithmetic is the dualization, denoted by "Dual" and defined as

$$\mathrm{Dual}([a, b]) \;=\; [b, a].$$

Two major results of Modal Interval Analysis are so-called the **-semantic theorem* and the ***-semantic theorem*, which enable us to prove either satisfaction or dissatisfaction of logic AE-formulas over the reals by means of interval computations (see Appendix). In particular, for any continuous real function f, there holds

$$\mathrm{Out}(f^*(\boldsymbol{P},\, \boldsymbol{V})) \subseteq [0, 0] \quad\Rightarrow\quad \forall(p, \boldsymbol{P}')\, \exists(v,\, \boldsymbol{V}')\; f(p, v) = 0,$$

where $\forall(p, \boldsymbol{P}')$ means $\forall p \in \boldsymbol{P}'$ and $\exists(v,\, \boldsymbol{V}')$ means $\exists v \in\, \boldsymbol{V}'$,
p and v are vectors of variables ranging over the real domains $\boldsymbol{P}'$ and $\boldsymbol{V}'$,
$\boldsymbol{P}$ is a vector of proper intervals, $\boldsymbol{V}$ is a vector of improper intervals and
$\mathrm{Out}(f^*(\boldsymbol{P},\, \boldsymbol{V}))$ is an outer approximation of the so-called *-semantic extension of f, which can be computed by using the modal interval arithmetic.

Note that the prime symbol on a variable (e. g., $\boldsymbol{P}'$) indicates a real domain, while the same variable without the prime symbol indicates a modal interval.

It is worth noting that there exists an efficient algorithm, referred to as f^*-algorithm, for computing inner and outer approximations of f^* [12]. A MATLAB implementation of the f^*-algorithm is also available [16] (see Section 3.2). We recommend the reader to consult Appendix for a short introduction to Modal Interval Analysis and the book [12] for a thorough exposition.

1. Quantified Set Inversion Algorithm

The Quantified Set Inversion (QSI) algorithm [10] is a numerically guaranteed approximate method based on set inversion [17] and Modal Interval Analysis and designed for estimation of the AE-solution sets to the problems with quantified real constraints. These solution sets, introduced in [5], are defined as

$$\varXi_{AE} \;=\; \bigl\{\, x \in \boldsymbol{X}_0' \mid \forall(p, \boldsymbol{P}')\, \exists(v,\, \boldsymbol{V}')\, \bigl(f_1(x, p, v) = 0 \;\wedge \ldots \wedge\; f_m(x, p, v) = 0\bigr) \,\bigr\}, \tag{1}$$

where $f_1, \ldots, f_m$ are functions from $\mathbb{R}^n$ to $\mathbb{R}^m$, x denotes a vector of free variables varying within the interval box $\boldsymbol{X}_0'$, p is a vector of universally quantified variables varying within the interval box $\boldsymbol{P}'$, and v is a vector of existentially quantified variables varying within $\boldsymbol{V}'$. In the sequel, we will denote $f = (f_1, f_2, \ldots, f_m)^\top$ for brevity. Applicability of QSI algorithm requires that each existentially quantified variable from v occurs in only one of the components $f_1, \ldots, f_m$.

In general, the set $\varXi_{AE}$ may have complex structure, which makes it impossible to describe it precisely and in a moderate number of operations. So, we need to estimate $\varXi_{AE}$, i. e., to present its approximate descriptions that meets requirements of the practice (see [5]). In order to produce an estimate of the solution set Ξ_{AE}, QSI algorithm sequentially divides the initial domain $\boldsymbol{X}'_0$ into three regions ('true', 'false', 'undefined') by means of the following instructions, hereinafter called "bounding rules":

1. The rule *Inside*: An interval box $\boldsymbol{X}$ is included in Ξ_{AE} if it renders the following formula true:

$$\mathrm{Out}\big(f_1^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{V})\big) \subseteq [0,0] \ \wedge \ldots \wedge \ \mathrm{Out}\big(f_m^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{V})\big) \subseteq [0,0], \tag{2}$$

   where $\boldsymbol{X}$ and $\boldsymbol{P}$ are proper intervals and $\boldsymbol{V}$ is an improper one, since inclusion (2) implies

$$\forall(x, \boldsymbol{X}')\,\forall(p, \boldsymbol{P}')\,\exists(v, \boldsymbol{V}')\, f(x,p,v) = 0.$$

   Therefore, the box $\boldsymbol{X}$ is labeled as 'true'. Note that, in Modal Interval Analysis, the inclusion in zero is defined.

2. The rule *Outside*: An interval box $\boldsymbol{X}$ is excluded from Ξ_{AE} if it renders the following formula true:

$$\mathrm{Inn}\big(f_1^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{V})\big) \nsubseteq [0,0] \ \vee \ldots \vee \ \mathrm{Inn}\big(f_m^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{V})\big) \nsubseteq [0,0], \tag{3}$$

   where $\boldsymbol{X}$, $\boldsymbol{V}$ are improper intervals, $\boldsymbol{P}$ is a proper one and Inn is an inner approximation of f^*. The point is that non-inclusion (3) implies

$$\exists(p, \boldsymbol{P}')\forall(x, \boldsymbol{X}')\,\forall(v, \boldsymbol{V}')\, f(x,p,v) \neq 0.$$

   Then, the box $\boldsymbol{X}$ is labeled as 'false'.

3. The rule *Undefined*: If none of the above is satisfied, then the interval box $\boldsymbol{X}$ is labeled as 'undefined' and bisected. The entire process iterates with the resulting boxes until a predefined precision ϵ is reached.

Algorithm 1. QSI algorithm

```
Require: $f(x,p,v) = 0$; $\boldsymbol{X}'$, $\boldsymbol{P}'$, $\boldsymbol{V}'$; $\epsilon$.
Ensure: $AE_{\mathrm{Inn}}$ and $AE_{\mathrm{Out}}$ of the solution set.
 1: $List = \{\boldsymbol{X}'\}$; $AE_{\mathrm{Inn}} = \{\emptyset\}$; $\triangle\Xi_{AE} = \{\emptyset\}$;
 2: while $List$ is not empty do
 3:    Dequeue $\boldsymbol{X}'$ from $List$;
 4:    if $Inside$ is true for $\boldsymbol{X}'$ then
 5:       Enqueue $\boldsymbol{X}'$ to $AE_{\mathrm{Inn}}$;
 6:    else if $Outside$ is true for $\boldsymbol{X}'$ then
 7:       Enqueue $\boldsymbol{X}'$ to $AE_{\mathrm{Out}}$;
 8:    else if $d(\boldsymbol{X}') < \epsilon$ then
 9:       Enqueue $\boldsymbol{X}'$ to $\triangle\Xi_{AE}$;
10:    else
11:       Bisect $\boldsymbol{X}'$ and enqueue the resulting boxes to $List$;
12:    end if
13: end while
```

Algorithm 1 shows the QSI algorithm in pseudo-code form, and we use the following notation there:

- *List* is a list of boxes;
- AE_{Inn} is a list of such boxes that $AE_{\mathrm{Inn}} \subseteq \Xi_{AE}$;

- AE_{Out} is a list of such boxes that $\varXi_{AE} \subseteq AE_{\text{Out}}$;
- *Inside* and *Outside* are inclusion and exclusion conditions expressed by formulas (2) and (3) respectively;
- Enqueue/Dequeue means the result of adding/extracting a box to/from the list;
- $d(\boldsymbol{X}')$ is a function returning the widest relative width of $\boldsymbol{X}'$ with respect to the original box;
- ϵ is a real value representing the desired precision.

1.1. Examples

1.1.1. Linear case

First, we consider the following AE-solution set of an interval linear system proposed by Irene Sharaya in [18] and defined as

$$\begin{aligned} \varXi_{\exists\exists} = \Big\{\ (x_1, x_2, x_3) \in (\boldsymbol{X}_1', \boldsymbol{X}_2', \boldsymbol{X}_3') \mid\ & \exists(a_{11}, [-1,1])\exists(a_{12}, [-2,2])\exists(a_{13}, [-2,2]) \\ & \exists(a_{21}, [-2,2])\exists(a_{22}, [-1,1])\exists(a_{23}, [-2,2]) \\ & \exists(a_{31}, [-2,2])\exists(a_{32}, [-2,2])\exists(a_{33}, [-1,1]) \\ & \quad a_{11}x_1 + a_{12}x_2 + a_{13}x_3 - 2 = 0 \\ & \wedge\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 - 2 = 0 \\ & \wedge\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 - 2 = 0\ \Big\}. \end{aligned} \tag{4}$$

This is, in fact, a piece of the united solution set to the interval linear system of equations

$$\begin{pmatrix} [-1,1] & [-2,2] & [-2,2] \\ [-2,2] & [-1,1] & [-2,2] \\ [-2,2] & [-2,2] & [-1,1] \end{pmatrix} x = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

contained in the interval box $(\boldsymbol{X}_1', \boldsymbol{X}_2', \boldsymbol{X}_3')$. If the box is "large enough", then $\varXi_{\exists\exists}$ coincides with the entire box $(\boldsymbol{X}_1', \boldsymbol{X}_2', \boldsymbol{X}_3')$ with a star removed around the origin of the coordinates.

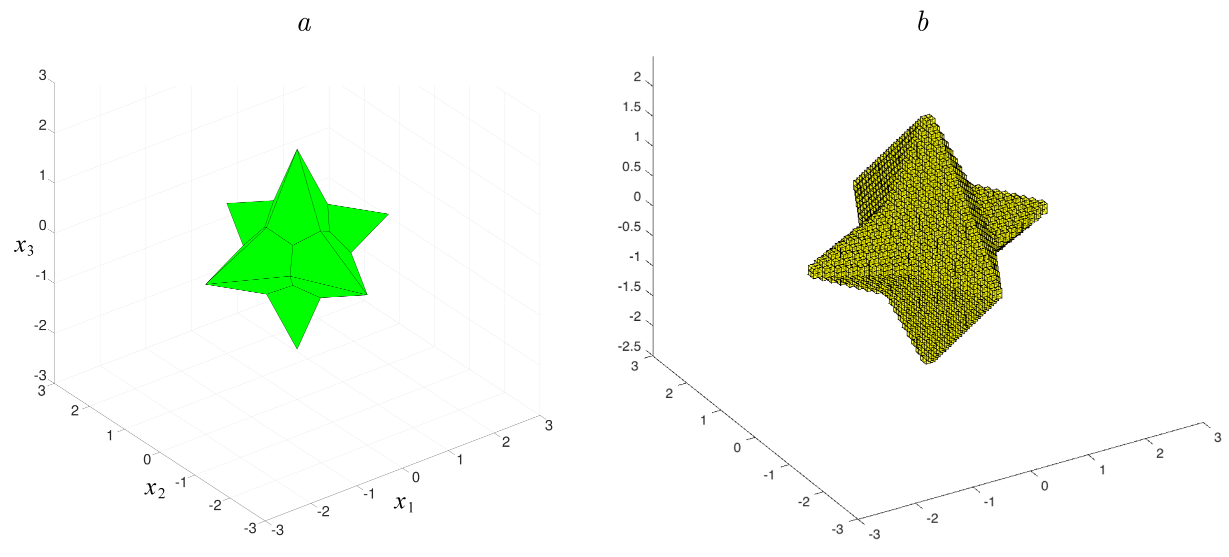

Fig. 1. The solution sets of (4) obtained by `IntLinInc3D` and QSI algorithm: *a* — solution by `IntLinInc3D`, *b* — solution by QSI (undefined boxes)

Note that all the equations in (4) are independent from each other, and each existentially quantified parameter occurs in only one equation. Under these conditions, we can avail ourselves of Theorem 3.6 from [5] stating that the intersection of the solution set with every orthant is a convex polyhedral set whose vertices are the solutions of the extreme point linear systems $Ax = b$, with A and b formed by the bounds of the intervals from $\boldsymbol{A}$ and $\boldsymbol{b}$. Using the MATLAB based package `IntLinInc3D` developed by Irene Sharaya [18, 19], one can visualize the solution set, and it is shown in Fig. 1, *a*. The picture was obtained in a computation time of 0.15 seconds on an Intel Core i5 3.4 GHz. An outer approximation of the same solution set was obtained by means of a MATLAB implementation of QSI algorithm (see Section 3) with $\epsilon = 0.02$ in a computation time of 20 minutes on an Intel Core i5 3.4 GHz. A graphical representation of our approximation in $\mathbb{R}^3$ is shown in Fig. 1, *b*. Note that only 'undefined' boxes have been plotted there for visualization purposes.

A comparison between the two methods shows that the approach proposed by Irene Sharaya (it is called "the boundary intervals method") is much more efficient in solving linear problems with each existentially quantified parameter occurring in only one equation. However, for non-linear problems, Sharaya's technique is not applicable, while QSI algorithm is quite suitable.

1.1.2. Non-linear case

Next, we consider a robust control problem from [6]. It requires computing the set of all feasible parameter vectors c for a parametric linear PI controller $\Gamma(c)$ that robustly stabilizes an uncertain linear time-invariant model of a process. This problem reduces to estimation of the following solution set

$$\Xi_{AE} = \left\{ c \in \boldsymbol{C}' \mid (\forall p \in \boldsymbol{P}')\, f(c,p) < 0 \right\}, \tag{5}$$

where $f(c,p)$ is obtained by the Routh criterion,

$$f(c,p) \triangleq \min \begin{pmatrix} p_2 + y_1 \\ p_2 p_3 + 1 + y_2 \\ p_2 p_3^2 + p_3 - \dfrac{p_2\left(p_3^2 + c_2 p_1 p_3^2\right)}{p_2 p_3 + 1} + y_3 \\ p_3^2 + c_2 p_1 p_3^2 - \dfrac{\left(p_2 p_3 + 1\right)^2 \left(c_1 p_1 p_3^2\right)}{\left(p_2 p_3^2 + p_3\right)\left(p_2 p_3 + 1\right) - p_2\left(p_3^2 + c_2 p_1 p_3^2\right)} + y_4 \\ c_1 p_1 p_3^2 + y_5 \end{pmatrix}. \tag{6}$$

For $\boldsymbol{C}' = [0,1] \times [0,1]$, $\boldsymbol{P}' = [0.9, 1.1] \times [0.9, 1.1] \times [0.9, 1.1]$, and $\epsilon = 0.02$, the result is shown in Fig. 2. It is obtained by QSI algorithm in 430 seconds on an Intel Core i5 3.4 GHz, where red zone indicates 'true' boxes, yellow means 'undefined' boxes and blue indicates 'false' boxes.

1.2. The general case

In the general vector case, where the same existential variables from $c \in \boldsymbol{C}'$ appear in multiple constraints, the problem can be converted to the one-dimensional case $m = 1$ by means of various tricks.

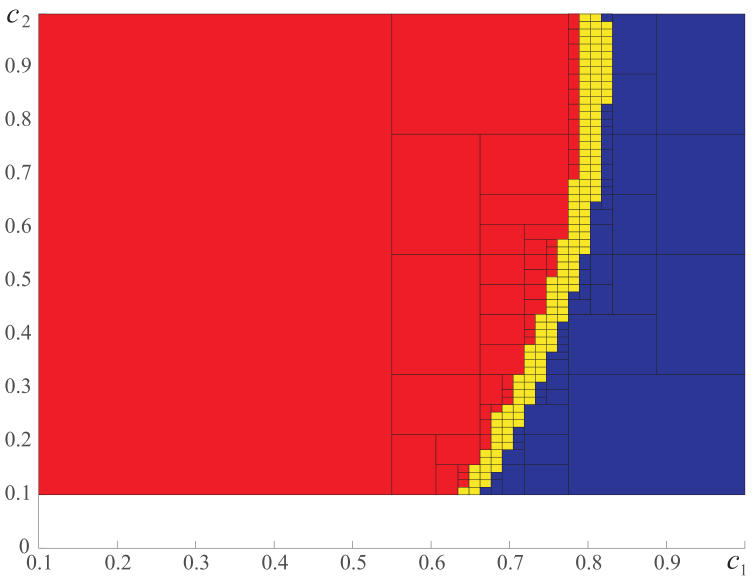

Fig. 2. A piece of an approximation of the AE-solution set defined by (5), (6)

In the case of inequality constraints, the problem can be converted by using the min/max function:

$$f(x,p,c,v) = \min\left\{f_1(x,p,c,v),\ldots,f_m(x,p,c,v)\right\}.$$

In the case of equality constraints, one may use the sum-of-squares function

$$f(x,p,c,v) = f_1^2(x,p,c,v) + \ldots + f_m^2(x,p,c,v),$$

or to the sum-of-absolute-values function

$$f(x,p,c,v) = |f_1(x,p,c,v)| + \ldots + |f_m(x,p,c,v)|,$$

since

$$f(x,p,c,v) = 0 \;\Leftrightarrow\; \big(f_1(x,p,c,v) = 0 \wedge \ldots \wedge f_m(x,p,c,v) = 0\big).$$

However, for the equality constraints, the above reformulation strategy poses significant problems. First of all, it is computationally very expensive to prove the inclusion in the zero, except for trivial cases, since f is a non-negative function and the f^*-algorithm only provides $\subseteq$-inner and $\subseteq$-outer approximations for f^*. Moreover, f is not monotonic for the values close to 0, and this is why reaching any of such inclusions usually requires long computations. Nevertheless, the "reformulation strategy" can be useful to prove that an expression is not included in zero, since

$$f^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\boldsymbol{V}) \not\subseteq [0,0] \;\Leftrightarrow\; \big(f_1^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\boldsymbol{V}) \not\subseteq [0,0] \vee \ldots \vee f_m^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\boldsymbol{V}) \not\subseteq [0,0]\big),$$

which is computationally tractable.

It is clear that in order to solve the general vector problems involving equality constraints with shared existential variables, a new algorithm is required. The next section introduces an extension of QSI algorithm that overcomes some of this limitations.

2. Extended Quantified Set Inversion Algorithm

As previously mentioned, in the general case, when the existentially quantified variables occur in several components, the *Inside* rule of the QSI algorithm is a necessary, but not sufficient condition for proving that an interval box $\boldsymbol{X}$ is included in AE-solution set described by equation (1). In order to make it a necessary condition, an additional requirement needs to be satisfied.

For the sake of simplicity, let us consider the particular case of two restrictions $f = (f_1, f_2)$ and one existentially quantified variable $c \in \boldsymbol{C}'$ that has occurrences in both f_1 and f_2. According to Theorem 4.4.3 from [12], if the improper interval $\boldsymbol{C}$ is transformed into its dual in f_1 or f_2 in all of its occurrences except one, then $\boldsymbol{X}$ is included in $\varXi_{AE}$ providing that the following condition is satisfied

$$Inside : \operatorname{Out}\big(f_1^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{C}, \boldsymbol{V})\big) \subseteq [0,0] \ \wedge\ \operatorname{Out}\big(f_2^*(\boldsymbol{X}, \boldsymbol{P}, \operatorname{Dual}(\boldsymbol{C}), \boldsymbol{V})\big) \subseteq [0,0] \qquad (7)$$

(we can take dualization of $\boldsymbol{C}$ in f_1^* instead of f_2^*), where $\boldsymbol{X}$ and $\boldsymbol{P}$ are proper intervals and $\boldsymbol{C}$ and $\boldsymbol{V}$ are improper ones. The latter corresponds to the semantic

$$\forall(x_1, \boldsymbol{X}'_1)\,\forall(x_2, \boldsymbol{X}'_2)\,\forall(p, \boldsymbol{P}')\,\exists(c, \boldsymbol{C}')\,\exists(v, \boldsymbol{V}') \\ \big(f_1^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{C}, \boldsymbol{V}) = 0 \ \wedge\ f_2^*(\boldsymbol{X}, \boldsymbol{P}, \boldsymbol{C}, \boldsymbol{V}) = 0\big).$$

However, since $\operatorname{Dual}(\boldsymbol{C})$ is a proper interval, the inclusion of $f_2^* \subseteq [0,0]$ can be hardly satisfied. One way to alleviate this problem is to implement yet another branch-and-bound algorithm over the interval $\boldsymbol{C}$ that looks for a sub-box of $\boldsymbol{C}$ satisfying the condition (7). This branch-and-bound (B&B) algorithm is similar to QSI algorithm, but c is a free variable vector, x and p are vectors of universally quantified variables, and v is a vector of existentially quantified variables. If condition (7) is satisfied for a sub-box of $\boldsymbol{C}$ resulted from the subdivision of $\boldsymbol{C}$, then B&B algorithm stops and returns 'true'. If all sub-boxes of $\boldsymbol{C}$ are labeled as 'false' by the *Outside* rule (3), or they reach a predefined precision ϵ_2, then the algorithm returns 'undefined'. Since the B&B algorithm can be computationally quite expensive, it is only launched if *Inside* rule (necessary condition) is satisfied. Therefore, the *Inside* rule for the extended QSI (*Extended Inside*) is composed of the original *Inside* rule and the B&B algorithm over $\boldsymbol{C}$. Algorithm 2 below shows the *Extended Inside* rule in a pseudo-code form.

In Algorithm 2:
- *List* means a list of boxes;
- *Inside* is inclusion expressed by condition (2);
- *Included* is inclusion expressed by condition (7);
- Enqueue/Dequeue means the result of adding/extracting a box to a list;
- $d(\boldsymbol{C}')$ is a function returning the widest relative width of $\boldsymbol{C}'$ with respect to the original box;
- ϵ_2 is a real value representing the required precision.

Algorithm 2. Inside rule for Extended QSI algorithm

```
Require: $f(x,p,v,c)=0$; $\boldsymbol{C}'$, $\boldsymbol{X}'$, $\boldsymbol{P}'$, $\boldsymbol{V}'$; $\epsilon_2$.
Ensure: Consistency
 1: $List = \{\boldsymbol{C}'\}$;
 2: Consistency=undefined;
 3: if $Inside$ is true for $\boldsymbol{X}'$ then
 4:    while $List$ is not empty do
 5:       Dequeue $\boldsymbol{C}'$ from $List$;
 6:       if $Included$ is true for $\boldsymbol{C}'$ then
 7:          Consistency = ‘true’;
 8:          break
 9:       else if $Outside$ is true for $\boldsymbol{C}'$ or $d(\boldsymbol{C}') < \epsilon_2$ then
10:          Eliminate $\boldsymbol{C}'$ from $List$;
11:       else
12:          Bisect $\boldsymbol{C}'$ and enqueue the resulting boxes to $List$;
13:       end if
14:    end while
15: end if
```

The *Extended Outside* rule for the extended QSI is the same as the original *Outside* rule

$$\mathrm{Inn}\big(f_1^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\,\boldsymbol{V})\big) \not\subseteq [0,0] \;\vee\; \mathrm{Inn}\big(f_2^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\boldsymbol{V})\big) \not\subseteq [0,0], \tag{8}$$

but supplemented with the following constraint

$$\mathrm{Inn}\big(f^*(\boldsymbol{X},\boldsymbol{P},\boldsymbol{C},\boldsymbol{V})\big) \not\subseteq [0,0], \tag{9}$$

where f is the sum-of-absolute values of f_1 and f_2 (see Section 1.2), $\boldsymbol{X}$, $\boldsymbol{P}$ are proper intervals and $\boldsymbol{C}$, $\boldsymbol{V}$ are improper ones. This additional constraint allows to eliminate boxes $\boldsymbol{X}$ that otherwise would not be eliminated.

Since *Extended Inside* rule is more computationally expensive than the *Extended Outside* rule, the latter is applied before the former within the extended QSI algorithm. Note that the *Extended Outside* could also be employed within the *Extended Inside* rule in place of the *Outside* rule. However, due to the additional computational cost of computing and testing inclusion (9), we do not use this alternative. Finally, note that the extended QSI can be easily generalized to m functions with multiple existentially quantified variables shared among several component expressions. However, its practical applicability might be limited due to the exponential complexity.

2.1. Application to Control Engineering

Let us be given the second-order single-input and single-output (SISO) system presented in [20]. The pole placement problem for such system is equivalent to the computations of the following AE-solution set:

$$\begin{aligned} \varXi_{AE} = \big\{\, (x_1,x_2) \in (X_1',X_2') \mid {}& \forall(q_1,\boldsymbol{Q}_1')\,\forall(q_2,\boldsymbol{Q}_2')\,\forall(q_3,\boldsymbol{Q}_3')\,\exists(\zeta,\boldsymbol{Z})\,\exists(\omega,\boldsymbol{\Omega}) \\ & \omega^2-(q_1+q_2+q_3)(q_1+q_3-x_2)+q_1^2q_2^2q_3-q_2q_3x_1=0 \\ & \wedge\; 2\zeta\omega - x_2 + 2q_1 + q_2 + 2q_3 = 0 \,\big\}, \end{aligned} \tag{10}$$

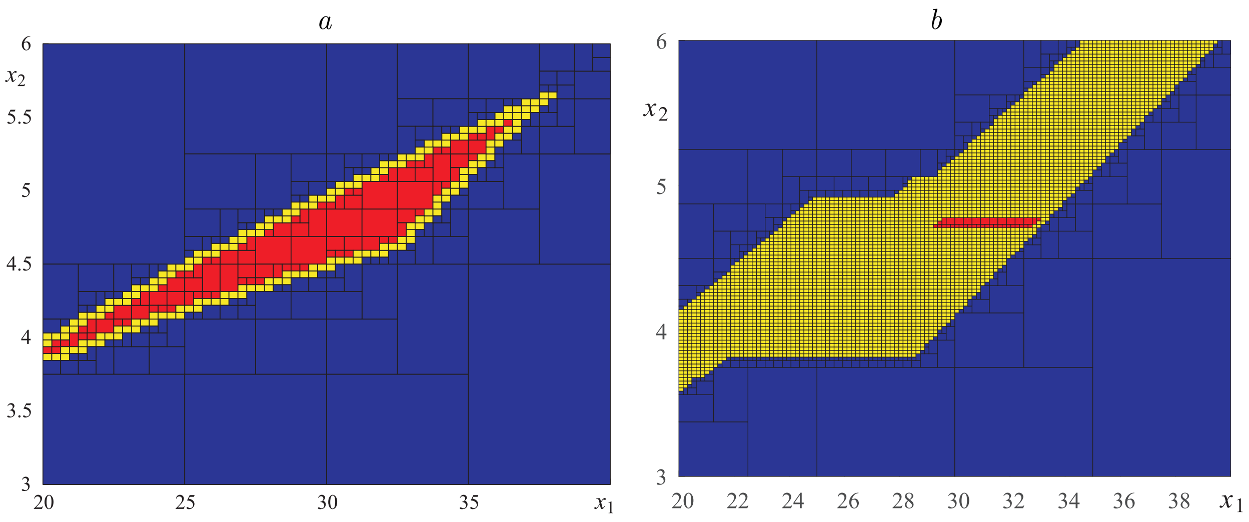

Fig. 3. Comparison of usual QSI and Extended QSI algorithms: *a* — estimation of solution set (10) by usual QSI, *b* — estimation of set (11) by extended QSI

where $q = (q_1, q_2, q_3)$ are the coefficients of the characteristic polynomial (they are considered uncertain), ζ and ω are coefficients of the design polynomial, and x_1 and x_2 are design parameters. Note that there exists the existentially quantified parameter ω occurring in both terms of the conjunction. Therefore, the extended QSI algorithm needs to be applied.

However, in the above particular problem, the existential variable ω can be eliminated, and we arrive at the equivalent AE-solution set

$$\begin{aligned}\Xi_{AE} = \big\{(x_1, x_2) \in (X_1', X_2') \mid \forall(q_1, \boldsymbol{Q}_1')\,\forall(q_2, \boldsymbol{Q}_2')\,\forall(q_3, \boldsymbol{Q}_3')\,\exists(\zeta, \boldsymbol{Z})\,\exists(\omega, \boldsymbol{\Omega})&\\ \omega^2 - (q_1 + q_2 + q_3)(q_1 + q_3 - x_2) + q_1^2 q_2^2 q_3 - q_2 q_3 x_1 = 0&\\ \wedge((x_2 - 2q_1 - q_2 - 2q_3)/(2\zeta))^2&\\ -(q_1 + q_2 + q_3)(q_1 + q_3 - x_2) + q_1^2 q_2^2 q_3 - q_2 q_3 x_1 = 0\,\big\},&\end{aligned} \tag{11}$$

which is amenable to the original QSI algorithm. The entire example is useful for validation purposes because we can compare the result with the one obtained by the extended QSI algorithm applied to (10).

For the intervals $\boldsymbol{\Omega} = [0.5, 2]$, $\boldsymbol{Z} = [0, 5]$, $\boldsymbol{Q}_1 = [0.4, 0.54]$, $\boldsymbol{Q}_2 = [0.5, 0.54]$, $\boldsymbol{Q}_3 = [0.5, 0.54]$ and $\epsilon = \epsilon_2 = 0.02$, QSI algorithm takes 8 min to obtain an estimate of the AE-solution set defined by equality (11). It is shown in Fig. 3, *a*. At the same time, for $\epsilon = \epsilon_2 = 0.01$, the extended QSI algorithm takes 60 min to obtain the estimate of the AE-solution set defined by (10) and shown in Fig. 3, *b*. In the pictures, the red zone indicates 'true' boxes, yellow means 'undefined' boxes and blue indicates 'false' boxes.

Taking a look at Fig. 3, *a* and *b* and evaluating the computation times, one can infer that the solution provided by the Extended QSI algorithm is correct, but less accurate and more computationally expensive than the one obtained with the original QSI algorithm for an equivalent problem without common existentially quantified variable in the conjunction terms (equality (10)). However, the Extended QSI algorithm is able to provide an approximate solution to a class of problems that the original QSI algorithm cannot solve.

3. Efficient Matlab Implementation

High-level numerically oriented (HLNO) programming languages such as MATLAB, Scilab or Octave are popular and well-established tools in the scientific and engineering communities. However, their computational efficiency sometimes limits their use in certain areas where intensive numerical computations are required, such as some problems of interval analysis. On the other hand, HLNO programming languages are well-renowned for their efficiency in vectorial and matrix computations. With this idea in mind, we have developed a novel implementation of f^*-algorithm [12], which is utilized within QSI algorithm [10], that aims at minimizing the use of explicit loops in the code and favours the use of vectorial computations. This idea has been previously employed for implementing a vectorial version of the so-called Set Inversion Via Interval Analysis (SIVIA) algorithm [21].

3.1. Vectorial Modal Interval Arithmetic

In order to operate with modal intervals, a modal interval arithmetic (also referred to as Kaucher arithmetic [15]) was implemented in MATLAB in vectorial manner. Note that such interval arithmetic operators do not use any explicit “for” loops in the MATLAB code. For this purpose, MATLAB logical indexing and the *bsxfun* built-in function were employed. In order to facilitate writing interval arithmetic expressions, a MATLAB class representing an interval vector was implemented. This class overloads all the arithmetic operators in order to operate with interval vectors.

To correct possible containment failures caused by rounding in the floating-point arithmetic, the rounding mode of the computer is changed to $-\inf$, when the lower bounds are computed, and changed to $+\inf$, when the upper bounds are computed. It is important to note that, by doing vectorial computations, the number of times the rounding mode is required to be change is much less than that with the original implementation, hence it improves the computation time. An example of using modal interval arithmetic can be found in Algorithm 3.

Algorithm 3. Example of Modal Interval arithmetic in MATLAB.

```
% variables
x = interval([1,2;4,3]);
y = interval([6,5;7,8]);
% function
z = sin(x/dual(y))
% OUTPUT
-----------------------------------------------------------------------------
 >> z =  [0.1659, 0.3895; 0.5409, 0.3663]
```

3.2. Vectorial implementation of f^*-algorithm

The basic idea behind the vectorial implementation of f^*-algorithm [12] is to evaluate all interval boxes within the branch-and-bound algorithm (i. e., cells) in a vectorial way instead of processing them one by one. Although not completely eliminated, the number of explicit loops in the code was reduced to a minimum. An open-source MATLAB implementation of the f^*-algorithm, referred to as Modal Interval Solver (MIC), can be downloaded from [16]. It provides a user-friendly interface to operate with modal intervals. Note that the

f^*-algorithm can also be used for approximating the range of a continuous function when only classic intervals are involved. An example of using MIC can be found in Algorithm 4.

Algorithm 4. Example of Modal Interval Calculator (MIC) utilization.

```
% proper variables
var{1} = {'x1',[4,0]};
var{2} = {'x2',[2,8]};
var{3} = {'x3',[-4,9]};
% improper variables
var{4} = {'x4',[3,-1]};
% function
f = 'x1^2+(x1+x2)^2+(x1+x2+x3)^2+(x1+x2+x3+x4)^2';
% parameters: tolerance and epsilon
param.tol = 1e-3;
param.eps = 1e-3;
% call mic - % inn: inner approximation; out: outer approximation
[inn,out] = mic(vars,f,param);
% OUTPUT
------------------------------------------------------------------------------
Total Elapsed Time(s): 5.3
Total Iterations: 50
f*= [[81.1653,609],[81,609]]
Tolerance= 0.1652
```

3.3. Parallel implementation of QSI

Both SIVIA and QSI algorithms are easy to parallelize due to its branch-and-bound nature. The Matlab Parallel Toolbox, and in particular the *parloop* command, was employed to speed up the computations of the extended QSI algorithm. It is important to note that the improvement in speed of the parallel implementation depend on the number of available cores in the computer running the algorithm. An open-source MATLAB implementation, for both the original and extended versions, of the QSI algorithm is available from the authors upon request. An example of using QSI that correspods to Example 1.1.2 can be found in Algorithm 5.

Algorithm 5. Program source for Example 1.1.2.

```
% Free variables
VARS{1} = {'c1',[0.1,1],'F'};
VARS{2} = {'c2',[0.1,1],'F'};
% Quantified variables
VARS{3} = {'p1',[0.9,1.1],'U'};
VARS{4} = {'p2',[0.9,1.1],'U'};
VARS{5} = {'p3',[0.9,1.1],'U'};
% Constraints
C{1} = {'c1*p1*p3^2',[0,inf]};
C{2} = {'(p2*p3+1)^2 - p2*(p3+c2*p1*p3)',[0,inf]};
C{3} = {'(1+c2*p1)*((p2*p3^2+p3)*(p2*p3+1)-p2*(p3^2+c2*p1*p3^2))-...
         (p2*p3+1)^2*(c1*p1)',[0,inf]};
% Call QSI: S: solution N: non-solution; U: undefined
eps=0.02;
[S,U,N] = qsi(C,VARS,[],eps,true);
% Plot results
draw_boxes(S',U',N',1,2);
```

Conclusion

Approximate estimation of AE-solution sets to parametric non-linear systems by means of numerical methods, such as those suggested by interval analysis, is still an open problem. This paper presents an extended version of the Modal-Interval-based Quantified Set Inversion (QSI) algorithm that overcomes some of the limitations of the original QSI algorithm occurred in the solution of the equality constraints with common existentially quantified variables.

When compared to the original QSI algorithm, the extended QSI provides approximations of the solution set with a lower resolution. This is explained by the fact that the conditions needed to proof that an interval box is a part of the solution set are in general more difficult to be satisfied. This is accompanied by a significant increase in the computation time, which limits the utilization of extended QSI to low-dimension problems. A user-friendly MATLAB toolbox including a modal interval arithmetic; an algorithm to perform complex model interval computations (i. e., f^*-algorithm); and the QSI algorithm is provided. It is important to note that such a toolbox allows to compute with classic intervals. In this particular case, the f^*-algorithm becomes an algorithm for approximating the range of a continuous function and the QSI becomes the so-called SIVIA algorithm. We believe that this toolbox can significantly help to promote the utilization of Modal Interval Analysis within the scientific community.

Appendix — Modal Interval Analysis

The basic object of the Modal Interval Analysis (shortly MIA, see [12]) is the *modal interval* $\boldsymbol{A}$ which is a pair formed by a classic interval $A' = [a_1, a_2]'$, its *domain*, and a logic quantifier $\forall$ or $\exists$, its *modality*. A modal interval $(\boldsymbol{A}, \exists)$ is called *proper interval*, and it is represented by $\boldsymbol{A} = [a_1, a_2]$, with $a_1 \leq a_2$. A modal interval $(\boldsymbol{A}, \forall)$ is called *improper interval*, and it is represented by $\boldsymbol{A} = [a_2, a_1]$, with $a_1 \leq a_2$. The set of modal intervals is denoted by $I^*(\mathbb{R})$. As a natural generalization, a k-dimensional modal interval vector is defined by $\boldsymbol{A} = (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k)$, and $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k$ are its 1-dimensional components. The set of modal interval vectors is denoted by $I^*(\mathbb{R}^k)$.

The *equality* and *inclusion* for two modal intervals $\boldsymbol{A} = [a_1, a_2]$ and $\boldsymbol{B} = [b_1, b_2]$ are defined in the following way:

$$\begin{aligned} [a_1, a_2] = [b_1, b_2] \quad &\Leftrightarrow \quad a_1 = b_1 \ \& \ a_2 = b_2, \\ [a_1, a_2] \subseteq [b_1, b_2] \quad &\Leftrightarrow \quad a_1 \geq b_1 \ \& \ a_2 \leq b_2. \end{aligned}$$

This makes the structure $(I^*(\mathbb{R}), \subseteq)$ to be a lattice, which is isomorphic to $((\mathbb{R}, \mathbb{R}), (\geq, \leq))$. The lattice operations *meet* and *join* for a bounded family $\boldsymbol{A}(I) = \{\, \boldsymbol{A}(i) \in I^*(\mathbb{R}) \mid i \in I\}$ of modal intervals (I means an index set) are

$$\bigwedge_{i \in I} \boldsymbol{A}(i) = \boldsymbol{A} \in I^*(\mathbb{R}) \text{ is such that } (\forall i \in I)\ \boldsymbol{X} \subseteq \boldsymbol{A}(i) \Leftrightarrow \boldsymbol{X} \subseteq \boldsymbol{A},$$

$$\bigvee_{i \in I} \boldsymbol{A}(i) = \boldsymbol{B} \in I^*(\mathbb{R}) \text{ is such that } (\forall i \in I)\ \boldsymbol{X} \supseteq \boldsymbol{A}(i) \Leftrightarrow \boldsymbol{X} \supseteq \boldsymbol{B}.$$

The above generalizes $(\boldsymbol{A} \wedge \boldsymbol{B})$ and $(\boldsymbol{A} \vee \boldsymbol{B})$ for the corresponding two-operand case. These operators can be easily obtained by means of operations applied to the bounds of the intervals:

$$\bigwedge_{i \in I} \boldsymbol{A}(i) = \left[\, \max_{i \in I} a_1(i), \min_{i \in I} a_2(i) \,\right], \quad \bigvee_{i \in I} \boldsymbol{A}(i) = \left[\, \min_{i \in I} a_1(i), \max_{i \in I} a_2(i) \,\right].$$

The dualization of $\boldsymbol{A} = [a_1, a_2]$ is a modal interval operator defined as

$$\mathrm{Dual}(A) = [a_2, a_1].$$

Let us be given a continuous function $f : \mathbb{R}^k \to \mathbb{R}$ of the variable $x = (x_1, \ldots, x_k)$, defined over the k-dimensional interval box $\boldsymbol{X} = (\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k)$. Also, we suppose that the variables of f are divided into two non-intersecting subsets. There exist two interval extensions of f, called **-semantic extension* and ***-semantic extension*, that correspond to the subdivision of the variables and their possible modality.

More precisely, if a subdivision $\boldsymbol{X} = (\boldsymbol{X}_p, \boldsymbol{X}_i)$ of the vector $\boldsymbol{X}$ into proper and improper interval components is specified, then we can define

$$\begin{aligned} f^*(\boldsymbol{X}) &= \bigvee_{x_p \in \boldsymbol{X}_p'} \bigwedge_{x_i \in \boldsymbol{X}_i'} \big[f(x_p, x_i), f(x_p, x_i) \big] = \\ &= \Big[\min_{x_p \in \boldsymbol{X}_p'} \max_{x_i \in \boldsymbol{X}_i'} f(x_p, x_i), \max_{x_p \in \boldsymbol{X}_p'} \min_{x_i \in \boldsymbol{X}_i'} f(x_p, x_i) \Big]. \end{aligned}$$

Called the **-semantic extension* of f over $\boldsymbol{X} = (\boldsymbol{X}_p, \boldsymbol{X}_i)$, and

$$\begin{aligned} f^{**}(\boldsymbol{X}) &= \bigwedge_{x_i \in \boldsymbol{X}_i'} \bigvee_{x_p \in \boldsymbol{X}_p'} \big[f(x_p, x_i), f(x_p, x_i) \big] = \\ &= \Big[\max_{x_i \in \boldsymbol{X}_i'} \min_{x_p \in \boldsymbol{X}_p'} f(x_p, x_i), \min_{x_i \in \boldsymbol{X}_i'} \max_{x_p \in \boldsymbol{X}_p'} f(x_p, x_i) \Big], \end{aligned}$$

called the ***-semantic extension* of f over $\boldsymbol{X} = (\boldsymbol{X}_p, \boldsymbol{X}_i)$. They are connected to each other by the equality

$$\mathrm{Dual}(f^*(\boldsymbol{X})) = f^{**}(\mathrm{Dual}(\boldsymbol{X})),$$

and they both form a cornerstone of Modal Interval Analysis because they have a remarkable logical meaning provided by the following "semantic theorems".

***-semantic theorem.** *Given a continuous real function $f : \mathbb{R}^k \to \mathbb{R}$ and a modal vector $\boldsymbol{A} \in I^*(\mathbb{R}^k)$, whenever $F(\boldsymbol{A}) \in I^*(\mathbb{R})$ exists,*

$$(f^*(\boldsymbol{A}) \subseteq F(\boldsymbol{A})) \iff (\forall \boldsymbol{a}_p \in \boldsymbol{A}_p')\ \mathrm{Q}(z, F(\boldsymbol{A}))\ (\exists \boldsymbol{a}_i \in \boldsymbol{A}_i')\ z = f(\boldsymbol{a}_p, \boldsymbol{a}_i).$$

****-semantic theorem.** *Given a continuous real functions $f : \mathbb{R}^k \to \mathbb{R}$ and a modal vector $\boldsymbol{A} \in I^*(\mathbb{R}^k)$, whenever $F(\boldsymbol{A}) \in I^*(\mathbb{R})$ exists,*

$$(f^{**}(\boldsymbol{A}) \supseteq F(\boldsymbol{A})) \iff (\forall \boldsymbol{a}_i \in \boldsymbol{A}_i')\ \mathrm{Q}(z, \mathrm{Dual}(F(\boldsymbol{A})))\ (\exists \boldsymbol{a}_p \in \boldsymbol{A}_p')\ z = f(\boldsymbol{a}_p, \boldsymbol{a}_i),$$

where $\mathrm{Q}(z, F(\boldsymbol{A})') = (\exists z \in F(\boldsymbol{A}))$ if $F(\boldsymbol{A})$ is proper and $\mathrm{Q}(z, F(\boldsymbol{A})) = (\forall z \in F(\boldsymbol{A})')$ if $F(\boldsymbol{A})$ is improper.

For a given k-dimensional interval $\boldsymbol{A}$, computing $f^*(\boldsymbol{A})$ or $f^{**}(\boldsymbol{A})$ is a hard and nontrivial problem, except for several simple cases. When the function f is a rational function of only one or two variables, the computations are quite easy, and the results are the same as those obtained with the Kaucher interval arithmetic [15]. More general, if f is a continuous real function $f : \mathbb{R}^k \to \mathbb{R}$ with syntactic tree where the nodes are the operators, the leaves are the variables, and the branches define the domain of each operator, f can also be operationally extended to the *modal syntactic *-extension* of f, defined by the computational program indicated by the syntactic tree of f when the variables are substituted by their corresponding intervals and their operators are transformed into their *-semantic extensions. Similarly, the *modal syntactic **-extension* is the function fR^{**} defined similarly to fR^{*}, but with the operators transformed into their **-semantic extensions.

The syntactic extensions produce intervals $F(\boldsymbol{A})$ that can satisfy the inclusions $f^*(\boldsymbol{A}) \subseteq F(\boldsymbol{A})$ or $f^{**}(\boldsymbol{A}) \supseteq F(\boldsymbol{A})$ of the semantic theorems. These inclusions can be interpreted as the fact that $F(\boldsymbol{A})$ is either an approximate computation of f^* or of f^{**}. Equivalently, by means of the semantic theorems, these inclusions provide a logical meaning to the interval computation of $F(\boldsymbol{A})$. Further results of Modal Interval Analysis enable us to compute such intervals in various cases depending on the monotonicity of f or the modality of the intervals involved.

When the above theorems and results are not applicable, then the so-called *f^*-algorithm* [12] may prove useful for inner and outer approximations of f^*. There exists a free practical implementation of f^*-algorithm in MATLAB [16].

References

[1] **Ratschan, S.** Applications of quantified constraint solving over the reals — bibliography, arXiv:1205.5571. Cornell Univ. Library, 2012. Available at: https://arxiv.org/abs/1205.5571v1

[2] **Tarski, A.** A decision method for elementary algebra and geometry. 2nd ed. Berkeley: Univ. of California Press, 1951. 63 p.

[3] **Collins, G.E.** Quantifier elimination for real closed fields by cylindrical algebraic decomposition // 2nd GI Conf. “Automata Theory and Formal Languages”, Kaiserslautern, May 20–23. Lecture Notes in Computer Science. Springer, 1975. Vol. 33. P. 134–183.

[4] **Benhamou, F., Goulard, F.** Universally quantified interval constraints // Principles and Practice of Constraint Programming — CP 2000: Proc. of the 6th Intern. Conf. CP 2000, Singapore, Sept. 18–21, 2000. Lecture Notes in Computer Science. Berlin; Heidelberg: Springer, 2000. Vol. 1894. P. 67–82. DOI:10.1007/3-540-45349-0_7.

[5] **Shary, S.P.** A new technique in systems analysis under interval uncertainty and ambiguity // Reliable Computing. 2002. Vol. 8, No. 5. P. 321–418.

[6] **Jaulin, L., Braems, I., Walter, E.** Interval methods for nonlinear identification and robust control // Proc. of the 41st IEEE Conf. on Decision and Control, Las Vegas, Nevada USA, December 2002. P. 4676–4681.

[7] **Ratschan, S.** Efficient solving of quantified inequality constraints over the real numbers // ACM Trans. on Comput. Logic. 2006. Vol. 7, iss. 4. P. 723–748.

[8] **Sharaya, I.A., Shary, S.P.** Tolerable solution set for interval linear systems with constraints on coefficients // Reliable Computing. 2011. Vol. 15, No. 4. P. 345–357.

[9] **Goldsztejn, A., Jaulin, L.** Inner approximation of the range of vector-valued functions // Reliable Computing. 2010. Vol. 14. P. 1–23.

[10] **Herrero, P., Vehí, J., Sainz, M.Á., Jaulin, L.** Quantified Set Inversion algorithm // Reliable Computing. 2005. Vol. 11, No. 5. P. 369–382.

[11] **Kreinovich, V., Nesterov, V.M., Zheludeva, N.A.** Interval methods that are guaranteed to underestimate (and the resulting new justification of Kaucher arithmetic) // Reliable Computing. 1996. Vol. 2, No. 2. P. 119–124.

[12] **Sainz, M.A., Armengol, J., Calm, R., Herrero, P., Jorba, L., Vehi, J.** Modal interval analysis. New Tools for Numerical Information: Lecture Notes in Mathematics, vol. 2091. Cham, Switzerland: Springer, 2014. 316 p.

[13] **Moore, R.E., Kearfott, R.B., Cloud, M.J.** Introduction to interval analysis. Philadelphia: SIAM, 2009. 223 p.

[14] **Hansen, E., Walster, G.W.** Global optimization using interval analysis. New York: Marcel Dekker, 2004.

[15] **Kaucher, E.** Interval analysis in the extended interval space $\mathbb{IR}$ // Computing Suppl. 1980. Vol. 2. P. 33–49.

[16] **Herrero, P., Sainz, M.A.** MIC: Model interval calculator, 2015. Available at: https://sites.google.com/site/modalintervalcalculator/

[17] **Jaulin, L., Walter, E.** Set inversion via interval analysis for nonlinear bounded-error estimation // Automatica. 1993. Vol. 32, No. 8. P. 1053–1064.

[18] **Sharaya, I.A.** `IntLinInc3D` — a software package for visualization of solution sets to interval linear 3D systems. MATLAB based software publicly. Available at: http://www.nsc.ru/interval/sharaya/

[19] **Sharaya, I.A.** Boundary Intervals Method for visualization of polyhedral solution sets // Reliable Computing. 2015. Vol. 19, No. 4. P. 435–467.

[20] **Ratschan, S., Vehí, J.** Robust pole clustering of parametric uncertain systems using interval methods // Robust Control Design 2003 (ROCOND 2003): A Proc. Vol. from the 4th IFAC Symp., Milan, Italy, 25–27 June, 2003 / (Ed.) S. Bittanti and P. Colaneri. Intern. Federation of Automatic Control, 2004. P. 323–328.

[21] **Herrero, P., Georgiou, P., Toumazou, C., Delaunay, B., Jaulin, L.** An efficient implementation of SIVIA algorithm in a high-level numerical programming language // Reliable Computing. 2012. Vol. 16. P. 239–251.

Received 29 May 2016