Is It Possible to Have a Feasible Enclosure-Computing Method Which Is Independent of the Equivalent Form?*

Marcin Michalak
Institute of Computer Science, Silesian University of Technology, ul. Academicka 16, 44-100 Glivice, Poland
marcin.michalak@polsl.pl

Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA
vladik@utep.edu

Abstract

The problem of computing the range $\boldsymbol{y}$ of a given function $f(x_1, \ldots, x_n)$ over given intervals $\boldsymbol{x}_i$ – often called the main problem of interval computations – is, in general, NP-hard. This means that unless P = NP, it is not possible to have a feasible (= polynomial time) algorithm that always computes the desired range. Instead, interval computations algorithms compute an enclosure $\boldsymbol{Y} \supseteq \boldsymbol{y}$ for the desired range. For all known feasible enclosure-computing methods – starting with straightforward interval computations – there exist two expressions $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ for computing the same function that lead to different enclosures. We prove that, unless P = NP, this is inevitable: it is not possible to have a feasible enclosure-computing method which is independent of the equivalent form.

Keywords: interval computations, enclosure, equivalent form, NP-hard
AMS subject classifications: 65G20, 65G40, 03D15, 68Q17

1 Formulation of the Problem

One of the main problems of interval computations. One of the main problems of interval computations has the following form:

- We are given an algorithm $f(x_1, \ldots, x_n)$ for computing a function of n real variables, and n intervals $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$.

*Submitted: June 28, 2011; Revised: August 19, 2012; September 3, 2012; January 24, 2013 Accepted: January 26, 2013.

- We need to compute the range

$$\boldsymbol{y} = [\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$$

  of the function $f(x_1, \ldots, x_n)$ under given intervals.

This problem is NP-hard. It is known (see, e.g., [5]) that the above problem is, in general, NP-hard.

Comment. It is widely believed that P≠NP. In this case, NP-hardness means that it is not possible to have a feasible (= polynomial time) algorithm that always computes the desired range.

Feasible methods for computing enclosures. Since we cannot always efficiently compute the exact range $\mathbf{y}$, and we want to guarantee that the value $f(x_1, \ldots, x_n)$ is contained in the estimated range, we need to find an *enclosure* $\boldsymbol{Y} \supseteq \boldsymbol{y}$. There exist many feasible techniques for computing enclosures: straightforward interval computations, mean value form, methods combined with bisection, etc.; see, e.g., [6].

Existing feasible methods for computing enclosure are not independent of the equivalent form. In straightforward interval computations, we

- represent the algorithm $f(x_1, \ldots, x_n)$ as a sequence of elementary arithmetic operations such as $+$, $-$, $\cdot$, $/$, min, max, etc. and then
- replace each elementary operation with the corresponding operation of interval arithmetic.

In this method, in general, the resulting enclosure may be different for different algorithms that compute the same function. For example, the algorithms $f(x_1) = x_1 - x_1$ and $g(x_1) = 0$ compute the same function 0 on the interval $[0, 1]$, but:

- for $f(x_1) = x_1 - x_1$ straightforward interval computations leads to an enclosure

$$[0, 1] - [0, 1] = [-1, 1],$$

  while
- for $g(x_1) = 0$, we get the enclosure $\mathbf{Y} = [0, 0] \neq [-1, 1]$.

Similarly, all other feasible methods for computing enclosure – at least those which are known to the authors – are not independent of the equivalent form: for each of these methods, there exist two algorithms that compute the same function but lead to different enclosures.

The impact of an equivalent form can be huge. The importance of selecting an appropriate equivalent form can illustrated by the result from [3, 4], according to which, for straightforward interval computations, we can pick any interval $\boldsymbol{Y} = \left[\underline{Y}, \overline{Y}\right]$ that is a superset of the actual range $\boldsymbol{y} = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$, and we can come up with an equivalent form $g(x_1, \ldots, x_n)$ that would give $\boldsymbol{Y}$ as the result.

This result holds when at least one of the intervals is non-degenerate (if all input intervals are degenerate, i.e., consists of a single point, then the range consists of a single value, and straightforward interval computation computes this value.)

This result is reasonably easy to prove. Indeed, let $\left[\,\underline{f},\overline{f}\,\right]$ be the enclosure for $\boldsymbol{y}$ which is obtained when we apply straightforward interval computations to the original expression $f(x_1,\ldots,x_n)$, and let x_i be a variable for which the input interval $[\underline{x}_i,\overline{x}_i]$ is non-degenerate, i.e., for which $\underline{x}_i<\overline{x}_i$. Then, for $\underline{b}\stackrel{\rm def}{=}\max\left(\underline{Y},\underline{f}\right)$ and $\overline{b}\stackrel{\rm def}{=}\min\left(\overline{Y},\overline{f}\,\right)$, the desired equivalent form has the form

$$g(x_1,\ldots,x_n)\stackrel{\rm def}{=}\min(\max(\underline{b},f(x_1,\ldots,x_n)),\overline{b})+$$

$$(\underline{b}-\underline{Y}\,)\cdot\min\left(\frac{x_i-x_i}{\overline{x}_i-\underline{x}_i},0\right)+\left(\overline{Y}-\overline{b}\right)\cdot\max\left(\frac{x_i-x_i}{\overline{x}_i-\underline{x}_i},0\right).$$

Since both $\left[\underline{Y},\overline{Y}\,\right]$ and $\left[\,\underline{f},\overline{f}\,\right]$ are supersets of the range, their intersection $\left[\underline{b},\overline{b}\,\right]$ is also a superset, so $\underline{b}\le f(x_1,\ldots,x_n)\le\overline{b}$. Thus, $\max(\underline{b},f(x_1,\ldots,x_n))=f(x_1,\ldots,x_n)$, hence $\min(\max(\underline{b},f(x_1,\ldots,x_n)),\overline{b})=\min(f(x_1,\ldots,x_n),\overline{b})=f(x_1,\ldots,x_n)$ and therefore, $g(x_1,\ldots,x_n)=f(x_1,\ldots,x_n)$ for all $x_i\in\boldsymbol{x}_i$.

When we apply straightforward interval computations to the expression $g(x_1,\ldots,x_n)$, we:

- first compute the enclosure $\left[\,\underline{f},\overline{f}\,\right]$ for the original expression $f(x_1,\ldots,x_n)$,
- then use the interval analog of max to compute the enclosure for $\max(\underline{b},f(x_1,\ldots,x_n))$ as

$$\max\left(\underline{b},\left[\,\underline{f},\overline{f}\,\right]\right)=\left[\max\left(\underline{b},\underline{f}\right),\max\left(\underline{b},\overline{f}\right)\right]=\left[\underline{b},\overline{f}\,\right];$$

- use the interval rule for min to compute the enclosure for $\min(\max(\underline{b},f(x_1,\ldots,x_n)),\overline{b})$ as $\min\left(\left[\underline{b},\overline{f}\,\right],\overline{b}\right)=\left[\underline{b},\overline{b}\,\right]$;
- the enclosure for $\dfrac{x_i-x_i}{\overline{x}_i-\underline{x}_i}$ is $[-1,1]$, so the enclosure for $\min\left(\dfrac{x_i-x_i}{\overline{x}_i-\underline{x}_i},0\right)$ is $[-1,0]$, and the enclosure $\max\left(\dfrac{x_i-x_i}{\overline{x}_i-\underline{x}_i},0\right)$ is $[0,1]$;
- thus, the final enclosure for $g(x_1,\ldots,x_n)$ is equal to

$$\left[\underline{b},\overline{b}\,\right]+(\underline{b}-\underline{Y}\,)\cdot[-1,0]+\left(\overline{Y}-\overline{b}\right)\cdot[0,1]=\left[\underline{Y},\overline{Y}\,\right].$$

So, indeed, interval computation's unreliability / imprecision is unbounded if we ignore the equivalent form.

Natural question. The above facts lead to the following natural question: Is it possible to have a feasible enclosure-computing method which is independent of the equivalent form?

2 Main Result

Let us provide an answer to the above question.

Definition 1. *By a* numerical algorithm*, we mean a composition of functions* $+$, $-$, $\cdot$, min, max*, and rational constants.*

Examples. Every polynomial with rational coefficients is a (finite) composition of additions, multiplication, and constants, and is, therefore, a numerical algorithm in the sense of Definition 1. For example, a polynomial $f(x_1, x_2) = (x_1 - 1) \cdot (x_2 - 2)$ is a composition of multiplication, two subtractions, and two constants 1 and 2.

In particular, a constant 0 itself is an example of a numerical algorithm in this sense; we will denote this numerical algorithm by f_0. The numerical algorithm $f_0(x_1, \ldots, x_n)$ computes the "zero function", i.e., a function which is identically 0.

Definition 2. *By a* feasible enclosure-computing method*, we mean a feasible algorithm A that, given a numerical algorithm $f(x_1, \ldots, x_n)$ and n bounded intervals $\boldsymbol{x}_1$, $\ldots$, $\boldsymbol{x}_n$ with rational endpoints, computes a bounded interval $A(f, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ with rational endpoints that contains the range*

$$\boldsymbol{y} = \{f(x_1, \ldots, x_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}.$$

Comment. We only consider *bounded* intervals, i.e., intervals for which both endpoints are real numbers (and not infinities). So, by this definition, a method is applicable to *all* numerical algorithms (in the sense of Definition 1) and all tuples of bounded intervals with rational endpoints. For example, straightforward interval computations is an example of a feasible enclosure-computing method – in the sense of this definition.

Proposition. *If $P \neq NP$, then for every feasible enclosure-computing method A, there exist two numerical algorithms $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ and n intervals $\boldsymbol{x}_1$, $\ldots$, $\boldsymbol{x}_n$ for which*

- *$g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)$ for all $x_1 \in \boldsymbol{x}_1$, $\ldots$, $x_n \in \boldsymbol{x}_n$, but*
- *$A(g, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \neq A(h, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.*

Comment. The proof of this proposition is presented after the next paragraph. Thus, it is not possible to have a feasible enclosure-computing method which is independent of the equivalent form.

Interesting open question. In general, when a general problem is proven to be NP-hard, a next natural question is to find important cases in which a feasible algorithm is possible. In our proposition, we prove that (if P $\neq$ NP, then) it is impossible to have a feasible enclosure-computing method which is applicable to *all* possible numerical algorithms (in the sense of Definition 1). So, a natural next question is to find interesting classes of numerical algorithms for which there exists a feasible enclosure-computing method which is independent of equivalent form. Or maybe it will turn out that for every class of numerical algorithms which is large enough (in some reasonable sense), no such feasible method is possible?[1]

[1]The authors are thankful to the anonymous referee for this idea.

Proof.

1°. In our proof, we will use a result proved at the end of the proof of Theorem 3.1 from [5]. This result is about a class of quadratic polynomials $\mathcal{P}$ with rational coefficients. For each polynomial $f(x_1, \ldots, x_n)$ from this class $\mathcal{P}$, its maximum

$$M_f \stackrel{\text{def}}{=} \max\{f(x_1, \ldots, x_n) : x_1 \in [0,1], \ldots, x_n \in [0,1]\}$$

on the box $[0,1] \times \ldots \times [0,1]$ is either less than or equal to 0, or greater than or equal to 1 ($M_f \le 0$ or $M_f \ge 1$). The result from [5] is that the following problem is NP-hard:

- *given:* $f \in \mathcal{P}$,
- *check:* whether $M_f \le 0$ or $M_f \ge 1$.

In our proof, we will consider these same intervals $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n = [0,1]$.

This class of polynomials comes from a modification of the historically first proof [1, 2] that the main problem of interval computations – the problem of computing the range of a given function over give intervals – is NP-hard even for polynomials. This proof was done by reducing a known NP-hard problem – checking whether a given 3-CNF propositional formula is satisfiable – to the main problem of interval computations.

In this known problem, we start with Boolean variables $z_1, \ldots, z_v$, i.e., variables which take two possible values: "true" and "false". A *literal* is defined as a variable z_i or its negation $\neg z_i$, a *clause* is defined as an expression $a \vee b$ or $a \vee b \vee c$, where a, b, and c are literals, and a *3-CNF formula* is defined an expression of the type $F_1 \,\&\, \ldots \,\&\, F_k$, where $F_1, \ldots, F_k$ are clauses. An example of such a formula is $F = (z_1 \vee z_2 \vee z_3) \,\&\, (z_1 \vee \neg z_2)$. A formula is called *satisfiable* if it is true for some Boolean values $z_1, \ldots, z_v$.

To each formula F, we put into correspondence a quadratic polynomial $p[F]$ with $n = v + k$ variables as follows:

- to each Boolean variable z_i, we put into correspondence a real-valued variable

$$p[z_i] = x_i;$$

- to a negation $\neg z_i$, we put into correspondence a linear expression

$$p[\neg z_i] = 1 - x_i;$$

- to each clause $F_j = a \vee b$, we put into correspondence an expression

$$p[F_j] = (p[a] + p[b] + x_{v+j} - 2)^2;$$

- to each clause $F_j = a \vee b \vee c$, we put into correspondence an expression

$$p[F_j] = (p[a] + p[b] + p[b] + 2x_{v+j} - 3)^2;$$

- finally, to the formula $F_1 \,\&\, \ldots \,\&\, F_k$, we put into correspondence an expression

$$p[F] = \sum_{i=1}^{v} x_i \cdot (1 - x_i) + \sum_{j=1}^{k} p[F_j].$$

For example, to the above formula $F = F_1 \,\&\, F_2$, where F_1 is $z_1 \vee z_2 \vee z_3$ and F_2 is $z_1 \vee \neg z_2$, we assign $p[z_1] = x_1$, $p[z_2] = x_2$, $f[z_3] = x_2$, $p[\neg z_2] = 1 - x_2$, $p[F_1] = (x_1 + x_2 + x_3 + 2x_4 - 3)^2$, $p[F_2] = (x_1 + (1 - x_2) + x_5 - 2)^2$, and

$$p[F] = x_1 \cdot (1 - x_1) + x_2 \cdot (1 - x_2) + x_3 \cdot (1 - x_3) + p[F_1] + p[F_2].$$

It can be proven that for every 3-CNF formula F, the smallest value m of the polynomial $p[F]$ on the box $[0,1] \times \ldots \times [0,1]$ satisfies the following two properties:

- if the formula F is satisfiable, then $m \le 0$; and
- if the formula F is not satisfiable, then $m \ge 0.09$.

For the polynomial $f[F] \stackrel{\text{def}}{=} 1 - \dfrac{p[F]}{0.09}$, its maximum $M_f = 1 - \dfrac{m}{0.09}$ on the box $[0,1] \times \ldots \times [0,1]$ has the following properties:

- if the formula F is satisfiable, then $M_f \ge 1$; and
- if the formula F is not satisfiable, then $M_f \le 0$.

By $\mathcal{P}$, we mean the set of all polynomials $f[F]$ obtained from all possible 3-CNF formulas F. Clearly, for each $f \in \mathcal{P}$, we have $M_f \le 0$ or $M_f \ge 1$, and – since checking satisfiability is NP-hard – it is NP-hard to check whether $M_f \le 0$ or $M_f \ge 1$.

2°. We will prove our result by contradiction. Let us assume that there exists a feasible enclosure-computing method A for which, for every two numerical algorithms $g(x_1, \ldots, x_n)$ and $h(x_1, \ldots, x_n)$ that compute the same function on the box

$$[0,1] \times \ldots \times [0,1],$$

we have $A(g, [0,1], \ldots, [0,1]) = A(h, [0,1], \ldots, [0,1])$:

$$(\forall x_1 \ldots \forall x_n \, (x_1 \in [0,1] \ldots x_n \in [0,1] \Rightarrow g(x_1, \ldots, x_n) = h(x_1, \ldots, x_n)) \Rightarrow$$

$$A(g, [0,1], \ldots, [0,1]) = A(h, [0,1], \ldots, [0,1]).$$

Let us show that in this case, we will be able to feasibly check, given a polynomial $f \in \mathcal{P}$, whether $M_f \le 0$ or $M_f \ge 1$.

3°. The method A is applicable to all numerical algorithms, including a numerical algorithm $f_0(x_1, \ldots, x_n)$ that simply returns a constant 0. Let us denote the result $A(f_0, [0,1], \ldots, [0,1])$ of applying the method A to this numerical algorithm f_0 by $[z_-, z_+]$. By Definition 2, both endpoints z^- and z^+ are rational numbers.

Since the method A is enclosure-computing, the range $[z_-, z_+]$ is an enclosure for the actual range $[0,0]$ of the zero function f_0. Thus, $z_- \le 0 \le z_+$.

4°. For a given $f \in \mathcal{P}$, to check whether $M_f \le 0$ or $M_f \ge 1$, we do the following:

- First, we design a new (auxiliary) numerical algorithm

$$g_f(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \max((z_+ + 1) \cdot f(x_1, \ldots, x_n), 0).$$

  Since the original polynomial f is a composition of $+$, $-$, $\cdot$, and rational constants, the new expression g_f is also a composition of $+$, $-$, $\cdot$, min, max, and rational constants, – i.e., it is a indeed a numerical algorithm in the sense of Definition 1.

- Then, we apply the method A to the numerical algorithm g_f and intervals $[0,1],\ldots,[0,1]$. As a result, we get a rational-valued enclosure $A(g_f,[0,1],\ldots,[0,1])$ for the range of the function g_f on the box $[0,1]\times\ldots\times[0,1]$. Let us denote this enclosure by $[\underline{A}_f,\overline{A}_f]$.
- We will then prove – in Part 5 – that the following two implications hold:
  - if $M_f\le 0$, then $\overline{A}_f=z_+$, and
  - if $M_f\ge 1$, then $\overline{A}_f\ge z_++1$.

Because of these two implications, by following above procedure and checking whether $\overline{A}_f=z_+$ or $\overline{A}_f\ge z_++1$, we will be able to check, in feasible (= polynomial) time, whether $M_f\le 0$ or $M_f\ge 1$. However, we know that the problem of detecting whether $M_f\le 0$ or $M_f\ge 1$ is NP-hard. Thus, the fact that we can solve this problem is polynomial time means that P = NP – and we assumed that P $\ne$ NP.

This contradiction proves that our original assumption – that there exists a feasible enclosure-computing method which is independent of the equivalent form – is wrong. The fact that a method is *not* independent of the equivalent form means that for every feasible enclosure-computing method A, there exist two numerical algorithms $g(x_1,\ldots,x_n)$ and $h(x_1,\ldots,x_n)$ with the following properties:

- the numerical algorithms $g(x_1,\ldots,x_n)$ and $h(x_1,\ldots,x_n)$ compute the same function for given n intervals $\boldsymbol{x}_1=\ldots=\boldsymbol{x}_n=[0,1]$, and
- $A(g,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)\ne A(h,\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$.

5°. To complete the proof, we thus need to prove the two above implications:

- that if $M_f\le 0$, then $\overline{A}_f=z_+$, and
- that if $M_f\ge 1$, then $\overline{A}_f\ge z_++1$.

Let us consider them one by one.

5.1°. Let us first consider the case when $M_f\le 0$. Let us prove that in this case, we have $\overline{A}_f=z_+$.

Indeed, since $M_f\le 0$, and M_f is the largest value of the polynomial $f(x_1,\ldots,x_n)$ on the box $[0,1]\times\ldots\times[0,1]$, this means that $f(x_1,\ldots,x_n)\le 0$ for all $x_i\in[0,1]$. Since $z_+\ge 0$, for all values $(x_1,\ldots,x_n)$ from the box $[0,1]\times\ldots\times[0,1]$, we have

$$(z_++1)\cdot f(x_1,\ldots,x_n)\le 0$$

and therefore, $g_f(x_1,\ldots,x_n)=\max((z_++1)\cdot f(x_1,\ldots,x_n),0)=0$. Hence, in this case, the numerical algorithm $g_f(x_1,\ldots,x_n)$ computes the same function as $f_0(x_1,\ldots,x_n)$ for all the values $x_1,\ldots,x_n$ from the box. Since we assumed that the method A is independent of the equivalent form, we can hence conclude that $A(g_f,[0,1],\ldots,[0,1])=A(f_0,[0,1],\ldots,[0,1])=[z_-,z_+]$. Thus, the upper endpoint $\overline{A}_f$ of the interval $A(g_f,[0,1],\ldots,[0,1])$ is indeed equal to z_+. The first implication is proven.

5.2°. Let us first consider the case when $M_f\ge 1$. Let us prove that in this case, we have $\overline{A}_f\ge z_++1$.

Indeed, let $x_1'\in[0,1],\ldots,x_n'\in[0,1]$ be a tuple on which the polynomial $f(x_1,\ldots,x_n)$ attains its maximum M_f on the box $[0,1]\times\ldots\times[0,1]$, i.e., for which $f(x_1',\ldots,x_n')=$

M_f. Since we assumed that $M_f \geq 1$, this means that for these values x'_i, we have $(z_+ + 1) \cdot f(x'_1, \ldots, x'_n) = (z_+ + 1) \cdot M_f \geq z_+ + 1 > 0$, and therefore,

$$g_f(x'_1, \ldots, x'_n) = \max((z_+ + 1) \cdot f(x'_1, \ldots, x'_n), 0) \geq z_+ + 1.$$

By Definition 2, the interval $[\underline{A}_f, \overline{A}_f] = A(g_f, [0,1], \ldots, [0,1])$ is an enclosure for the range of the function $g_f(x_1, \ldots, x_n)$ on the box $[0,1] \times \ldots \times [0,1]$. Thus, for each $x_i \in [0,1]$, the corresponding value $g_f(x_1, \ldots, x_n)$ must be contained in this enclosure. In particular, the enclosure $[\underline{A}_f, \overline{A}_f]$ must contain the value $g_f(x'_1, \ldots, x'_n)$:

$$\underline{A}_f \leq g_f(x'_1, \ldots, x'_n) \leq \overline{A}_f.$$

From $g_f(x'_1, \ldots, x'_n) \geq z_+ + 1$ and $\overline{A}_f \geq g_f(x'_1, \ldots, x'_n)$, we can now conclude that $\overline{A}_f \geq z_+ + 1$. So the second implication is also proven.

5.3°. Both implications are proven and thus, the Proposition is true.

Acknowledgments. The first author was supported by the European Union from the European Social Fund (grant agreement number: UDA-POKL.04.01.01-106/09). This work was also partly supported by the US National Science Foundation grants HRD-0734825 and DUE-0926721, and by Grant 1 T36 GM078000-01 from the US National Institutes of Health.

The authors are thankful to all the participants of the 13th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing RSFDGrC'2011 (Moscow, Russia, June 25–30, 2011) and to the anonymous referees for valuable discussions.

References

[1] Gaganov, A. A.: *Computational complexity of the range of the polynomial in several variables*, Leningrad University, Math. Department, M.S. Thesis, 1981 (in Russian).

[2] Gaganov, A. A.: *Computational complexity of the range of the polynomial in several variables*, Cybernetics, 1985, pp. 418–421.

[3] Koshelev, M.: *Every Superinterval of the Function Range Can Be An Interval-Computations Enclosure*, The Chinese University of Hong Kong, Department of Mechanical & Automation Engineering, Technical Report CUHK-MAE-99-003, January 1999.

[4] Koshelev, M.: *Every superinterval of the function range can be an interval-computations enclosure*, Reliable Computing 6, pp. 219–223, 2000.

[5] Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

[6] Moore, R. E., Kearfott, R. B., Cloud, M. J.: *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylvania, 2009.