Acta Cybernetica **27** (2026) 493–514.

Effective Representation and Fast Computing With Polyarc Bounded Intervals

Gábor Geréb[ab] and András Sándor[cd]

Abstract

Complex interval arithmetic is a powerful tool for the analysis of computational errors. The naturally arising rectangular, polar, and circular interval types yield overly relaxed bounds. The later introduced polygonal type allows for arbitrarily precise representation for a higher computational cost. We propose the polyarc interval type as an effective generalization of the abovementioned types. The polyarc interval can represent all types and most of their arithmetic combinations precisely and has a better approximation capability with that of the polygonal interval. In particular, in specific cases of antenna tolerance analysis and robot localization it can achieve perfect accuracy for lower computational cost then the polygonal type, which we show in a relevant case study.

Keywords: interval arithmetic, computational geometry, antennas, robotics

1 Introduction

Interval analysis is an effective mathematical technique that allows numerical analysis of problems involving sets [23]. It has long been used to put bounds on computational errors and has recently been applied in robotics, robust control, and antenna design. It also proved to be useful for finding all the solutions of nonlinear equations and inequalities [19]. Interval arithmetic studies the properties of the numerical representation and arithmetic operations of intervals, and therefore it is an essential part of interval analysis.

Pioneered by Moore [22], the interval arithmetic for analyzing real-valued computational errors was soon extended to complex numbers in the form of rectangular and polar intervals by Boche [4], and as circular intervals by Gargantini and Henrici [13]. Hansen [16] studied the linear algebra of complex intervals and introduced a generalized interval arithmetic. The field received increased attention in the 1990s: Ohta [24] introduced polygon interval arithmetic, a journal titled Interval

[a]Department of Informatics, University of Oslo, Norway
[b]E-mail: gaborge@uio.no, ORCID: 0000-0002-1972-4784
[c]Budapest University of Technology and Economics, Budapest, Hungary
[d]E-mail: sandor.andras@renyi.hu, ORCID: 0000-0002-8707-9182

DOI: 10.14232/actacyb.314364

Computations (later renamed Reliable Computing) was established [21, 20], the Matlab/Octave software package called INTLAB was published by [27], and the BLAS FORTRAN package received an interval extension [9]. Books on interval analysis and arithmetic were published by Petkovic and Petkovic [26], Jaulin et al. [19], Moore et al. [23] and Dawood [6].

Complex interval arithmetic greatly benefited from its interconnections with geometric algebra, which is widely used in the fields of computer-aided design, image processing, mathematical morphology, geometrical optics, and dynamical stability analysis. The application of the Minkowski algebra to complex intervals opened up the possibility of the representation and arithmetic combination of intervals bounded by arbitrary explicit and implicit curves in the complex plane [12, 10]. Efficient algorithms for calculating the Minkowski sum of polygons [7], borrowed from computational geometry, have been successfully applied in the design of robust control systems [24, 25] and the tolerance analysis of antenna arrays [28]. The polygonal representation typically produced much more accurate results than the original representations, given that the vertex count was high enough (Figure 1). However, a high vertex count came with a high computational cost.

In the tolerance analysis of sensor arrays, the polar intervals defined by independent amplitude and phase intervals are typically the primary operands of evaluation. Motivated by the success of the polygonal representation and its shortcomings in representing polar and circular intervals and their arithmetic combinations, we set out to find a more suitable interval type. Replacing vertices by circular arcs came as a natural extension and led to a new interval type, the polyarc-bounded (polyarcular) interval. By providing perfect representation for a much wider set of intervals at a similar complexity as the polygonal interval, the new interval type suited our application well.

Polyarc is an explicit curve type, defined by an ordered set of circular arcs and consists of the defining arcs and implicit edges between them (Figure 2). It allows the exact representation of the boundary of the (bounded) rectangular, polar, circular, and polygonal intervals. It is closed under addition, negative, reciprocal, union, and intersection operations, and in some cases under multiplication, too.

In this paper we present the previously unpublished polyarc interval type. In Section 2 we summarize the necessary background theory and the existing interval arithmetic methods. We present the polyarc interval type in Section 3, along with the discussion of its properties and a description of our implementation, which is available as an open source code in the form of a Matlab package (see Supplementary materials). To demonstrate the applicability of our method, we show two case-studies in Section 4, one from antenna design and another from robotics, both intuitive yet current questions of academic and industrial interest. Finally, we summarize our findings in Section 5 and discuss future research opportunities.

For the sake of brevity, the typesetting of symbols in equations carry specific meanings, which is summarized in Table 1.

Table 1: Typesetting in mathematical expressions

Typesetting	Meaning	Example
Calligraphic $(\mathcal{I}, \mathcal{R})$	Sets of curves or intervals	$\mathcal{I}(\mathbb{R}) = \{\boldsymbol{t} = [\underline{t}, \overline{t}] \big\vert \underline{t} \le \overline{t}\}$
Bold-italic $(\boldsymbol{a}, \boldsymbol{A})$	Sets	$\boldsymbol{A} = \{A \in \mathbb{C} \big\vert \mathrm{Re}(A) = 1\}$
Italic (a, A)	Numbers	$A \in \boldsymbol{A} = \{t + it \big\vert t \in \boldsymbol{t}\}$
Lowercase $(a, \boldsymbol{a})$	Real numbers and sets	$a \in \boldsymbol{a} \subset \mathbb{R}$
Uppercase $(A, \boldsymbol{A})$	Complex numbers and sets	$A \in \boldsymbol{A} \subset \mathbb{C}$
Sans-sherif (n, N)	Natural numbers and sets	$\mathsf{n} \in \{1..\mathsf{N}\} \subset \mathbb{N}$
Under-, overlined $(\underline{t}, \overline{t})$	Infimum and supremum	$[\underline{t}, \overline{t}] \ni t,\ \underline{t} \le t \le \overline{t}$

2 Background

2.1 Complex intervals

Complex intervals are certain subsets of the complex plane $\mathbb{C}$, which we will consider our universe. Since there is no common definition for complex intervals (some prefer the term: complex sets), for this discussion we assume that all of them are connected and bounded sets. We also assume that they each have a piece-wise smooth, simple, closed (Jordan) boundary curve. The set of Jordan curves is $\mathcal{J}(\mathbb{C})$.

Definition. *A complex set belongs to the set $\mathcal{I}(\mathbb{C})$ of complex intervals if it is bounded by a piecewise smooth Jordan curve:*

$$\boldsymbol{A} \in \mathcal{I}(\mathbb{C}) \iff \partial \boldsymbol{A} \in \mathcal{J}(\mathbb{C}).$$

The three most common complex interval types have emerged as an extension of real intervals $\mathcal{I}(\mathbb{R})$ to the complex plane. These are the rectangular, polar [4] and circular [13] intervals. Illustrative examples can be found in Figure 1.

Definition. *A rectangular interval*

$$\boldsymbol{a} + \boldsymbol{b}i = \{Z \in \mathbb{C} \mid \mathrm{Re}(Z) \in \boldsymbol{a}, \mathrm{Im}(Z) \in \boldsymbol{b}\} \in \mathcal{R}(\mathbb{C})$$

(with $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{I}(\mathbb{R})$) is the Cartesian product of two real intervals.

Definition. *A polar interval is defined by the polar product of two real intervals, that is*

$$\boldsymbol{r}e^{i\boldsymbol{\varphi}} = \{Z \in \mathbb{C} \mid |Z| \in \boldsymbol{r}, \angle Z \in \boldsymbol{\varphi}\} \in \mathcal{P}(\mathbb{C})$$

where $\boldsymbol{r} \in \mathcal{I}(\mathbb{R})$ is the radial and $\boldsymbol{\varphi} \in \mathcal{I}(\mathbb{R})$ is the angular interval, and the center is the origin.

Definition. *A circular interval is a closed disk in the complex plane, that is*

$$O + [0, r] \cdot e^{i[-\pi, \pi]} = \{Z \in \mathbb{C} \mid |Z - O| \le r\} \in \mathcal{C}(\mathbb{C})$$

with center $O \in \mathbb{C}$ and radius $r \in \mathbb{R}$.

Polygon interval arithmetic was introduced in [24, 25] to represent uncertainty in robust control systems. Since a polygon can approximate any simple closed curve with a finite dataset, it can also represent any complex interval with arbitrary precision limited only by computational constraints. The set of polygonal curves is $\bar{\mathcal{J}}(\mathbb{C})$.

Definition. *Polygonal intervals are complex intervals bounded by polygonal curves.*

$$\boldsymbol{A} \in \mathcal{G}(\mathbb{C}) \subset \mathcal{I}(\mathbb{C}) \iff \partial \boldsymbol{A} \in \bar{\mathcal{J}}(\mathbb{C})$$

Each interval type above has a finite data representation and specific algorithms for arithmetic and set operations. An interval can be represented by different types, and the representation can be cast from one type to the other. We consider an interval to be a member of a given type if it can be exactly represented by it. If an interval is not a member of the corresponding type, then we assume the smallest inclusive interval of the type, in which case the representation will not be tight.

Definition. *The tightness of a representation* $\boldsymbol{A}^{\mathcal{X}} \in \mathcal{X}(\mathbb{C})$ *of the complex interval* $\boldsymbol{A} \in \mathcal{I}(\mathbb{C})$ *is the ratio of its original size and its represented size.*

$$\tau(\boldsymbol{A}^{\mathcal{X}}) = \frac{\mu(\boldsymbol{A})}{\mu(\boldsymbol{A}^{\mathcal{X}})} \in [0, 1],$$

where $\mu(\cdot)$ *is the Lebesgue measure on* $\mathbb{C}$.

2.2 Interval arithmetic

Arithmetic operations on intervals are typically defined by the Minkowski algebra, which is a collection of pointwise operations on sets.

Definition. *For* $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{I}(\mathbb{C})$

$$\begin{aligned}
\boldsymbol{A} \oplus \boldsymbol{B} &= \{A + B \,|\, A \in \boldsymbol{A}, B \in \boldsymbol{B}\}, \\
\boldsymbol{A} \otimes \boldsymbol{B} &= \{A \times B \,|\, A \in \boldsymbol{A}, B \in \boldsymbol{B}\}, \\
-\boldsymbol{A} &= \{-A \,|\, A \in \boldsymbol{A}\}, \\
\boldsymbol{A}^{-1} &= \{A^{-1} \,|\, A \in \boldsymbol{A}\}, \\
\boldsymbol{A} \ominus \boldsymbol{B} &= \{A + (-B) \,|\, A \in \boldsymbol{A}, B \in \boldsymbol{B}\} = \boldsymbol{A} \oplus (-\boldsymbol{B}), \\
\boldsymbol{A} \oslash \boldsymbol{B} &= \{A \times B^{-1} \,|\, A \in \boldsymbol{A}, B \in \boldsymbol{B}\} = \boldsymbol{A} \otimes (\boldsymbol{B}^{-1}).
\end{aligned}$$

Minkowski addition ($\oplus$) and multiplication ($\otimes$) are both closed binary operations. The element-wise negation is a closed unary operation. This means that the results of the sum, product and negative of complex intervals are also complex intervals. However, the reciprocal is only a partial unary operation because the complex zero has no inverse on the complex plane and hence intervals including zero have unbounded inverses. The element-wise subtraction ($\ominus$) and division ($\oslash$) can be defined as the combination of the operations mentioned above and consequently they are closed and partial binary operations respectively. Similarly to real intervals, the Minkowski addition and multiplication of complex intervals are commutative and associative but not distributive [15, 26].

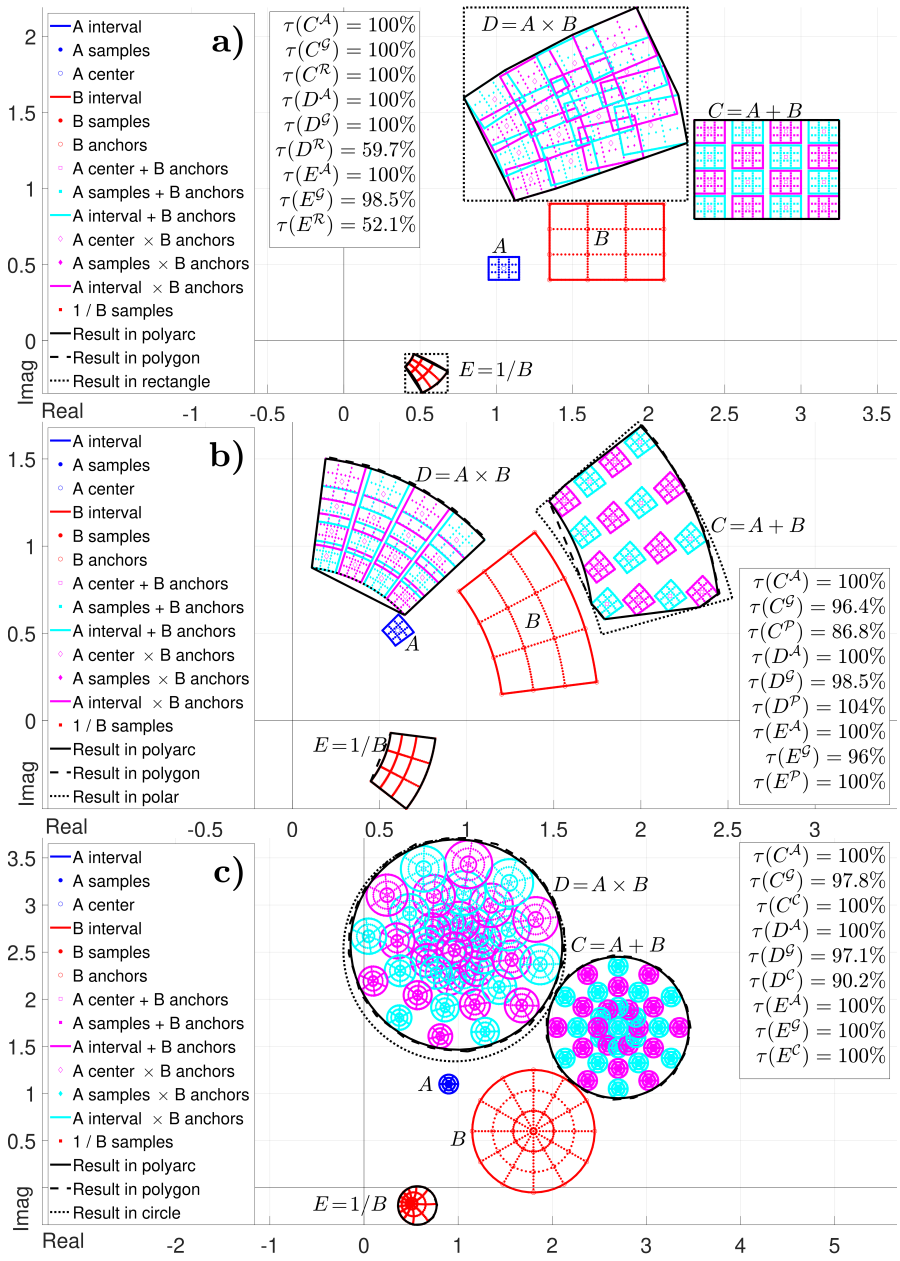


Figure 1: Sum, product, and reciprocal of **a)** rectangular ($\mathcal{R}$), **b)** polar ($\mathcal{P}$) and **c)** circular ($\mathcal{C}$) intervals. Each interval type is casted to polygonal ($\mathcal{G}$) and polyarc ($\mathcal{A}$) types and the tightness ($\tau(\cdot)$) of the arithmetic results are compared.

Table 2: Arithmetic and set properties of complex interval types. For example the polar type is not closed under addition, so $\boldsymbol{A}, \boldsymbol{B} \in \mathcal{P}(\mathbb{C}) \implies \boldsymbol{A}+\boldsymbol{B} \in \mathcal{A}(\mathbb{C})$. Blue symbols indicate when the operation points outside the operand type, but can be represented by another type. In this case we indicate the most specific type that includes the result considering that $\mathcal{R} \subset \mathcal{G}$ and $\mathcal{R}, \mathcal{P}, \mathcal{C}, \mathcal{G} \subset \mathcal{A}$. Red symbols indicate when none of the types can represent the result. (* If one operand is a polar interval, the result is polyarcular, which has a relevance in our first case study in Section 4. † The intersection operation assumes that the operands are connected, while the union operation assumes that the result is connected.) For formal proof of these properties, see Supplementary materials.

$\boldsymbol{A}, \boldsymbol{B}$	$\mathcal{R}(\mathbb{C})$	$\mathcal{P}(\mathbb{C})$	$\mathcal{C}(\mathbb{C})$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$
$\boldsymbol{A}+\boldsymbol{B}$	$\mathcal{R}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$	$\mathcal{C}(\mathbb{C})$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$
$\boldsymbol{A} \times \boldsymbol{B}$	$\mathcal{I}(\mathbb{C})$	$\mathcal{P}(\mathbb{C})$	$\mathcal{I}(\mathbb{C})$	$\mathcal{I}(\mathbb{C})$	$\mathcal{I}(\mathbb{C})^*$
$-\boldsymbol{A}$	$\mathcal{R}(\mathbb{C})$	$\mathcal{P}(\mathbb{C})$	$\mathcal{C}(\mathbb{C})$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$
$\boldsymbol{A}^{-1}$	$\mathcal{A}(\mathbb{C})$	$\mathcal{P}(\mathbb{C})$	$\mathcal{C}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$
$\boldsymbol{A} \cap \boldsymbol{B}^\dagger$	$\mathcal{R}(\mathbb{C})$	$\mathcal{P}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$
$\boldsymbol{A} \cup \boldsymbol{B}^\dagger$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$	$\mathcal{G}(\mathbb{C})$	$\mathcal{A}(\mathbb{C})$

Theorem. *For* $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathcal{I}(\mathbb{C})$

$$\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{B} \oplus \boldsymbol{A}, \qquad \boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{B} \otimes \boldsymbol{A},$$
$$(\boldsymbol{A} \oplus \boldsymbol{B}) \oplus \boldsymbol{C} = \boldsymbol{A} \oplus (\boldsymbol{B} \oplus \boldsymbol{C}), \qquad (\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C}),$$
$$\boldsymbol{A} \otimes (\boldsymbol{B} \oplus \boldsymbol{C}) \subseteq (\boldsymbol{A} \otimes \boldsymbol{B}) \oplus (\boldsymbol{A} \otimes \boldsymbol{C}).$$

Also, there is no additive and multiplicative inverse element for non-degenerate complex intervals, and therefore no inverse operations either. (Intervals consisting of a single non-zero complex number have inverses.) We only have the following trivial inclusions.

$$\begin{aligned} \boldsymbol{A} \oplus (-\boldsymbol{A}) &\ni 0, \quad & (\boldsymbol{A} \oplus \boldsymbol{B}) \ominus \boldsymbol{B} &\supseteq \boldsymbol{A}, \\ \boldsymbol{A} \otimes \boldsymbol{A}^{-1} &\ni 1, \quad & (\boldsymbol{A} \otimes \boldsymbol{B}) \oslash \boldsymbol{B} &\supseteq \boldsymbol{A}. \end{aligned} \tag{1}$$

Interval types are not closed under all operations. In other words, performing an operation on intervals of a given type may result in an interval that cannot be exactly represented with that type. Table 2 summarizes the arithmetic properties of our complex interval types. These properties can be formally proven using geometric algebra, in fact, they are direct consequences of the well-known arithmetic properties of lines and circles [12]. The rigorous proof of the properties exceeds the scope of the present paper, but a review paper is under preparation, where it will be presented. In practice, the interval arithmetic algorithms determine the smallest inclusive interval around the result [4, 13, 26].

3 Polyarc interval

3.1 Concept

The boundaries of rectangular, polar, and circular intervals are all piecewise smooth Jordan curves consisting of edges and arcs. Similarly, polygonal intervals are bounded by edges by definition.

Definition. *An edge* $\mathbf{\Gamma} \in \bar{\mathcal{O}}(\mathbb{C})$ *between the points* $P_1, P_2 \in \mathbb{C}$ *can be given by the* $[0,1] \to \mathbb{C}$ *parametrization*

$$\bar{\Gamma}(P_1, P_2)(t) = (1-t)P_1 + tP_2.$$

Definition. *An arc* $\mathbf{\Gamma} \in \mathring{\mathcal{O}}(\mathbb{C})$ *centered at* $O \in \mathbb{C}$*, with radius* $r \in \mathbb{R}$ *and angular interval* $\boldsymbol{\varphi} \in \mathcal{I}(\mathbb{R})$ *is given by the* $[0,1] \to \mathbb{C}$ *parametrization*

$$\mathring{\Gamma}(O, r, \boldsymbol{\varphi})(t) = O + r \exp\left((1-t)\, i\, \underline{\varphi} + t\, i\, \overline{\varphi}\right).$$

A curve consisting of edges and arcs can precisely represent all the intervals of the mentioned types. The polyarc curve is a direct extension of the polygonal curve, where we replace the vertices with arcs keeping the implicit edges between them (Figure 2). In turn, we propose the corresponding new interval type that contains all the complex interval types above and has the same or better approximation capability for arbitrary complex intervals as the polygonal intervals.

Definition. *A polyarc curve (or polyarc)* $\mathbf{\Gamma} \in \mathring{\mathcal{J}}(\mathbb{C})$ *can be given by the* $[0, 2\mathsf{N}] \to \mathbb{C}$ *parametrization*

$$\Gamma(t) = \begin{cases} \mathring{\Gamma}_{\mathsf{n}}(t - 2\mathsf{n} + 2) & \text{for } t \in [2\mathsf{n}-2, 2\mathsf{n}-1], \\ \bar{\Gamma}_{\mathsf{n}}(t - 2\mathsf{n} + 1) & \text{for } t \in [2\mathsf{n}-1, 2\mathsf{n}], \end{cases}$$

for $\mathsf{n} \in \{1..\mathsf{N}\}$*, with alternating arc and edge pieces (either can be a single point if necessary). Precisely*

$$\begin{aligned} \mathring{\Gamma}_{\mathsf{n}}(t) &= \mathring{\Gamma}(O_{\mathsf{n}}, r_{\mathsf{n}}, \boldsymbol{\varphi}_{\mathsf{n}})(t), \\ \bar{\Gamma}_{\mathsf{n}}(t) &= \bar{\Gamma}(P_{2\mathsf{n}-1}, P_{2\mathsf{n}})(t), \end{aligned}$$

satisfying

$$P_{2\mathsf{n}-1} = O_{\mathsf{n}} + r_{\mathsf{n}} e^{i\overline{\varphi}_{\mathsf{n}}}, \quad P_{2\mathsf{n}} = O_{\mathsf{n}+1} + r_{\mathsf{n}+1} e^{i\underline{\varphi}_{\mathsf{n}+1}},$$

with $O_{\mathsf{N}+1} = O_1$, $r_{\mathsf{N}+1} = r_1$, $\boldsymbol{\varphi}_{\mathsf{N}+1} = \boldsymbol{\varphi}_1$, $P_{2\mathsf{N}} = P_0$.

The set of polyarc curves is $\mathring{\mathcal{J}}(\mathbb{C}) \subset \mathcal{J}(\mathbb{C})$.

Definition. *The polyarc intervals* $\mathcal{A}(\mathbb{C})$ *are complex intervals bounded by polyarcs.*

$$\boldsymbol{A} \in \mathcal{A}(\mathbb{C}) \iff \partial \boldsymbol{A} \in \mathring{\mathcal{J}}(\mathbb{C})$$

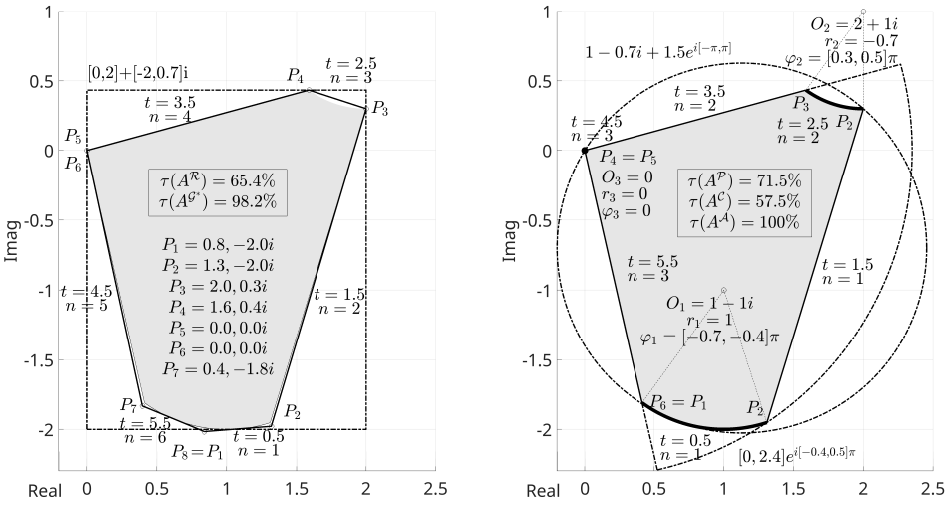


Figure 2: Example of a complex interval represented by various complex interval types. The gray area identifies the given complex interval selected to be perfectly representable by the polyarc type; solid black lines identify the convex polygonal ($\mathcal{G}^*$) and polyarc ($\mathcal{A}$) boundary curves. The grey dash-dotted lines indicate the boundaries of inclusive rectangular ($\mathcal{R}$), polar ($\mathcal{P}$) and circular ($\mathcal{C}$) interval type objects. Representation tightness $\tau(\cdot)$ is listed in text-boxes.

3.2 Properties

The properties of the polyarc interval type are directly related to that of the boundary curve, which consists of arcs and edges. Since an interval is uniquely defined by its boundary curve, arithmetic and set operations can be performed by determining the result boundary. Moreover, the result boundary can be directly calculated from the operand boundaries using the Minkowski algebra. Set operations can be performed by selecting the appropriate subset of the merged operand boundary segments forming the outer or inner boundary of the joint set, so it is trivial that the polyarc intervals will be closed under these operations. The arithmetic properties of arcs and edges can be derived from the properties of lines and circles, which has been thoroughly analyzed by Farouki et al. in [12], starting with Theorem 2.2.

Even without a rigorous analysis, which would exceed the scope of the present paper, we can assume the following properties. Since lines and circles are closed under negation, arcs and edges will be also closed under it, and thus the negative of a polyarc interval also belongs to this type. The reciprocal operation turns non-zero-crossing lines into zero-crossing circles and vice versa, therefore arcs and edges are not closed under it, but since the result boundary will consist of arcs and edges, it will also be of the polyarc type – with the obvious caveat of the complex zero we

have discussed earlier (see Figure 1).

When it comes to binary operations, the key question is whether the envelope of the curve sum or product contains anything else than arcs and edges [11]. The sum of lines has either no envelope or a line envelope when they are parallel. The sum of a line and a circle has an envelope of two lines. Last, the sum of circles has an envelope of one or two circles. We can therefore conclude that polyarc intervals are closed under the addition operation. However, the product of lines and circles may involve envelopes of parabola, hyperbole, ellipse and Cartesian oval curves, except when at least one operand is a zero-crossing line or a zero-centered circle. Therefore, polyarc intervals are not closed under multiplication. Table 2 summarizes the arithmetic and set properties of polyarc intervals. A more detailed description of these properties including their formal proof is available in the Supplementary materials.

3.3 Implementation

The polyarc interval arithmetic has been implemented using the following data types. A set of arithmetic and set operations including addition, multiplication, negative, reciprocal, union and intersection have been implemented for each data type. We used double precision floating point type (double) to represent real numbers, and two real numbers to represent complex numbers.

In our implementation real intervals are represented by storing their endpoints (2 doubles); edges are represented by storing their complex endpoints (4 doubles); arcs are represented by storing their complex type center, real type radius and real interval type argument (5 doubles). Polygonal intervals are represented by storing the ordered set of complex type vertices of the smallest bounding polygon of a given vertex count N ($2\mathsf{N}$ doubles). Finally, polyarc intervals are represented by storing the ordered set of defining arcs (see Remark 3.3) of the smallest bounding polyarc of a given arc count N ($5\mathsf{N}$ doubles).

Remark. polyarc curves consist of arcs defined by their data set and the implicit edges connecting the end-points of adjacent arcs. It is possible to suppress circular segments by setting $r_{\mathsf{n}} = 0$, while the linear segment can be suppressed by making sure that the endpoints are equal. Concave arcs can be created using negative radius values.

A polyarc interval can be efficiently stored by its defining arcs only (some of which might have zero radius[1]). However, for the complete representation of the boundary curve – which will be necessary for the arithmetic operations – the implicit edges and vertices have to be extracted (see Definition 3.1).

Other interval types (rectangular, polar, circular and polygonal) can be casted to the polyarc type using Algorithm 1. Figure 2 gives a demonstrative example of a complex interval represented by the polyarc interval type.

[1] We allow the radius of an arc to be zero to allow the representation of vertices with arc-type objects, which simplifies the implementation of the algorithms.

Algorithm 1 Casting an interval to the polyarc type

1: Decompose the interval boundary to an ordered set of edges and arcs.
2: Convert the vertices at the intersection of edges to zero-radius arcs[1].
3: Store the arcs (including the converted vertices) in a counter-clockwise order.

Remark. Roughly speaking, edges ensure the continuity of the curve, whereas vertices ensure the continuity of the direction of the curve. Since the derivative jumps at the vertices, its value there should be represented by a real interval. Therefore, we consider a vertex as a combination of a point and an angular interval, which is the same as a zero-radius arc. We can then consider vertices as implicit arcs and represent them as arc type variables. Zero-length edges, which occur when consecutive arcs meet at the same point, and redundant vertices, which can occur when a defining arc has zero radius, are removed from the curve.

Let us see how did we implement operations of polyarcs.

Negative is the simplest unary operation, since edges are closed under it and the curve derivative does not have to be considered. It is sufficient to negate the defining arcs to get the result curve. The negative of an arc is

$$-\mathring{\mathbf{\Gamma}} = -\mathring{\mathcal{O}}(\mathbb{C}\big|O, r, \boldsymbol{\varphi}) = \mathring{\mathcal{O}}(\mathbb{C}\big| -O, r, \boldsymbol{\varphi} + \pi). \tag{2}$$

Reciprocal is slightly more involved, because the reciprocal of a non-zero-crossing edge is an arc, and the reciprocal of an arc that is on a zero touching circle is an edge. The result boundary curve can be calculated following Algorithm 2.

Algorithm 2 Determining the reciprocal of a polyarc interval

1: Exclude the defining arcs with zero radius.
2: Add implicit edges to the list according to the segment order in the curve.
3: Calculate the reciprocal of each segment.
4: Store the arc type result segments and the vertices at the intersection of edges.

The reciprocal of an arc is

$$1/\mathring{\mathbf{\Gamma}} = 1/\mathring{\mathcal{O}}(O, r, \boldsymbol{\varphi}) = \begin{cases} \bar{\mathcal{O}}(\mathbb{C}\big|1/\mathring{\Gamma}(0), 1/\mathring{\Gamma}(1)) \\ \mathring{\mathcal{O}}(\mathbb{C}\big|O^*, r^*, [\angle(1/\mathring{\Gamma}(0) - O^*), \angle(1/\mathring{\Gamma}(1) - O^*)]), \end{cases} \tag{3}$$

where $O^* = \mathring{\nu}/(1 - (r|\mathring{\nu}|)^2)$ and $r^* = -r|\mathring{\nu}|^2/(1 - (r|\mathring{\nu}|)^2)$ are the center and radius of the reciprocal arc respectively, and $\mathring{\nu} = 1/O$ is the normalization coefficient that scale-rotates the operand arc on a real one centered circle. The reciprocal of an edge is

$$1/\bar{\mathbf{\Gamma}} = 1/\bar{\mathcal{O}}(\mathbb{C}\big|P_1, P_2) = \begin{cases} \bar{\mathcal{O}}(\mathbb{C}\big|1/P_1, 1/P_2) & \text{if zero-crossing,} \\ \mathring{\mathcal{O}}(\bar{\nu}/2, |\bar{\nu}|/2, [\angle(1/P_1 - \bar{\nu}/2), \angle(1/P_2 - \bar{\nu}/2)] & \text{else,} \end{cases} \tag{4}$$

where $a = \mathrm{Im}(P_1 - P_2)/\mathrm{Re}(P_1 - P_2)$ is the slope of the operand edge and $\bar{\nu} = \exp(-i \arctan(a + \pi/2)/(P_1 \cos(\arctan(a + \pi/2)))$ is the normalization coefficient that scale-rotates the operand edge on a real one crossing vertical line.

Trimming is a special unary operation that turns a self-intersecting curve into a Jordan one by extracting the inner or outer boundary. It is an indispensable part of the implementation of the binary operations: addition, multiplication, union and intersection. For our purposes, we implemented a simplified trimming method shown in Algorithm 3. The method is based on the idea of following the boundary curve in the counter-clockwise direction and turning right at each intersection if the outer boundary is required; and turning left if the inner boundary is required. This method cannot identify holes in the interval, which can occur for example when two polar intervals with wide angular intervals are multiplied. The resulted annulus is turned into a circle by the simplified trimming algorithm. For a more sophisticated algorithm see [11].

Algorithm 3 Trimming a self-intersecting polyarc curve

1: Split the boundary curve segments where they intersect each other.
2: Select and store the curve segment containing the smallest real valued point.
3: Find segments with start-points equal to the end-point of the selected one.
4: Select and store the segment with the lowest or highest starting Gauss map value depending whether the outer or inner boundary is to be extracted.
5: Repeat step 3 and 4 until the end-point of the selected segment equals the start-point of an already stored segment.
6: If the inner boundary is extracted, remove the stored segments until the one that is connected to the last one.
7: Keep the stored arcs and the vertices at the intersection of stored edges as defining arcs for the result.

Union can be directly implemented using the trimming method by choosing the outer boundary option. The result is only valid if the operand intervals are connected.

Intersection is implemented by extracting the inner boundary with the trimming method.

Addition is the simpler of the two binary operations, because the result boundary consists solely of arcs and edges. It requires the extraction of the implicit edges and vertices (arcs). The result boundary segments are a subset of the pair-wise sum of the operand boundary segments including the edges and vertices. We identify this subset using the Gauss map matching and trimming operations.

Gauss map matching identifies those pairs of operand boundary segments that contribute to the result boundary. It is based on the observation that when two curve segments (edge or arc) are added, the normal angle interval of the result is the intersection of that of the operands. The Gauss map of a curve can be seen as

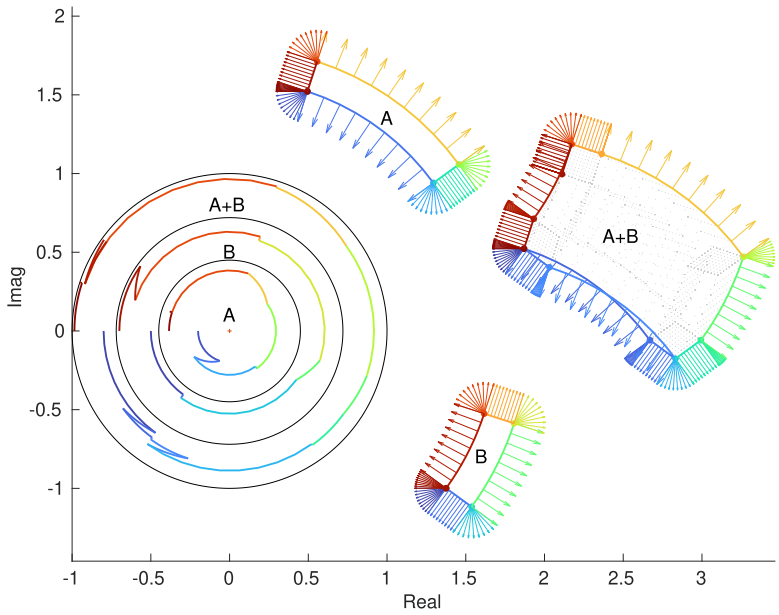


Figure 3: Two polar intervals, $\boldsymbol{A}$ and $\boldsymbol{B}$, and their polyarc sum before trimming. The color of the arcs and the normal vectors indicate the Gauss map value (e.g. $\gamma(\Gamma_A, t)$). The Gauss maps are presented as three concentric curves in the unit circle (with the result outside), where the angle and color indicates the Gauss map value γ, and the radius indicates the curve parameter t. (The radii has been offset to form the concentric rings on the left.)

its normal angle as a function of the curve parameter. A more detailed discussion about Gauss map matching is available in the Supplementary materials.

Definition. *The Gauss map interval of a regular curve segment* $\Gamma(\boldsymbol{t})$ *is*

$$\boldsymbol{\gamma}_{\Gamma,\boldsymbol{t}} = \gamma(\Gamma, \boldsymbol{t}) = \angle i\Gamma'(\boldsymbol{t}).$$

Proposition. *For two boundary segments* $\boldsymbol{\Gamma}_{A,\mathsf{n}} \subset \partial \boldsymbol{A}$ *and* $\boldsymbol{\Gamma}_{B,\mathsf{k}} \subset \partial \boldsymbol{B}$

$$\boldsymbol{\Gamma}_{A,\mathsf{n}} \oplus \boldsymbol{\Gamma}_{B,\mathsf{k}} \subset \partial(\boldsymbol{A} \oplus \boldsymbol{B}) \implies \boldsymbol{\gamma}_{\Gamma_{A,\mathsf{n}}} \cap \boldsymbol{\gamma}_{\Gamma_{B,\mathsf{k}}} \neq \emptyset.$$

Since the Gauss-map value is constant along edges, the sum of two edges $\bar{\boldsymbol{\Gamma}}_A = \bar{\mathcal{O}}(\mathbb{C}\big|P_{A,1}, P_{A,2})$ and $\bar{\boldsymbol{\Gamma}}_B = \bar{\mathcal{O}}(\mathbb{C}\big|P_{B,1}, P_{B,2})$ is only part of the result boundary if they are parallel and have the same direction, in which case the result edge is

$$\bar{\boldsymbol{\Gamma}}_A + \bar{\boldsymbol{\Gamma}}_B \underset{\gamma_{\Gamma_A}=\gamma_{\Gamma_B}}{=} \bar{\mathcal{O}}(\mathbb{C}\big|P_{A,1} + P_{B,1}, P_{A,2} + P_{B,2}). \tag{5}$$

The Gauss-map value of an arc (or vertex) $\mathring{\mathbf{\Gamma}}_B = \mathring{\mathcal{O}}(\mathbb{C}\big|O_B, r_B, \boldsymbol{\varphi}_B)$ is always changing monotonously along the segment, therefore it is only a single point on the arc that matches the Gauss map of the operand edge, in which case the result segment is the operand edge translated by this point on the arc.

$$\bar{\mathbf{\Gamma}}_A + \mathring{\mathbf{\Gamma}}_B \underset{\gamma_{\Gamma_A} \in \gamma_{\Gamma_B}}{=} \bar{\mathcal{O}}(\mathbb{C}\big|P_{A,1}+O_B+r_B e^{i\gamma_{\Gamma_B}}, P_{A,2}+O_B+r_B e^{i\gamma_{\Gamma_B}}) \tag{6}$$

If we consider the Gauss map of an arc (or vertex) as an interval, then the matching of two arcs results the intersection of these intervals, which in some cases can consist of two disconnected intervals due to the angular wrapping (e.g. $[0.8, 2.2]\pi \cap [-0.2, 1.2]\pi$). The Gauss-map matched sum of two arcs $\mathring{\mathbf{\Gamma}}_A$ and $\mathring{\mathbf{\Gamma}}_B$ then results zero, one or two arcs that is

$$\mathring{\mathbf{\Gamma}}_A + \mathring{\mathbf{\Gamma}}_B \underset{\gamma_{\Gamma_A} \cap \gamma_{\Gamma_B} \neq \emptyset}{=} \mathring{\mathcal{O}}(\mathbb{C}\big|O_A + O_B, r_A + r_B, \boldsymbol{\varphi}_A \cap \boldsymbol{\varphi}_B) \tag{7}$$

Figure 3 shows the addition of two polyarc intervals.

A simplified addition algorithm has been implemented for convex operands. A convexity test can be implemented by checking if any of the arcs have negative radius and if any of the vertices have higher Gauss map value at the preceding segment than the following one. The algorithm can then follow the *MinkowskiSum* algorithm in [7, Ch. 13] with the only modification that it uses the Gauss map values instead of the edge angles and instead of adding vertices it adds arcs as in (7). Since the sum of convex intervals has a simple boundary, the trimming operation is unnecessary and the Gauss-map matching is simplified from the quadratic time complexity to linear. Also, because the edges resulting from adding an operand edge to a curve segment of the other operand implicitly results from adding the operand vertices to the other operand segments, the extraction of the implicit edges is unnecessary. In our code implementation, we created a special interval type optimized for fast addition of convex polyarc intervals. The type, called *polyarx interval*, uses a different storage method to avoid the extraction step of the implicit vertices.

Multiplication is similar to the addition, because the product can be seen as addition in logarithmic domain. However, while the envelope [5] of the set formed by the addition of arcs and edges is always bounded by arcs or edges, this is not true for the set formed by multiplying arcs and edges [12]. Therefore, the boundary of two intervals' product can contain segments that are neither arcs nor edges, where the arc-edge product set touches the envelope. This interval cannot be exactly represented by a polyarc curve, but needs to be approximated. To find the subset of pair-wise product of operand boundary segments that are on the result boundary, we used the log-Gauss map matching and trimming algorithms. The log-Gauss matching is based on the fact that the log-Gauss map interval of a given segment in the product interval boundary is the intersection of the log-Gauss map intervals of the corresponding operand curve segments. Similar to the addition algorithm, this allows to extract the boundary curve segments by eliminating operand pairs with

non-overlapping log-Gauss map intervals. Our algorithm also contains the analytical functions for determining whether a given curve segment touches the envelope, in which case it will be apporoximated by an arc. Figure 4 shows an example of the product of two polyarc intervals in both the cartesian and logarithmic domains, and the log-Gauss map of the boundary curve of both operands and the result.

Definition. *The log-Gauss map interval of a regular curve segment* $F(\boldsymbol{t})$ *is*

$$\mathring{\boldsymbol{\gamma}}_{F,\boldsymbol{t}} = \mathring{\gamma}(F,\boldsymbol{t}) = \gamma(\log F,\boldsymbol{t}) = \angle\frac{F'(\boldsymbol{t})}{F(\boldsymbol{t})} = \boldsymbol{\gamma_{F,t}} - \angle F(\boldsymbol{t}).$$

Proposition. *For two boundary segments* $\boldsymbol{\Gamma}_{A,\mathsf{n}} \subset \partial\boldsymbol{A}$ *and* $\boldsymbol{\Gamma}_{B,\mathsf{k}} \subset \partial\boldsymbol{B}$

$$\boldsymbol{\Gamma}_{A,\mathsf{n}} \otimes \boldsymbol{\Gamma}_{B,\mathsf{k}} \subset \partial(\boldsymbol{A} \otimes \boldsymbol{B}) \implies \mathring{\boldsymbol{\gamma}}_{\Gamma_{A,\mathsf{n}}} \cap \mathring{\boldsymbol{\gamma}}_{\Gamma_{B,\mathsf{k}}} \neq \emptyset.$$

Algorithm 4 Determining the product of arcs and edges in the general case

1: Determine the normalized product curve from the operand parameters.
2: Determine the intersection of the operands' log-Gauss map intervals.
3: Find the curve segment corresponding to the log-Gauss map intersection.
4: If the result segment does not touch the envelope, it is an arc, jump to step 6.
5: Fit an arc to the curve, so it is touching from the outside (see Algorithm 5).
6: Scale-rotate the result arc using the reciprocal of the normalization factor.

The special cases of edge and arc multiplication (when at least one operand is on a zero-crossing line or a zero-centered circle) can be simply implemented using addition in the logarithmic domain, because the logarithmic image of a zero-crossing edge and a zero-centered arc are both edges. The only difficulty is finding the point on the curve segment that corresponds to a given log-Gauss map, which is less straightforward than in case of the Gauss map value. In our implementation we used the built-in Matlab function *vpasolve* to numerically solve the inverse parametric equation of the curve segment[2]. The implementation of the general cases of edge and arc products require the geometric algebra described by Farouki et al in [12], and the normalization uses Theorem 2.2. Here, the product function is derived for the normalized (real one crossing vertical) line and (real one centered) circle. Since the normalization does not change the log-Gauss map value, we can determine the result curve the following Algorithm 4.

A simplified multiplication algorithm can be implemented when both operands are convex in the logarithmic domain. A convexity test can be implemented by checking if any of the arcs have negative radius, any of the edges have decreasing absolute value, or any of the vertices have higher log-Gauss map value at the preceding segment than the following one. The simplified algorithm works similar to the *MinkowskiSum* algorithm in [7, Ch. 13] with the modification that it should use the log-Gauss map value instead of the edge angle, and instead of adding vertices

[2] We solve the $\mathring{\gamma}(F,t_0) = \gamma_0$ equality to get t_0.

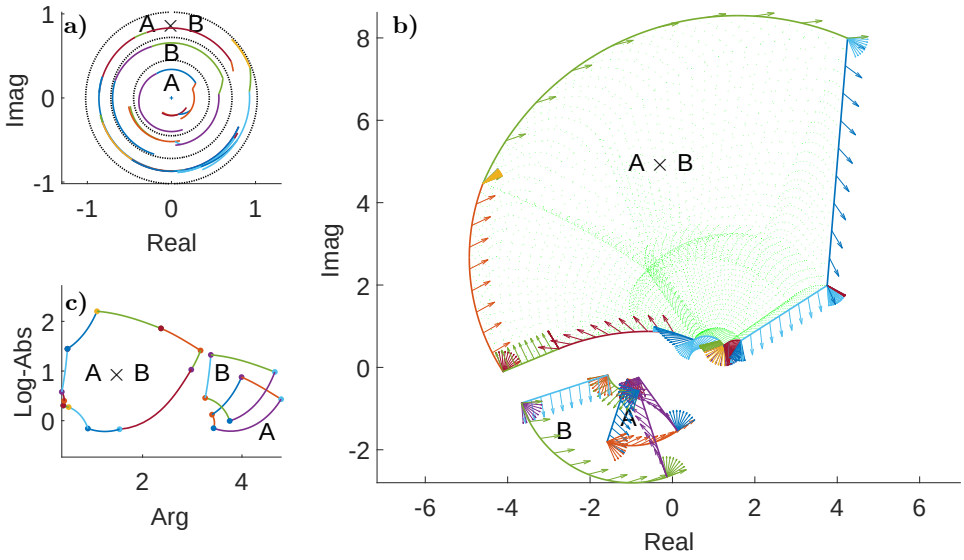


Figure 4: Two polar intervals , $\boldsymbol{A}$ and $\boldsymbol{B}$, and and their polyarc product after trimming. **a)** shows the log-Gauss maps presented as three concentric curves in the unit circle, where the angle and color indicates the log-Gauss map value $\mathring{\gamma}$, and the radius indicates the position on the curve t. (The radii has been offset to form the concentric rings.) One can observe that the result segments located in the outer ring are at the angular intersection of the operand segments in the inner rings. **b)** shows the intervals in the complex plane, where color of the arcs indicate the log-Gauss map value (e.g. $\mathring{\gamma}(\Gamma_A, t)$), and the arrows indicate the log-normal of the curves, which is the normal angle minus the angle of the vector pointing to the point of the curve from the origin. The colors and arrows allow the identification of the operand segment pairs that contribute to a given result boundary segment. The green point cloud indicates samples from the result interval formed by multiplying samples from the operand boundaries. **c)** shows the intervals in the complex logarithmic space, where $\log(\boldsymbol{A}\times\boldsymbol{B}) = \log(\boldsymbol{A}) + \log(\boldsymbol{B})$. One can observe that the result in this domain is the sum of the operands. While not indicated, the log-normal angle of a curve at a given point in this domain is the normal angle of the curve.

it should multiply operand segments with log-Gauss-map matching. The products of arcs and edges would be too long to present in this paper. The method used for fitting an arc to the product envelope curve is described in Algorithm 5 [3] . The implementation was derived from the results in [12] and can be found in the open-source code (see Supplementary materials).

[3]To ensure inclusive bounds, the arc is fitted from the "outside", which should be interpreted as the right hand side of the curve while facing in the increasing curve parameter direction.

Algorithm 5 Fitting an arc to a product curve segment touching the envelope

1: Determine the implicit function of the envelope curve.
2: Find the endpoints of the curve segment by numerically solving it for the log-Gauss map parameter bounds of the operands.
3: Find the center of the circle that goes through the endpoints and touches the envelope segment from the outside by numerically solving an inequality.
4: The fitted arc is defined by the center and the two endpoints.

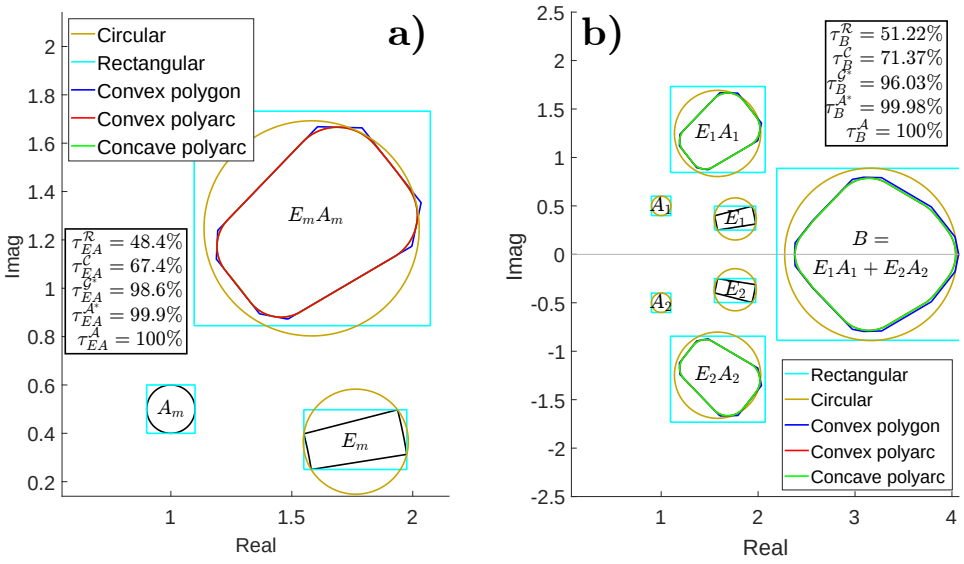


Figure 5: Example of a sensor array tolerance analysis using various complex interval types. **a)** shows the multiplication of a circular ($\boldsymbol{A}_{\mathsf{m}}^{\mathcal{C}} = \sum_{\mathsf{n}} \boldsymbol{A}_{\mathsf{m,n}}^{\mathcal{C}}$) and a polar ($\boldsymbol{E}_{\mathsf{m}}^{\mathcal{P}}$) interval. **b)** shows the summation of two such circular-polar products. Each plot contains a text-box with the tightness of the rectangular ($\mathcal{R}$), circular ($\mathcal{C}$), convex polygonal ($\mathcal{G}^*$), convex polyarc ($\mathcal{A}^*$) and concave polyarc ($\mathcal{A}$) representations of the result intervals ($\boldsymbol{E}_{\mathsf{m}}\boldsymbol{A}_{\mathsf{m}}$ and $\boldsymbol{B}$).

4 Case studies

4.1 Sensor array tolerance analysis

In this section, we present a case study that motivated the development of the polyarc interval type. The tolerance analysis of antenna arrays using interval analysis is an active research area within sensor array design and signal processing [18, 17]. We have previously analyzed the worst-case spatial response of acoustic arrays affected by calibration errors and mutual coupling using rectangular, circular and

polygonal interval [2]. None of these types could provide an exact representation of the complex interval of the array response, which resulted in relaxed bounds. Looking for a tighter solution, we found that given the physical model, the polyarc interval type yields an exact bound, which we demonstrate in the following.

Let $\{\boldsymbol{E}_{\mathsf{m}} \in \mathcal{P}(\mathbb{C}) \big| \mathsf{m} \in \{1..\mathsf{M}\}\}$ represent the combined amplitude and phase sensitivity interval of the individual elements of a sensor array, and let $\{\boldsymbol{A}_{\mathsf{m},\mathsf{n}} \in \mathcal{C}(\mathbb{C}) \big| \mathsf{m},\mathsf{n} \in \{1..\mathsf{M}\}\}$ represent the coupling coefficient interval of each pair of elements, including the self-coupling $\boldsymbol{A}_{\mathsf{n},\mathsf{n}} = 1$. Then, assuming a narrow-band, far-field operation, the complex response interval of the array is

$$\boldsymbol{B} = \sum_{\mathsf{m}} \boldsymbol{E}_{\mathsf{m}} \sum_{\mathsf{n}} \boldsymbol{A}_{\mathsf{m},\mathsf{n}}. \tag{8}$$

Since the boundary segments of the polar intervals are all from zero crossing lines or zero-centered circles, the product of a circular and a polar interval can be exactly represented using edges and arcs. Finally, the sum of polyarc intervals is polyarcular; therefore, the complex response interval is of the polyarc type: $\boldsymbol{B} \in \mathcal{A}(\mathbb{C})$.

While rectangular and circular types are computationally light, they introduce a significant loss of tightness at the type-casting and through the multiplication [29, 1]. The convex polygonal type can approximate the convex arcs with arbitrary precision, but this comes with a price of increased computational complexity [28]. We found that the polyarc type is an ideal choice for this application, because its concave implementation can provide perfect tightness, while its convex implementation provides tighter and faster calculation than the polygonal interval arithmetic. Figure 5 shows an example of the complex beampattern interval representation. A more detailed version of this case study, including run-time and accuracy comparison of the polyarc interval arithmetic with a number of other methods is available in [14].

4.2 Robot localization

Desrochers et al. introduce a measurement problem, in which the localization of a robot is performed by measuring the distance and direction of three landmarks with some tolerance [8]. In their paper, they use contractor programming, which is an interval analysis technique – for the outer approximation of the solution set, which is the region containing the possible locations. Since the problem can be also formulated as a complex interval arithmetic problem, in this section, we show a solution using the polyarc interval.

Let $\boldsymbol{r}_{\mathsf{n}} \in \mathcal{I}(\mathbb{C})$ represent the interval of range measurement and $\boldsymbol{\varphi}_{\mathsf{n}} \in \mathcal{I}(\mathbb{C})$ represent the interval of direction measurement of the n^{th} landmark at the known location $P_{\mathsf{n}} \in \mathbb{C}$. We can then represent the set of possible relative locations of the n^{th} landmark as a polar interval $\boldsymbol{A}_{\mathsf{n}} = \boldsymbol{r}_{\mathsf{n}} e^{i\boldsymbol{\varphi}_{\mathsf{n}}} \in \mathcal{P}(\mathbb{C})$. Given N landmarks, we can determine the set of the possible locations of the robot based on the combination of the measurement as

$$\boldsymbol{B} = \bigcap_{\mathsf{n}=1}^{\mathsf{N}} (P_{\mathsf{n}} - \boldsymbol{A}_n) \in \mathcal{A}(\mathbb{C}). \tag{9}$$

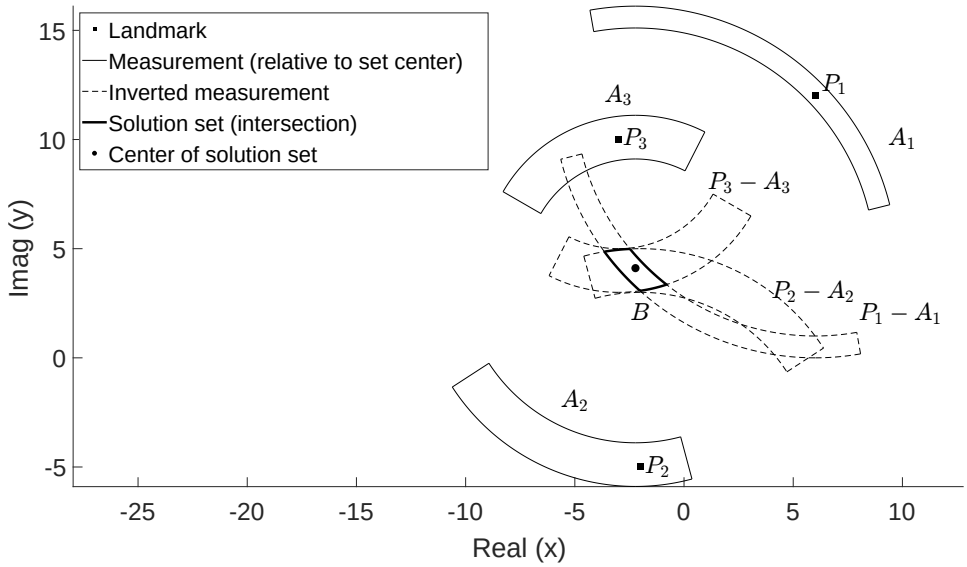


Figure 6: Example of a robot location problem solved with polyarc interval type given the landmark locations $P = \{6 + 12i, -2 + 5i, -3 + 10i\}$, range measurements $\boldsymbol{r} = \{[11, 12], [8, 10], [5, 7]\}$, and direction measurements $\boldsymbol{\varphi} = \{[14, 100], [-147, -75], [63, 150]\}^\circ$.

Since the result boundary consist of the boundary segments of translated polar intervals, it can be exactly representing using a polyarc curve. Figure 6 the example based on the paper [8]. It has to be noted that the intersection operation can result multiple non-connected intervals, as would be the case in this example if the third landmark would not be present. Such cases can be handled by combining contractors with a branch and prune algorithm. This, however, is not available in our current reference implementation.

5 Conclusion

In this paper, we showed that all commonly used complex interval types can be represented and arithmetically combined using the polyarc interval type with improved tightness and similar complexity as of the polygonal method. This makes the polyarc type a valuable option for performing calculations in various cases of interval analysis. We have shown that, similar to the polygonal type, simple arithmetic operations on convex polyarc intervals can be also performed with a computational complexity linearly dependent on the number of elements constituting the operand boundaries.

We presented an implementation of the polyarc interval arithmetic and provided an open-source code library as a reference (see Supplementary materials). We presented a case study from antenna design that served as our motivation to develop this method. We showed that in this particular case polyarc interval outperforms the existing methods, when high accuracy is desired. A second case study has been presented from robotics, where polyarc intervals provide an alternative to contractors. These examples show that the polyarc interval type is applicable in practical design tasks.

The reference implementation provided with this paper is for demonstrating the algorithms and generating reproducible experiments, but not intended for production use. The development of an open-source package in a non-proprietary language (e.g. Python) that would conform with the 1788-2015 - IEEE Standard for Interval Arithmetic is desirable, and would be natural continuation of the presented work.

When large number of intervals are combined by binary operations, the simplification of the result boundary is often desired to avoid the explosion of boundary element count. The removal of collinear vertices is an efficient technique to simplify polygons, which could be extended to polyarcs in the future by identifying joinable edges and arcs.

In our current implementation of the polyarc multiplication, we use the built-in Matlab function *vpasolve* to determine points on the result curves with a given log-Gauss map value, and to find the tightest arc approximating the non-polyarcular segments. Finding closed form solutions to these functions could significantly increase the speed of the algorithm.

Ironically, one of the most significant challenges in the implementation is the propagation of numerical errors, which is the origin of interval arithmetic. Such errors can make tests of inclusion and equality unreliable, which are indispensable for the geometric calculations with arcs and edges. While we could handle this problem for the majority of cases by setting tolerances based on our observations, a thorough analysis of an error propagation could result in a more robust implementation.

The derivation and implementation of non-linear functions on complex intervals is also desirable, and a potential direction for future research. Furthermore, a more detailed study of the robitic localization problem could be of interest.

Supplementary materials

More details about the polyarc arithmetic, including the formal proof of its arithmetic properties is available in a preprint document at https://arxiv.org/abs/2402.06430.

Our reference implementation of real and complex interval arithmetic methods is available as a Matlab package both in the live repository at https://github.com/unioslo-mn/ifi-complex-interval-arithmetic and in the frozen release at [3]. The package contains classes for real intervals, rectangular, polar, circular, convex polygonal, convex polyarc and concave polyarc type complex intervals.

References

[1] Anselmi, N., Salucci, M., Rocca, P., and Massa, A. Power pattern sensitivity to calibration errors and mutual coupling in linear arrays through circular interval arithmetics. *Sensors*, 16(6):791, 2016. DOI: 10.3390/s16060791.

[2] Arnestad, H. K., Geréb, G., Lønmo, T. I. B., Kirkebø, J. E., Austeng, A., and Näsholm, S. P. Worst-case analysis of array beampatterns using interval arithmetic. *The Journal of the Acoustical Society of America*, 153(6):1–7, 2023. DOI: 10.1121/10.0019715.

[3] Arnestad, H. K. and Geréb, G. Beampattern Interval Analysis Toolbox, 2022. DOI: 10.5281/zenodo.6856232.

[4] Boche, R. E. Complex interval arithmetic with some applications. Technical Report LMSC4-22-66-1, Locheed Missiles and Space, 1966. URL: https://www.reliable-computing.org/archive/Moores_early_papers/Boche_complex.pdf.

[5] Bruce, J. W. and Giblin, P. J. What is an envelope? *The Mathematical Gazette*, 65(433):186–192, 1981. DOI: 10.2307/3617131.

[6] Dawood, H. *Theories of Interval Arithmetic: Mathematical Foundations and Applications*. LAP LAMBERT Academic Publishing Gmbh & Co. KG, 2011. ISBN: 9783846501542.

[7] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. *Computational Geometry: Algorithms and Applications*. Springer, Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-77974-2.

[8] Desrochers, B. and Jaulin, L. A minimal contractor for the polar equation: Application to robot localization. *Engineering Applications of Artificial Intelligence*, 55:83–92, 2016. DOI: 10.1016/j.engappai.2016.06.005.

[9] Dongarra, J. J. BLAS (Basic Linear Algebra Subprograms), 1995. URL: https://netlib.org/blas/.

[10] Farouki, R. T., Han, C. Y., and Hass, J. Boundary evaluation algorithms for Minkowski combinations of complex sets using topological analysis of implicit curves. *Numerical Algorithms*, 40(3):251–283, 2005. DOI: 10.1007/s11075-005-4565-9.

[11] Farouki, R. T., Moon, H. P., and Ravani, B. Algorithms for Minkowski products and implicitly-defined complex sets. *Advances in Computational Mathematics*, 13(3):199–229, 2000. DOI: 10.1023/A:1018910412112.

[12] Farouki, R. T., Moon, H. P., and Ravani, B. Minkowski geometric algebra of complex sets. *Geometriae Dedicata*, 85(1):283–315, 2001. DOI: 10.1023/A:1010318011860.

[13] Gargantini, I. and Henrici, P. Circular arithmetic and the determination of polynomial zeros. *Numerische Mathematik*, 18(4):305–320, 1971. DOI: 10.1007/BF01404681.

[14] Geréb, G., Arnestad, H. K., Lønmo, T. I. B., Kirkebø, J. E., Näsholm, S. P., and Austeng, A. Exact power pattern bounds for arrays affected by calibration errors and mutual coupling. *IEEE Transactions on Antennas and Propagation*, pages 1–1, 2025. DOI: 10.1109/TAP.2025.3570168.

[15] Giardina, C. and Dougherty, E. *Morphological methods in image and signal processing*. Society for Industrial and Applied Mathematics, 1988. DOI: 10.1137/1032073.

[16] Hansen, E. R. A generalized interval arithmetic. In Nickel, K., editor, *Interval Mathematics*, Lecture Notes in Computer Science, pages 7–18, Berlin, Heidelberg, 1975. Springer. DOI: 10.1007/3-540-07170-9_2.

[17] He, G., Gao, X., and Zhang, R. Impact analysis and calibration methods of excitation errors for phased array antennas. *IEEE Access*, 9:59010–59026, 2021. DOI: 10.1109/ACCESS.2021.3073222.

[18] He, G., Gao, X., Zhou, H., and Zhu, H. Comparison of three interval arithmetic-based algorithms for antenna array pattern upper bound estimation. *Electronics Letters*, 55(14):775–776, 2019. DOI: 10.1049/el.2019.1229.

[19] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E. *Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer, Berlin, 2001. DOI: 10.1007/978-1-4471-0249-6.

[20] Kearfott, R. E., Musaev, E. A., Nesterov, V. M., and Yakovlev, A. G. Reliable Computing | Volumes and issues, 1995. URL: https://link.springer.com/journal/11155/volumes-and-issues.

[21] Kreinovich, V. and Lauter, C. Interval Computations, 2023. URL: https://www.cs.utep.edu/interval-comp/main.html.

[22] Moore, R. E. *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Stanford University California, 1963.

[23] Moore, R. E., Kearfott, R. B., and Cloud, M. J. *Introduction to Interval Analysis*. SIAM, Philadelphia, 2009. ISBN: 9780898716696.

[24] Ohta, Y., Gong, L., and Haneda, H. Polygon interval arithmetic and design of robust control systems. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 1065–1067 vol.2, 1990. DOI: 10.1109/CDC.1990.203765.

[25] Ohta, Y. Nonconvex polygon interval arithmetic as a tool for the analysis and design of robust control systems. *Reliable Computing*, 6(3):247–279, 2000. DOI: 10.1023/A:1009926413485.

[26] Petkovic, M. and Petkovic, L. D. *Complex Interval Arithmetic and Its Applications*. John Wiley & Sons, Berlin, 1998. ISBN: 9783527401345.

[27] Rump, S. M. INTLAB — INTerval LABoratory. In Csendes, T., editor, *Developments in Reliable Computing*, pages 77–104. Springer Netherlands, Dordrecht, 1999. DOI: 10.1007/978-94-017-1247-7_7.

[28] Tenuti, L., Anselmi, N., Rocca, P., Salucci, M., and Massa, A. Minkowski sum method for planar arrays sensitivity analysis with uncertain-but-bounded excitation tolerances. *IEEE Transactions on Antennas and Propagation*, 65(1):167–177, 2017. DOI: 10.1109/TAP.2016.2627548.

[29] Zhang, Y., Zhao, D., Wang, Q., Long, Z., and Shen, X. Tolerance analysis of antenna array pattern and array synthesis in the presence of excitation errors. *International Journal of Antennas and Propagation*, 2017. DOI: 10.1155/2017/3424536.