Directed Interval Arithmetic in *Mathematica*: Implementation and Applications

E. D. Popova
Institute of Biophysics, Bulgarian Academy of Science
Acad. G. Bonchev str. bldg. 21, BG-1113 Sofia, Bulgaria
e-mail: epopova@bgearn.acad.bg

C. P. Ullrich
Institute for Computer Science, University of Basel
Mittlere str. 142, CH-4056 Basel, Switzerland
e-mail: ullrich@ifi.unibas.ch

This work was supported by the Swiss National Science Foundation and the Bulgarian National Science Fund under grant No. I-507/95.

Contents

1 Introduction

Computer algebra systems developed rapidly last decade and attracted many users with their possibilities for symbolic algebraic manipulations, exact integer/rational computations, good vizualisation facilities and arbitrary precision floating-point arithmetic. Facing real-life computational problems, however, these systems proved that there is no universal computational tool. The arithmetic operations on arbitrary precision floating-point numbers in Maple [3], for example, proved to be accurate within 0.6 ulp (unit in the last place). But this is claimed only for single operations and no features exist for studying the effects of the propagation either of roundoff errors or of data uncertainties. Doing computations with very large integer/rational numbers or symbolically may cause for some problems a super-exponential growth in the storage requirement or may be unbounded in time. The numerical stability of the final expression may be in doubt too.

On the other hand, there exists a considerable two-way traffic between numerical and symbolic computations [29]. A number of classical algebraic algorithms (e. g. Gaussian elimination for linear equation solving) were "embedded" in floating-point systems providing a supplementary monitoring of possible singularities and potentially enhanced information return. The usage of floating-point and interval arithmetic in intermediate computations appeared to result in a dramatic speed-up for some algebraic algorithms [4, 9]. Such hybrid algorithms combine the speed of numerical computations with the exactness of symbolic methods providing still guaranteed correct results. Most scientific/engineering problems require also a combination of analytic and numerical techniques. Many real-life problems deal with interval input data instead of approximate values, i. e. design objectives and constrains, automatic control, identification and interpolation under bounded uncertainties.

All these forced the developers to supply computer algebra systems with interval arithmetic [5, 17, 11]. Starting with an experimental demonstrative interval arithmetic package, *Mathematica* [35] has included `Interval` as a kernel function in its 2.2 version. The interval arithmetic in *Mathematica* Version 2.2 is genuine interval arithmetic, complete with outward rounding and multi-intervals [17]. The interval data object is integrated smoothly within the rest of the system and most of the built-in functions are designed to work together in a unified manner. With its graphic, symbolic and numerical capabilities *Mathematica* is an excellent tool for education and exploration.

Although conventional interval arithmetic [1, 28] is widely used in interval analysis and has numerous applications, it possesses only few algebraic properties. Lattice operations are not closed with respect to the inclusion relation. Due to the lack of inverse elements with respect to the addition and multiplication operations, the solution of the algebraic interval equations $A + X = B$ and $A \times Y = B$, can not be generally expressed in terms of the interval operations even if they actually exist. There is no distributivity between addition and multiplication except for certain special cases. A considerable scientific effort is put into developing special methods and algorithms that try to overcome the difficulties imposed by the algebraic incompleteness of the conventional interval arithmetic structure. For example, arithmetic operations between conventional intervals can be used for rough outer inclusion of functional ranges. But the bounds computed by naive interval evaluation are often too pessimistic to be useful. Again several strategies have been developed to compute tighter bounds. Arithmetic operations between conventional intervals are also of little use for the computation of inner inclusions.

Algebraic incompleteness and disadvantages of the conventional interval arithmetic have led to various proposals for possible extensions. The algebraic completion of the conventional interval arithmetic structure leads to a set of directed intervals and an extension of the conventional interval operations. Developed basically by E. Kaucher [13]–[16], further investigated by E. Gardenes et al. [7], and others [2, 25], directed interval arithmetic is obtained as an extension of the set of normal intervals by improper intervals and a corresponding extension of the definitions of the interval arithmetic operations. The corresponding extended interval arithmetic structure possesses group properties with respect to addition and multiplication operations. Directed interval arithmetic is useful for a straightforward computation of inner and outer inclusion of functional ranges [23]. It seems to be promising for the numerical solution of various practical problems related to interpolation and identification under uncertainties, control theory etc. [7, 23, 34]. Some conventional interval problems can be embedded and solved there effectively [7, 19, 34]. An attractive goal is to make use of the algebraic completeness of the directed interval arithmetic embedding it in a computer algebra system and investigating how the algebraic properties of this arithmetic can be exploited for symbolic manipulation of interval expressions, automatic theorem proving, developing of hybrid symbolic-numeric algorithms and effective implementation of certain numerical algorithms.

First step to this goal was the implementation of the basic arithmetic on directed intervals in computer algebra system *Mathematica*. This report presents an experimental *Mathematica* package `directed.m` that extends *Mathematica* interval capabilities by providing a new data object (`Directed`) representing directed multi-intervals, as well as operations and functions for basic arithmetic on them. We give here an overview of the package design and its usage. The advantages of the implemented extension of the conventional interval arithmetic is demonstrated on some numerical algorithms involving directed intervals and implemented as *Mathematica* functions. Several examples outline possible applications and directions for further investigations.

2 Directed Interval Arithmetic

This section is devoted to the interval arithmetic. The set of finite normal (proper) intervals $IR = \{[a,b] \mid a \leq b, a, b \in R\}$ is a commutative semigroup with cancellation law as regard to the operation addition, resp. multiplication (in the latter case intervals containing zero are excluded) [14]. It is well known that if S is a commutative semigroup with cancellation law, then S can be isomorphically embedded into an unique group $G(S)$, whose elements are couples of elements (A,B), $A, B \in S$ (more precisely all such couples, factorized by the equivalence relation $(A,B) = (C,D) \Longleftrightarrow A + D = B + C$). E. Kaucher shows [16] that the set of intervals $\{[a,b] \mid a, b \in R\}$ is a group under a properly defined addition, and, what is more interesting, multiplication, and therefore this set is equivalent (up to isomorphism) to the group of factorized couples of normal intervals mentioned above.

S. Markov shows [26] that under certain additional conditions a commutative semigroup with cancellation law, S, can be uniquely embedded into the group (called directed) $D = S \otimes \{+,-\}$ with elements of the form $(A;\pm)$, $A \in S$ (called directed elements). This idea generalizes the well known scheme of extending the set of natural numbers up to the group of whole numbers and allows a straightforward interpretation of the results in the directed group in terms of the elements of the original semigroup S. An important ingredient of Markov's conditions is the existence of a negation operator in the semigroup S [26]. It is shown [26] that the set $\{[a,b] \mid a, b \in R\}$ of Kaucher's (extended, generalized) intervals satisfies Markov's conditions and therefore is a directed group in the sense of Markov's theorem; furthermore the extended intervals are directed elements of IR (directed intervals) satisfying group algebraic properties.

In this work we make use of the interpretation of the extended intervals as "directed intervals", given in [26], especially in relation to their application to computation of functional ranges [23]. For this reason we prefer to call the extended intervals directed intervals. However, in the software applications we shall not make use of the directed form $(A;\pm)$ of the intervals because the arithmetic operations on directed intervals in end-point form can be implemented at no additional cost comparing to the conventional interval arithmetic, while an implementation using the directed form will be more costly [32].

2.1 Basic Formulae

In this section we shall briefly outline some basic formulae of the extended interval arithmetic structure $\mathcal{K}$ proposed by E. Kaucher [13]–[16].

The set of all finite proper intervals $IR = \{[a,b] \mid a, b \in R,\ a \leq b\}$ is extended into the set $D = \{[a,b] \mid a, b \in R\} \cong R^2$ of all ordered couples of finite real numbers called directed intervals. A directed interval $A = [a^-, a^+] \in D$ is either proper if $a^- \leq a^+$, or improper if $a^- \geq a^+$, so that

$$D = \{[a,b] \mid a, b \in R\} = IR \cup \overline{IR}, \qquad \overline{IR} = \left\{[a^-, a^+] \mid a^- \geq a^+;\ a^-, a^+ \in R\right\}.$$

Denote $\mathcal{T} = \{A \in IR \mid a^- a^+ \leq 0\} \cup \{A \in \overline{IR} \mid a^- a^+ \leq 0\} = Z \cup \overline{Z}$. For $A \in D$ the symbol a^s with $s \in \{+,-\}$ denotes certain end-point of A and the "product" st for $s, t \in \{+,-\}$ is defined by $++ = -- = +$ and $+- = -+ = -$, so that a^{s+} is well defined.

For a directed interval A define "sign" $\sigma : D \setminus \{[a^-, a^+] \mid a^- a^+ < 0\} \to \{+, -\}$ by

$$\sigma(A) = \begin{cases} +, & \text{if } (0 \le a^-) \; and \; (0 \le a^+); \\ -, & \text{if } (a^- \le 0) \; and \; (a^+ \le 0) \quad (\text{but } A \neq [0, 0]), \end{cases}$$

and a binary variable "direction" by

$$\tau(A) = \begin{cases} +, & \text{if } a^- \le a^+, \\ -, & \text{otherwise} . \end{cases} \tag{1}$$

The operations of the interval arithmetic structure $\mathcal{K} = \{D, +, \times, \subseteq\}$ are extensions of the interval arithmetic relation and operations from the conventional interval space $\{IR, +, \times, /, \subseteq\}$ [1, 28] into D

$$A \subseteq B \iff (b^- \le a^-) \; and \; (a^+ \le b^+), \quad \text{for } A, B \in D; \tag{2}$$

$$A + B = [a^- + b^-, a^+ + b^+], \quad \text{for } A, B \in D ; \tag{3}$$

$$A \times B = \begin{cases} [a^{-\sigma(B)} b^{-\sigma(A)}, a^{\sigma(B)} b^{\sigma(A)}], & \text{for } A, B \in D \setminus \mathcal{T}, \\ [a^{\delta\tau(B)} b^{-\delta}, a^{\delta\tau(B)} b^{\delta}], \; \delta = \sigma(A), & \text{for } A \in D \setminus \mathcal{T}, \; B \in \mathcal{T}, \\ [a^{-\delta} b^{\delta\tau(A)}, a^{\delta} b^{\delta\tau(A)}], \; \delta = \sigma(B), & \text{for } A \in \mathcal{T}, \; B \in D \setminus \mathcal{T}, \\ [\min\{a^- b^+, a^+ b^-\}, \; \max\{a^- b^-, a^+ b^+\}], & \text{for } A, B \in Z, \\ [\max\{a^- b^-, a^+ b^+\}, \; \min\{a^- b^+, a^+ b^-\}], & \text{for } A, B \in \overline{Z}, \\ 0 , \quad \text{for } A \in Z, \; B \in \overline{Z} \quad \text{or} \quad A \in \overline{Z}, \; B \in Z . \end{cases} \tag{4}$$

Note that according to definition (2) any improper interval $A = [a^-, a^+]$ such that $a^+ \le b \le a^-$ is contained in the point interval $B = [b, b]$, $A \subseteq B$.

From (4) we have $(-1) \times B = [-b^+, -b^-] = -B$ for $B \in D$. Thus the extension of the conventional interval subtraction into D can be obtained as a composite operation

$$A - B = A + (-B) = [a^- - b^+, a^+ - b^-], \qquad A, B \in D. \tag{5}$$

The substructures $(D, +, \subseteq)$ and $(D \setminus \mathcal{T}, \times, \subseteq)$ of $\mathcal{K}$ are isotone groups [13]. The inverse elements with respect to the operations $+$ and $\times$ are:

$$-_h A = [-a^-, -a^+], \quad \text{for } A \in D;$$
$$1/_h A = [1/a^-, 1/a^+], \quad \text{for } A \in D \setminus \mathcal{T}.$$

The monadic operator conjugation (called also dual) defined by

$$A_- = [a^+, a^-] = -_h(-A) = -(-_h A) \tag{6}$$

expresses an element-to-element symmetry in D and has the properties:

$$A \subseteq B \iff A_- \supseteq B_-, \qquad (A \circ B)_- = A_- \circ B_-, \qquad \circ \in \{+, -, \times, /\}.$$

For $A \in D \setminus \mathcal{T}$ there exists also a unique operator "set inversion" $1/A = 1/_h A_- = [1/a^+, 1/a^-]$ such that $1/_h(1/A) = 1/(1/_h A) = A_-$. Thus, the extension of the conventional interval operation A/B for $A \in D$, $B \in D \setminus \mathcal{T}$ is also obtained as a composite operation:

$$A/B = A \times (1/B) = \begin{cases} [a^{-\sigma(B)}/b^{\sigma(A)}, a^{\sigma(B)}/b^{-\sigma(A)}], & \text{for } A, B \in D \setminus \mathcal{T}, \\ [a^{-\delta}/b^{-\delta\tau(A)}, a^{\delta}/b^{-\delta\tau(A)}], \; \delta = \sigma(B), & \text{for } A \in \mathcal{T}, \; B \in D \setminus \mathcal{T}. \end{cases} \tag{7}$$

D is a lattice with respect to $\subseteq$ with the following lattice operations:

$$\inf_{\subseteq}(A,B) = A \wedge B = [\max\{a^-,b^-\},\min\{a^+,b^+\}], \quad (8)$$
$$\sup_{\subseteq}(A,B) = A \vee B = [\min\{a^-,b^-\},\max\{a^+,b^+\}]. \quad (9)$$

The lattice operations satisfy the following properties

$$(A \circ B) + C = (A + C) \circ (B + C) \quad \text{for } A,B,C \in D \text{ and } \circ \in \{\wedge, \vee\},$$
$$(A \circ B) \times C = (A \times C) \circ (B \times C) \quad \text{for } A,B,C \in D \setminus \mathcal{T} \text{ and } \circ \in \{\wedge, \vee\},$$
$$(A \wedge B)_- = A_- \vee B_-.$$

Another order relation is also defined by

$$A \preceq B \iff (a^- \le b^-) \; and \; (a^+ \le b^+) \quad (10)$$

with the properties

$$A \preceq B \iff A_- \preceq B_-, \quad A \preceq B \iff -A_- \preceq -B_-, \quad A \preceq B \Longrightarrow A + C \preceq B + C.$$

Let $R^* = R \cup \{-\infty, \infty\}$ and denote by $D_{\mathcal{I}} = \{[a,b] \mid a,b \in R^*\}$ the set of all finite and infinite directed intervals. The intervals from $D_{\mathcal{I}}$ are called *inner directed intervals* in contrast to the *outer directed intervals* (proper or improper), obtained by division by intervals containing/contained in zero to be defined below.

Using that $-\infty \le a \le +\infty$ for all $a \in R^*$ and the conventional rules for manipulations with infinities (see for example [18]) the definitions of the relation $\subseteq$ and the arithmetic operations $+, -, \times, /$ are extended from $D \times D$ into $D_{\mathcal{I}} \times D_{\mathcal{I}}$ by replacing D with $D_{\mathcal{I}}$ in (2)–(5), (7).

For some Newton-like algorithms using conventional interval arithmetic it is essential to divide by an interval containing zero. The operation $1/B$ for $B \in \{[a^-,a^+] \mid a^-a^+ < 0\}$ is defined [13] as a set of two intervals as follows:

$$1/B = 1/[b^-,b^+] = \{[-\tau(B)\infty, 1/b^-], \; [1/b^+, \tau(B)\infty]\}.$$

Such a set of two equally directed inner intervals (one involving ∞, the other $-\infty$) is called *outer directed interval.* These intervals are generalization of the so called Kahan's intervals [12], [20].

2.2 Properties and Laws

The following properties of the structure $\mathcal{K} = (D, +, \times, _-, \subseteq)$ hold:

K1. $A \circ B = B \circ A$, for $A, B \in D$, $\circ \in \{+, \times\}$.

K2. $(A \circ B) \circ C = A \circ (B \circ C)$, for $A, B, C \in D$, $\circ \in \{+, \times\}$.

K3. $X = [0,0] = 0$ and $Y = [1,1] = 1$ are the unique neutral elements with respect to the addition and multiplication operations. That is for all $A \in D$

$$A = X + A \Leftrightarrow X = [0,0], \qquad A = Y \times A \Leftrightarrow Y = [1,1].$$

K4. Every element $A \in D$ has an unique inverse element with respect to $+$ and every element $A \in D \setminus \mathcal{T}$ possesses an unique inverse element with respect to $\times$. These are the elements $-A_-$, resp. $1/A_-$, i. e.:
$0 = A - A_-$ and $1 = A/A_-$.

K5. i) For $A, B, C, A + B \in D \setminus \mathcal{T}$

$$(A + B) \times C = (A \times C_{\sigma(A)\sigma(A+B)}) + (B \times C_{\sigma(B)\sigma(A+B)});$$

ii) For $A, B, C \in IR$ we have $A \times (B + C) \subseteq A \times B + A \times C$;
iii) For $A, B, C \in \overline{IR}$ we have $A \times (B + C) \supseteq A \times B + A \times C$.

K6. Let $\circ \in \{+, -, \times, /\}$. Then $X \subseteq X_1, \Rightarrow X \circ C \subseteq X_1 \circ C$. As a corollary we have $X \subseteq X_1,\ Y \subseteq Y_1 \Rightarrow X \circ Y \subseteq X_1 \circ Y_1$.

K7. D is a lattice with respect to $\subseteq$. The lattice operations are the intersection (8) and the convex hull (9) of two $\mathcal{K}$-intervals. The lattice operations are closed with respect to the extended inclusion relation (2).

2.3 Roundings and Inclusions

Analogous to the conventional interval computer arithmetic [18] a computer arithmetic for directed intervals is defined by semimorphism [6].

Let SR^* be a symmetric screen over R^* and $SD_{\mathcal{I}} = \{[a^-, a^+] \in D_{\mathcal{I}} \mid a^-, a^+ \in SR^*\}$, then $\{SD_{\mathcal{I}}, \subseteq\}$ is a screen of $\{D_{\mathcal{I}}, \subseteq\}$.

Define rounding $\square : D_{\mathcal{I}} \longrightarrow SD_{\mathcal{I}}$ as a monotonic function with the properties:

1. $\square\,(A) = A, \quad A \in SD_{\mathcal{I}}$;
2. $A \subseteq B \Longrightarrow \square\,(A) \subseteq \square\,(B), \quad$ for $A, B \in D_{\mathcal{I}}$;
3. For $A, B \in D_{\mathcal{I}}$

$$A \subseteq \square\, A, \quad \square = \diamond \quad \text{(outward rounding)},$$
$$A \supseteq \square\, A, \quad \square = \bigcirc \quad \text{(inward rounding)},$$
$$\diamond A = [\nabla a^-, \triangle a^+], \qquad \bigcirc A = [\triangle a^-, \nabla a^+],$$

where $\nabla, \triangle$ are the corresponding directed roundings $\nabla, \triangle : R^* \longrightarrow SR^*$ [18].

If $\circ \in \{+, -, \times, /\}$ is an arithmetic operation in $D_{\mathcal{I}}$, the corresponding computer operation $\boxdot$ in $SD_{\mathcal{I}}$ is defined by

$$A \boxdot B = \square\,(A \circ B), \quad \text{for} \quad A, B \in SD_{\mathcal{I}}, \quad \square \in \{\diamond, \bigcirc\}.$$

The explicit formulae for the computation of the result of the extended interval operations in $SD_{\mathcal{I}}$ are summarized as follows:
For $A, B \in SD_{\mathcal{I}}$ and $\circ \in \{+, -, \times, /\}$

$$A \mathbin{\diamond\!\!\!\cdot} B := \diamond\,(A \circ B) = [\nabla\,(A \circ B)^-,\ \triangle\,(A \circ B)^+];$$
$$A \odot B := \bigcirc\,(A \circ B) = [\triangle\,(A \circ B)^-,\ \nabla\,(A \circ B)^+].$$

The computer operations on directed intervals are inclusion isotone:

$$A \subseteq B \Longrightarrow A \boxdot C \subseteq B \boxdot C, \quad \text{for } A, B, C \in SD_{\mathcal{I}}, \ \circ \in \{+, -, \times, /\}, \ \Box \in \{\Diamond, \bigcirc\}.$$

The following inclusion assertions [7] are of major importance in obtaining inner and outer numerical approximations:

- For $A \in SD_{\mathcal{I}}$

$$\begin{array}{rclcccccl} (\Diamond(A_-))_- & = & \bigcirc A & \subseteq & A & \subseteq & \Diamond A & = & \bigcirc(A_-) \\ \bigcirc(A_-) & = & (\Diamond A)_- & \subseteq & A_- & \subseteq & (\bigcirc A)_- & = & \Diamond(A_-). \end{array}$$

- For $A \in SD_{\mathcal{I}}$ and $\circ \in \{+, -, \times, /\}$

$$\begin{array}{rcccl} (A_- \mathbin{\diamond\!\!\!\cdot}\ B_-)_- & \subseteq & A \circ B & \subseteq & A \mathbin{\diamond\!\!\!\cdot}\ B \\ A \odot B & \subseteq & A \circ B & \subseteq & (A_- \odot B_-)_- \,. \end{array}$$

- Let $F[\{\circ_1, \ldots, \circ_m\}, \{A_1, \ldots, A_n\}]$ be a rational function where $\circ_i \in \{+, -, \times, /\}$, $i = 1, \ldots, m$ and $A_j \in D_{\mathcal{I}}$, $j = 1, \ldots, n$, then

$$F[\{\odot_i\}_{i=1}^m, \{\bigcirc A_j\}_{i=1}^n] \subseteq F[\{\circ_i\}_{i=1}^m, \{A_j\}_{j=1}^n] \subseteq F[\{\mathbin{\diamond\!\!\!\cdot}_i\}_{i=1}^m, \{\Diamond A_j\}_{j=1}^n]$$

  and

$$\begin{array}{c} F[\{\mathbin{\diamond\!\!\!\cdot}_i\}_{i=1}^m, \{\Diamond\,((A_j)_-)\}_{j=1}^n]_- = F[\{\odot_i\}_{i=1}^m, \{\bigcirc A_j\}_{i=1}^n] \\ F[\{\mathbin{\diamond\!\!\!\cdot}_i\}_{i=1}^m, \{\Diamond A_j\}_{j=1}^n] = F[\{\odot_i\}_{i=1}^m, \{\bigcirc\,((A_j)_-)\}_{j=1}^n]_- \end{array}$$

Last equalities show that inner approximation/inclusion can be obtained in directed interval arithmetic only by means of outward directed roundings and operation conjugation instead of using inward directed roundings. That means there is no need of implementation of directed interval arithmetic operations with inward rounding.

3 Implementation

3.1 *Mathematica* package directed.m

The *Mathematica* package `directed.m` was designed as an experimental demonstrative package intended to provide some functionality that can not be obtained by conventional interval arithmetic supported by the *Mathematica* kernel function `Interval`. Main attention at this stage of the project was paid to the implementation of the basic operations and functions on directed intervals. It was done according to the specifications of BIAS (Basic Interval Arithmetic Subroutines), generalized for single directed intervals [32]. Designing directed interval arithmetic of *Mathematica* we tried to keep and preserve all the functionality provided by the kernel function `Interval` since conventional interval arithmetic is a special case of directed interval arithmetic. `Interval` data object supports conventional multi-intervals and thus the so called Kahan's intervals [12, 20] and the arithmetic on them as a special case. This and versatility that provide list data structures and computer algebra system itself gave us good reasons to implement Kahan's intervals extended to *inner* and *outer* directed intervals (see the end of Section 2.1) furthermore that multi-intervals have attracted some researchers to use them in a variety of algorithms and programming systems [10, 33]. Meaning of the operations and functions implemented in `directed.m` on directed multi-intervals is described in Section 3.2.

Implementing directed interval arithmetic we followed the common naming conventions in *Mathematica*. The names of all functions, variables, options and constants start with capital letter, e. g. `Directed`. If a name consists of two or more words, then the first letter of each word is capitalized, e. g. `IntHull, IntIntersection`, etc. Most names of objects built into *Mathematica* are complete words. Abbreviations are only used where they are extremely well-known. At this experimental stage of the development of the package `directed.m` we shortened the names as much as possible. We omitted the word `Interval` from most of the names and used `Directed`, naming data objects representing directed intervals instead of `DirectedInterval` or `Proper` instead of `ProperInterval`. Some directed interval functions start with the abbreviation `Int` instead of `Interval`, e. g. `IntHull, IntIntersection`, etc. Predicate functions returning `True` or `False` have names ending with `Q`, e. g. `IntMemberQ, InclusionQ`, etc.

Mathematica allows defining of transformation rules for functions that are already built into *Mathematica*. As a result, one can enhance, or modify, the features of built-in functions. Why have we not applied this to the built-in *Mathematica* function `Interval`? This capability is powerful, but potentially dangerous. If the given rules are incorrect, then *Mathematica* will give incorrect answers. At the experimental stage of this package it was better to define and use a new data object. Another reason was that we could not overcome the infinite recursion rounding outwardly the list elements representing directed intervals, which required the introduction of an optional parameter `Round` in the syntax of directed interval data object. Names, definitions, and/or properties of some functions defined for `Interval` were not so appropriate for directed intervals and we have used other names and/or implement other definitions. Differences in the behaviour of `Interval` and `Directed` data objects and functions are discussed in Section 3.3. In order to provide a convenient manipulation with directed intervals many new functions were defined.

The Appendix lists *Mathematica* package `directed.m` containing definitions of data object `Directed` representing directed multi-intervals and definitions of the basic arithmetic on directed intervals.

Certain compromises have to be made in a system like *Mathematica*, which must run on a variety of machines. The outward rounding is performed *a posteriori* (after the interval is calculated) rather than as directed rounding in hardware. In particular the end-points are calculated with whatever arithmetic is appropriate and then they are adjusted outwardly several ulps. For the arithmetic operations addition, subtraction, and multiplication the error in the arithmetic is assumed to be less than 1 ulp and adjusting outwardly by 1 ulp guarantees correct bounds. For division (which is in fact multiplication by the inverse) two rounding errors may have occurred and so the result has to be adjusted by 3 ulps according to an adequate error estimation. For integer powers of intervals appropriate adjustments are likewise made to ensure that the resulting bounds are in fact correct. This package uses the `ulp` function (see the Appendix) from *Mathematica* package `NumericalMath‘IntervalAnalysis‘`.

Almost all operations and functions on directed intervals require some comparison between end-points of the arguments. *Mathematica* is a symbolic processing system and neither its relational operations $<, <=, >, >=$, nor `Min`, `Max` functions evaluate exact singletons like `Sin[2], Cos[4]`, etc. and mathematical constants (`Infinity, E, Pi, ...`), which may occur in the end-points of a directed interval. In order to provide proper manipulation with directed intervals involving such entities, the numerical approximation takes a substantial part of the operations and functions on directed intervals. Normalization procedure of the directed multi-intervals merges the set-theoretical intersecting, equally directed elements of a directed multi-interval into single elements and then put all elements into normal (canonical) order. The canonical order in *Mathematica* is determined by the character ordering, so numerical approximation of the interval end-points is used at the normalization procedure, too. The internal algorithms that *Mathematica* uses to evaluate mathematical functions are set up to maintain as much precision as possible. In some cases, however, it is simply impractical to do this, and *Mathematica* gives results that have lower precision [35]. The fact that different ways of doing the same calculation can give different numerical answers means, among other things, that comparison between approximate real numbers must be treated with care.

```
In[1] := {a, b} = { Interval[{1/3, 1/3}] // N, N[{1/3, 1/3}] }
Out[1] = {Interval[{0.3333333333333333, 0.3333333333333334}],
  {0.3333333333333333, 0.3333333333333333}}

In[2] := {a[[1]] === b, a[[1]] == b}
Out[2] = {True, True}

In[3] := {a[[1, 2]] === b[[2]], a[[1, 2]] == b[[2]]}
Out[3] = {True, True}

In[4] := a[[1, 2]] > b[[2]]
Out[4] = False
```

Implementing directed interval arithmetic we have taken all the precautions to provide correct behaviour of the implemented functions.

```
In[5] := (a[[1, 2]] - b[[2]]) > 0
Out[5] = True
```

Expressions like 0/0, 0×`Infinity` or `Infinity - Infinity` are examples of indeterminate numerical results. Whenever an indeterminate result is produced in an arithmetic computation, *Mathematica* prints a warning message, and then returns `Indeterminate` as the result of computation. `Indeterminate` "poisons" any arithmetic computation, and leads to an indeterminate result [35]. Thus, the symbol `Indeterminate` plays a role in *Mathematica* similar to the Not-a-Number (NaN) object in the IEEE Floating-Point Standard. Implementing directed interval arithmetic in *Mathematica* we followed the generalization of BIAS specifications [32] which extends the scope of validity of interval operations and provide consistency between interval and IEEE arithmetics. The *Mathematica* symbol `Indeterminate` participates in `Interval` data object as well as in `Directed` data object but the arithmetic operations and functions on directed intervals involving `Indeterminate` follow the model defined in [31] and specified in [32]. The necessity of handling the symbol `Indeterminate` leads to an essential aspect of symbolic systems like *Mathematica* that the conditions may yield neither `True` nor `False`. For example, the condition `x < y` does not yield `True` or `False` unless `x` and `y` have specific numeric values. Most of the functions implementing directed interval arithmetic substantially use `If[test, t, f, u]` giving `u` if `test` evaluates to neither `True` nor `False`.

Mathematica supports *upvalues*, which allow definitions to be associated with symbols that do not appear directly as their head. Consider, for example, a definition like `Min[Directed[{x_, y_}]] := rhs`. One possibility is that this definition could be associated with the symbol `Min`, and considered as a *downvalue* of `Min`. This is however not the best thing either from the point of view of organization or efficiency. `Min` is a built-in *Mathematica* function, while `Directed` is a function we have added and a definition should be given as an upvalue for `Directed` since the function `Min` is more common then `Directed`. In this case we think of definition for `Min[Directed[{x_, y_}]]` as giving relations satisfied by `Directed`. As a result, it is more natural to treat the definitions as upvalues for `Directed` than as downvalues for `Min`. By giving a definition for `Directed[x]` + `Directed[y]` as an upvalue for `Directed`, we associate the definition with `Directed`. In this case *Mathematica* tries the definition when it finds a `Directed` inside a function such `Plus`. Since `Directed` occurs much less frequently than `Plus`, this is a much more efficient procedure. All the functions related to directed intervals are given upvalues for `Directed` (see Appendix).

According to the generalization of BIAS specification [32], `Sign` and `Direction` functions are defined to deliver integer values ± 1 or 0, instead of taking their values from the set of the two symbols $\{+, -\}$. This is of great importance for symbolic manipulation with directed intervals. Most of the formulae, expressions and laws involving directed intervals have conditional form depending on sign and/or direction of some of the participating directed intervals. Having integer values ± 1 or 0 as a result of `Sign` and `Direction` function, the access to the end-points of the directed intervals becomes straightforward. Thus, like the concise $\pm$ representation of formulae involving directed intervals (Section 2.1), we have a concise programming of latter and an open possibility for symbolic manipulation (see, for example, function `Times` in the Appendix).

Evaluation of `Directed` is slower than the evaluation of `Interval` due to two main reasons. First, the slower speed is resulting from running interpreted code instead of kernel function. The second reason is given by more complicated algorithms of some functions with a result depending on the direction of the arguments. The definition requirement "first end-point $\leq$ second end-point" for the conventional intervals, for example, saves many of the conditional statements, which can not be escaped implementing directed interval arithmetic. The normalization procedure, which is part of the evaluation of the `Directed` data object, introduces an additional burden of this data type. Since the normalization is not desired for some applications, a next version of this package may involve it as an optional single function. Showing numerous applications and advantages over conventional intervals, an effective usage of the directed interval arithmetic will require its inclusion in the kernel of *Mathematica*.

3.2 Usage

A demonstrative *Mathematica* notebook `directed.ma` illustrates how the new data object and the functions are defined in the *Mathematica* package `directed.m` work and how to use them.

The following command loads the package from the directory `user`.

```
In[6] := <<user\directed.m
```

All the functions contained in `directed.m` are now available for use.

On-line help for obtaining information about all the functions provided by the package `directed.m` can be received using the `?` operator.

```
In[7] := ?Directed
Out[7] = Directed[{a1, b1, Round}, {a2, b2, Round}, ... ] is a data object
   that represents a directed multi interval. Round is an optional
   parameter specifying outward rounding for the corresponding
   interval element. Basic arithmetic on directed intervals is
   automatic. See also: Direction, Sign, First, Second, Min, Max,
   Abs, Proper, Dual, IntMemberQ, IntInteriorQ, IntHull,
   IntIntersection, InclusionQ, InclusionEQ.
```

3.2.1 Data Type Directed

Directed[{a, b}]	**is the directed interval from a to b.**
Directed[{a_1, b_1, Round}, {a_2, b_2, Round}, . . .]	**is the set of outwardly rounded directed intervals a_1 to b_1, a_2 to b_2, ... called directed multi-interval.**

Conventional (proper) intervals supported also by the kernel function `Interval` are a special case of the directed intervals. This is the directed interval from 1 to 4.

```
In[8] := Directed[{1, 4}]
Out[8] = Directed[{1, 4}]
```

First end-point of an improper directed interval is greater than the second end-point.

```
In[9] := Directed[{3/4, 1/4}]
Out[9] = Directed[{3/4, 1/4}]
```

A directed interval may be composed from a single number. Such an interval is called degenerated.

```
In[10] := Directed[4]
Out[10] = Directed[{4, 4}]
```

With inexact numbers, outward rounding is used to ensure that the interval always contains the "true" value.

```
In[11] := Directed[0.]
Out[11] = Directed[{-(2.225073858507201*10^-308),
       2.225073858507201*10^-308}]
```

For non-degenerate intervals an optional parameter `Round` should be specified to ensure outward directed rounding of the end-points.

```
In[12] := a = Directed[{2.3, 4, Round}]
Out[12] = Directed[{2.299999999999999, 4}]
```

First end-points of `a` and `b` differ since `a` is rounded but `b` is not.

```
In[13] := b = Directed[{2.3, 4}]
Out[13] = Directed[{2.3, 4}]
In[14] := a[[1, 1]] - b[[1, 1]]
Out[14] = -(4.440892098500627*10^-16)
```

Several directed intervals may be part of a single data object `Directed`, called directed multi-interval.

```
In[15] := Directed[{3, 2}, {2.3, 4, Round}]
Out[15] = Directed[{2.299999999999999, 4}, {3, 2}]
```

Intersecting, equally directed elements of a directed multi-interval are merged into single intervals, which are put into normal order.

```
In[16] := Directed[{3, 2}, {0, 7}, 0., {2.3, 4, Round}, {2, -1}]
Out[16] = Directed[{-(2.225073858507201*10^-308), 7}, {3, -1}]
```

Symbolic interval arithmetic is not supported by this package.

```
In[17] := Directed[{3, 2}, 0., {2.3, 4}, {a, b}]
Out[17] = Directed[{3, 2}, 0., {2.3, 4}, {a, b}]
```

Numerical evaluation is used everywhere it is necessary, in particular to decide whether interval elements intersect.

```
In[18] := Directed[{1, Pi}, {E, Sqrt[26]}, {4, 7}]
Out[18] = Directed[{1, 7}]
```

3.2.2 Utility Functions

Several utility functions provide convenient manipulations with directed intervals.

Direction[int]	**returns the direction for every element of the directed multi-interval int: -1 for proper and 1 for improper interval elements.**
Sign[int]	**returns the sign of every element of the directed multi-interval int: -1 for negative, 1 for positive and 0 for neither positive nor negative interval elements.**
First[int]	**delivers the first end-point of every element of the directed interval**
Second[int]	**delivers the second end-point of every element of the directed multi-interval int.**
Min[int]	**delivers the greatest lower bound of the directed multi-interval int.**
Max[int]	**delivers the least upper bound of the directed multi-interval int.**
Abs[int]	**delivers the absolute value of the directed multi-interval int.**
Proper[int]	**delivers the corresponding proper projection of the directed multi-interval int.**
MidPoint[int]	**delivers the mid-point of every element of the directed multi-interval int.**

A list of directions is returned for a directed multi-interval.

```
In[19] := Direction[Directed[{-Infinity, -1}, {3, -Infinity}]]
Out[19] = {1, -1}
```

Numerical approximation and normalization are performed first.

```
In[20] := Direction[a = Directed[{3, 2}, {E, Sqrt[2]}]]
Out[20] = -1
In[21] := a
Out[21] = Directed[{3, 2^(1/2)}]
```

Zero is returned as sign of a directed interval consisting of positive and negative numbers.

```
In[22] := Sign[Directed[{-1, 2}]]
Out[22] = 0
```

Note, that `Directed` puts all elements of a directed multi-interval into a normal order which may not correspond to the order they have been entered.

```
In[23] := Sign[a = Directed[{-1/3, 2}, {0, 5.7, Round}, {-2/3, -1}]]
Out[23] = {-1, 0}
In[24] := a
Out[24] = Directed[{-2/3, -1}, {-1/3, 5.700000000000001}]
```

List data structure is also used for representing directed intervals.

```
In[25] := FullForm[Directed[{-1, 2}, {4, 7}]]
Out[25] = FullForm[Directed[List[-1, 2], List[4, 7]]]
```

The end-points of a directed interval can be accessed by functions `First` and `Second`

```
In[26] := First[Directed[{-Infinity, -1}]]
Out[26] = -Infinity
In[27] := Second[a = Directed[{4, 7}, {12, Infinity}]]
Out[27] = {7, Infinity}
```

or by referring to the corresponding list elements:

```
In[28] := {a[[1, 2]], a[[2, 2]]}
Out[28] = {7, Infinity}
```

The smallest upper bound of the directed multi-interval `a` is `Infinity`.

```
In[29] := Max[a]
Out[29] = Infinity
```

The smallest number in the next interval is `Sqrt[2]`.

```
In[30] := Min[Directed[{7, Sqrt[2]}, {E, 3}]]
Out[30] = 2^(1/2)
```

Sometimes it might be necessary to speed up numerical computations involving `Min/Max` functions on directed intervals, for example plotting interval functions. If the directed intervals do not involve *Mathematica* constants or exact singletons, the kernel *Mathematica* functions `Min/Max` can be applied to the head of the directed intervals.

```
In[31] := Min @@ a
Out[31] = 4
```

Kernel `Min/Max` functions can be applied to deliver the greatest lower or the least upper bound of every element in a directed multi-interval but only for intervals which do not involve *Mathematica* constants or exact singletons.

```
In[32] := Max /@ List @@ a
Out[32] = {7, Infinity}

In[33] := Max /@ List @@ Directed[{7, Sqrt[2]}, {E, 3}]
Out[33] = {Max[{7, 2^(1/2)}], Max[{3, E}]}
```

To get a bound on the magnitude of the numbers in a directed multi-interval use `Abs` function.

```
In[34] := Abs[Directed[{-5, 3}, {2, -1}]]
Out[34] = 5
```

To obtain the proper projection of a directed multi-interval use the function `Proper` instead of kernel function `Interval`, which will cause double rounding of the end-points.

```
In[35] := Proper[a = Directed[{-6, 0., Round}]]
Out[35] = Directed[{-6, 2.225073858507201*10^-308}]
In[36] := Interval @@ a
Out[36] = Interval[{-6, 4.450147717014403*10^-308}]
```

3.2.3 Lattice Interval Functions

IntHull[int_1, int_2, ...]	**gives the convex hull of the directed intervals int_1, int_2, ...**
IntIntersection[int1, int2, ...]	**gives the intersection of the directed intervals int_1, int_2, ...**

The following delivers the convex hull of two directed intervals.

```
In[37] := IntHull[Directed[{-Infinity, -13}, {E, -1}],
    Directed[{2, Sqrt[2]}]]
Out[37] = Directed[{-Infinity, 2^(1/2)}, {2, 2^(1/2)}]
```

Symbolic elements are not considered.

```
In[38] := IntHull[Directed[{2.3, 4}, {-1, 12}],
    Directed[5.7], Directed[{a, b}]]
Out[38] = Directed[{-1, 12}]
```

The intersection of disjoint proper intervals is an improper directed interval.

```
In[39] := IntIntersection[Directed[{-12, -3}], Directed[{4, 7}]]
Out[39] = Directed[{4, -3}]
```

Here is another intersection of directed intervals.

```
In[40] := IntIntersection[Directed[{-3, Sqrt[7]}, {2.3, -1.1, Round}],
  Directed[{E, Infinity}, {3/2, -2}],
  Directed[{-11, -Infinity}, {7, 2}]]
Out[40] = Directed[{7, -Infinity}]
```

3.2.4 Order Relations

Directed[{a, b}] is contained in Directed[{c, d}] Iff (c <= a and b <= d and {a, b} $\neq$ {c, d}).

InclusionQ[int, x]	delivers True if number x is contained in some element of the directed multi-interval int and False otherwise.
InclusionQ[x, int]	delivers True if every element of the directed multi-interval int is contained in Directed[{x, x}] and False otherwise.
InclusionQ[int_1, int_2]	delivers True if every element of the directed multi-interval int_2 is contained in some element of the directed multi-interval int_1 and False otherwise.
InclusionQ[q_1, ..., q_p]	delivers True if the q_i form an `InclusionQ` sequence, where q_i are directed intervals (and numbers).
InclusionEQ[int, x]	delivers True if number x is contained in or is equal to some element of the directed multi-interval int and False otherwise.
InclusionEQ[x, int]	delivers True if every element of the directed multi-interval int is contained in or is equal to the Directed[{x, x}] and False otherwise.
InclusionEQ[int_1, int_2]	delivers True if every element of the directed multi-interval int_2 is contained in or is equal to some element of the directed multi-interval int_1 and False otherwise.
InclusionEQ[q_1, ..., q_p]	delivers True if the q_i form an `InclusionEQ` sequence, where q_i are directed intervals (and numbers).

Number 5 contains the improper directed interval [7, 2].

```
In[41] := InclusionQ[5, Directed[{7, 2}]]
Out[41] = True
In[42] := InclusionQ[Directed[{7, 2}], 5]
Out[42] = False
```

$\mathsf{int}_1 < \mathsf{int}_2$	yields True if every element in the directed multi-interval int_1 is less than some element in the directed multi-interval int_2 and False otherwise.
$\mathsf{x} < \mathsf{int}$; $\mathsf{int} < \mathsf{x}$	test the same order relation `Less` between number x and directed multi-interval int.
$\mathsf{q}_1 < \cdots < \mathsf{q}_p$	yields True if the q_i form an increasing sequence, where q_i are directed intervals (and numbers).
$\mathsf{int}_1 <= \mathsf{int}_2$	delivers True if every element in the directed multi-interval int_1 is in relation `LessEqual` with some element in the directed multi-interval int_2 and False otherwise.
$\mathsf{x} <= \mathsf{int}$; $\mathsf{int} <= \mathsf{x}$	test order relation `LessEqual` between number x and directed multi-interval int.
$\mathsf{q}_1 <= \cdots <= \mathsf{q}_p$	yields True if the q_i form a non-decreasing sequence, where q_i are directed intervals (and numbers).
$\mathsf{int}_1 > \mathsf{int}_2$	delivers True if every element in the directed multi-interval int_1 is greater than every element in the directed multi-interval int_2 and False otherwise.
$\mathsf{x} > \mathsf{int}$; $\mathsf{int} > \mathsf{x}$	test order relation `Greater` between number x and directed multi-interval int.
$\mathsf{q}_1 > \cdots > \mathsf{q}_p$	yields True if the q_i form a decreasing sequence, where q_i are directed intervals (and numbers).
$\mathsf{int}_1 >= \mathsf{int}_2$	delivers True if every element in the directed multi-interval int_1 is greater than or equal to some element in the directed multi-interval int_2 and False otherwise.
$\mathsf{x} >= \mathsf{int}$; $\mathsf{int} >= \mathsf{x}$	test order relation `GreaterEqual` between number x and directed multi-interval int.
$\mathsf{q}_1 >= \cdots >= \mathsf{q}_p$	yields True if the q_i form a non-increasing sequence, where q_i are directed intervals (and numbers).

Let us define two directed intervals.

```
In[43] := (a = Directed[{1, 4}, 5, {7, 12}];
    b = Directed[{8, 9}, {10, 4}];)
```

Numerical approximation is used when required.

```
In[44] := InclusionEQ[a, E]
Out[44] = True
```

Directed interval `b` is contained in the directed interval `a`.

```
In[45] := InclusionQ[a, b]
Out[45] = True
```

A sequence of directed intervals and numbers may be tested for inclusion.

```
In[46] := InclusionQ[a, b, E]
Out[46] = True
```

For multi-intervals the implication $A \subseteq B \implies B \subseteq A$ does not always hold.

```
In[47] := InclusionEQ[a = Directed[{-7, -12}, {4, 13}],
      b = Directed[{4, 5}, {6, 7}] ]
Out[47] = True

In[48] := InclusionEQ[b, a]
Out[48] = False
```

Directed intervals `a` and `b` are not in relation "less".

```
In[49] := a < b
Out[49] = False
```

$[1, 4]$ is equal to the first element of the directed interval `a`, so the inequality is `True`.

```
In[50] := Directed[{1, 4}] <= a
Out[50] = True
```

but ...

```
In[51] := Directed[{1, 4}] >= a
Out[51] = False
```

3.2.5 Set Theoretical Interval Functions

IntMemberQ[int, x]	**delivers True if the number x is member of some element of the directed multi-interval int and False otherwise.**
IntMemberQ[int_1, int_2]	**delivers True if the set of values defined by every element of the directed multi-interval int_2 is contained in the set of values defined by some element of the directed multi-interval int_1 and False otherwise.**
IntInteriorQ[int, x]	**delivers True if the number x is in the interior of some element of the directed multi-interval int and False otherwise.**
IntInteriorQ[int_1, int_2]	**delivers True if the set of values defined by every element of the directed multi-interval int_2 is in the interior of the set of values defined by some element of the directed multi-interval int_1 and False otherwise.**

[12, 3] contains 3 and 5 as set theoretical members but not according to the extended inclusion relation.

```
In[52] := IntMemberQ[Directed[{12, 3}], 3]
Out[52] = True
In[53] := IntInteriorQ[Directed[{12, 3}], 5]
Out[53] = True
```

As for the other interval functions numerical approximation is used to make the appropriate conclusion.

```
In[54] := IntMemberQ[Directed[{Sin[4], Sin[2]}], 0.]
Out[54] = True
In[55] := IntInteriorQ[Directed[{Sin[4], Sin[2]}],
 Directed[{1, 2}]]
Out[55] = False
```

3.2.6 Computations

Basic arithmetic on directed multi-intervals is automatic.

`Dual` is an important operation which reverses the end-points of a directed interval.

```
In[56] := Dual[a]
Out[56] = Directed[{4, 1}, {5, 5}, {12, 7}]
```

Instead of Dual you may apply kernel function Reverse to a directed multi-interval.

```
In[57] := Reverse /@ a
Out[57] = Directed[{4, 1}, {5, 5}, {12, 7}]
```

You can do arithmetic with multi-intervals.

```
In[58] := a - Dual[b]
Out[58] = Directed[{-9, 8}, {-3, -4}]
```

Arithmetic with a mixture of numbers and directed intervals also works.

```
In[59] := Directed[{2, 3}] + 5.7
Out[59] = Directed[{7.7, 8.7}]
```

With inexact numbers, directed rounding is used to ensure that the directed interval always contains the "true" value.

```
In[60] := Directed[3.] - 3
Out[60] = Directed[{-(4.440892098500628*10^-16),
      4.440892098500628*10^-16}]
```

Division by a directed interval with 0 in its interior results in two semi-infinite directed intervals.

```
In[61] := 1 / Directed[{3, -2}]
Out[61] = Directed[{-1/2, -Infinity}, {Infinity, 1/3}]
```

Power function produces no exceeding points.

```
In[62] := Directed[{-2, 4}]^(4)
Out[62] = Directed[{0, 256}]
```

- Dual[a] is the additive inverse of the directed interval a.

```
In[63] := a = Directed[{3.1, -2, Round}];

In[64] := a - Dual[a]
Out[64] = 0

In[65] := a - Reverse /@ a
Out[65] = 0

In[66] := a -= Dual[a]
Out[66] = 0
```

`1/Dual[b]` is the multiplicative inverse of the directed interval `b`.

```
In[67] := b = Directed[{-7.3, -1, Round}];

In[68] := b / Dual[b]
Out[68] = 1

In[69] := b / Reverse /@ b
Out[69] = 1

In[70] := b /= Dual[b]
Out[70] = 1
```

Mathematica function `N` converts all numbers to `Real` form and `N[expr, n]` performs computations with n-digit precision numbers. By analogy with the *Mathematica* function `N` we have defined function `R` to produce correctly rounded numerical values for directed intervals.

```
In[71] := Directed[{Sqrt[7], E}] // N
Out[71] = Directed[{2.645751311064591, 2.718281828459045}]
```

Sometimes applying function `R` is more convenient than using the parameter `Round` when we specify a directed interval.

```
In[72] := a = Directed[{2.1, 3.6}];
In[73] := SameQ[Directed[{2.1, 3.6, Round}], R[a]]
Out[73] = True
```

Arithmetic operations and functions on directed intervals are implemented in the *Mathematica* package `directed.m` according to the strict definitions of outwardly rounded computer operations (Section 2.3) providing that the resulting directed interval always encloses (according to the extended inclusion relation) the true result. Sometimes, an inner inclusion of the true interval solution can be very useful giving an estimation of the tightedness of the obtained outer interval solution. An inner inclusion interval is an interval which is contained in the true solution interval. Inner inclusions in conventional interval arithmetic can be obtained only if inwardly rounded interval operations and functions are implemented in addition to the outwardly rounded ones. This will require an extension of the set of operation symbols and will introduce some inconvenience for the users. An other important property of the directed interval arithmetic, showed in Section 2.3 is that inner inclusions can be obtained only by outwardly rounded operations and corresponding conjugation of the input interval expression.

To obtain an outer inclusion for the expression $a \times (b+c) - d$ we need an inward rounding of d and outward rounding of a, b, c and the arithmetic operations. Let a, b, c and d be the directed intervals

```
In[74] := {a, b, c, d} = {Directed[{11., 9.}], Directed[{4., 2.}],
 Directed[{2.3, 1.5}], Directed[{-7.5, -3.1}]};
```

```
In[75] := sOuter = R[a] (R[b] + R[c]) - R[Dual[d]]
Out[75] = Directed[{76.79999999999996, 34.60000000000002}]

In[76] := sInner = Dual[ R[Dual[a]] (R[Dual[b]] + R[Dual[c]]) - R[d] ]
Out[76] = Directed[{76.80000000000005, 34.59999999999999}]

In[77] := InclusionQ[sOuter, sInner]
Out[77] = True
```

3.3 Comparison to the built-in Interval Data Type

In order to clarify the usage of *Mathematica* functions related to conventional and directed intervals we shall make comparison of the behaviour of `Interval` and `Directed`. The user should be aware of some aspects of the implementation of `Interval` that will help a correct its usage. In this section we share our experience in using *Mathematica* for interval computations.

`directed.m` contains several functions that are specific for manipulation with directed intervals. These are the utility functions `Direction, Sign, First, Second, Proper` and function `Dual` implementing the corresponding monadic operation on directed intervals.

The function `Abs` maps `Interval` to `Interval` and `Directed` to a numerical value.

```
In[78] := Abs[Interval[{-11, -7}, {2, 3}]]
Out[78] = Interval[{0, 11}]

In[79] := Abs[Directed[{-11, -7}, {2, 3}]]
Out[79] = 11
```

Kernel function `IntervalUnion` provides set theoretical union of intervals (merging only intersecting intervals) and thus does not fulfill the definition of the interval lattice operation convex hull of intervals.

```
In[80] := IntervalUnion[Interval[{2, 4}], Interval[{3, 7}]]
Out[80] = Interval[{2, 7}]

In[81] := IntervalUnion[Interval[{2, 3}], Interval[{5, 7}]]
Out[81] = Interval[{2, 3}, {5, 7}]

In[82] := IntHull[Directed[{2, 3}], Directed[{5, 7}]]
Out[82] = Directed[{2, 7}]
```

In Section 3.1 we pointed out how important the numerical approximation is for the implementation of interval functions in a computer algebra system. Indeed, numerical approximation is used in most of the functions related to the `Interval` data object except for `Abs` function and comparison operations $<$, $<=$, $>$, $>=$.

```
In[83] := Abs[a = Interval[{Sin[4], Cos[2]}, {Sqrt[2], E}]]
Out[83] = Interval[{Abs[Cos[2]], Abs[Sin[4]]}, {Sqrt[2], E}]

In[84] := Interval[{-Infinity, -100}] < a
Out[84] = -100 < Sin[4]

In[85] := Interval[{-Infinity, -100}] >= a
Out[85] = -Infinity >= E
```

while

```
In[86] := Directed[{-Infinity, -100}] >= Directed @@ a
Out[86] = False
```

The lattice operations convex hull and intersection provide set theoretical functionality for conventional (proper) intervals but not for directed intervals. That is why instead of functions `IntervalUnion` and `IntervalIntersection` performing lattice operations on `Interval` data objects we have defined functions `IntHull` and `IntIntersection` to perform lattice operations on directed intervals. Functions `IntMemberQ` and `IntInteriorQ` were defined to provide reflexive, resp. antireflexive set theoretical functionality for directed intervals too.

In order to clarify the next explanations let us remind the definition of an order relation. A relation $\leq$ in a set M is called an order relation and $\{M, \leq\}$ an ordered set if for $a, b, c \in M$

$$\begin{array}{rcll} & & a \leq a & \text{reflexivity} \\ a \leq b \wedge b \leq c & \Longrightarrow & a \leq c & \text{transitivity} \\ a \leq b \wedge b \leq a & \Longrightarrow & a = b & \text{antisymmetry.} \end{array}$$

In an ordered set $\{M, \leq\}$ an antireflexive ordering is defined by $a < b \;:\Leftrightarrow\; (a \leq b \wedge a \neq b)$, for $a, b \in M$.

`IntervalMemberQ` tests the reflexive inclusion order relation between `Interval` data objects. `directed.m` contains two functions `InclusionQ` and `InclusionEQ` testing the antireflexive, and reflexive inclusion order relation between directed intervals.

```
In[87] := IntervalMemberQ[a = Interval[{1, 3}, 4, {7, 12}],
     Interval[{7, 10}]]
Out[87] = True
In[88] := Inclusion[Directed[{-Infinity, -100}],
 Directed[{Sin[4], Sin[2]}, {Sqrt[2], E}]]
Out[88] = False
```

`InclusionQ` and `InclusionEQ` functions were extended to test the corresponding relation between a sequence of directed intervals and/or numbers.

```
In[89] := InclusionQ[Directed[{1, 3}, {7, 12}], 2.3,
     Directed[{7, 1}]]
Out[89] = True
```

It is possible to test antireflexive inclusion combining `IntervalMemberQ` and `UnsameQ` for single `Interval` operands

```
In[90] := IntervalMemberQ[Interval[{7, 10}], Interval[{8, 9}]] &&
   UnsameQ[Interval[{7, 10}], Interval[{8, 9}]]
Out[90] = True
```

but not for multi-`Interval` operands.

```
In[91] := (a = Interval[{2, 3.}, {7, 12}];
    b = Interval[{2, 3}, {7, 12}];)
In[92] := IntervalMemberQ[a, b] && UnsameQ[a, b]
Out[92] = True
In[93] := InclusionQ[Directed @@ a, Directed @@ b]
Out[93] = False
```

Unfortunately, comparison operations $<=$, $<$, $>=$, $>$ for `Interval` data objects are not implemented according to the corresponding definition for reflexive, and antireflexive order relation.

```
In[94] := Interval[{1, 3}] < Interval[{3, 7}]
Out[94] = False
```

The corresponding antisymmetry property also can not be inferred from this implementation.

```
In[95] := Interval[{1, 3}] <= Interval[{1, 3}]
Out[95] = False
In[96] := Interval[{1, 3}] >= Interval[{1, 3}]
Out[96] = False
```

A definition for order relation between (directed) intervals saying "int_1 is in the corresponding relation with int_2 if every element of int_1 is in the corresponding relation with *every* element of int_2" fulfill the three definition properties for order relation only for single intervals, not for multi-intervals. To make the corresponding definitions of $<=$, $<$, $>=$, $>$ correct for multi-intervals too, we have implemented (in contrast to `Interval` predicates) a definition saying "int_1 is in the corresponding relation with int_2 if every element of int_1 is in the corresponding relation with *some* element of int_2". Obviously, a precise study of the arithmetic structure involving (directed) multi-intervals is required.

Kernel function `Interval` supports the symbol `Indeterminate`, resulting indeterminate numerical computations.

```
In[97] := a = Interval[{3, Infinity}] - Infinity
   Infinity::indet:
      Indeterminate expression -Infinity + Infinity
encountered.
Out[97] = Interval[{Indeterminate, -Infinity}]
```

Kernel interval operations and functions deal with intervals involving the symbol `Indeterminate` but they reverse the end-points of the interval involving `Indeterminate`

```
In[98] := a + Interval[{5, 7}]
Out[98] = Interval[{-Infinity, Indeterminate}]
```

and sometimes produce strange results.

```
In[99] := IntervalMemberQ[a, -Infinity]
Out[99] = False

In[100] := IntervalMemberQ[Out[ ], -Infinity]
Out[100] = True

In[101] := IntervalIntersection[a, Interval[{2, 3}]]
Out[101] = Interval[{-Infinity, Infinity}, {Indeterminate, Infinity}]
```

Exceptions are again the relational operations.

```
In[102] := Interval[{2, 3}] < a
Out[102] = 3 < Indeterminate
```

Directed interval arithmetic in *Mathematica* follows a model that provides consistency between interval arithmetic and IEEE floating-point arithmetic.

```
In[103] := Directed[{2, 3}] < Directed @@ a
Out[103] = False

In[104] := IntMemberQ[Directed @@ a, -Infinity]
Out[104] = True

In[105] := IntIntersection[Directed @@ a, Directed[{2, 3}]]
Out[105] = Interval[{Indeterminate, 3}]
```

A *Mathematica* notebook `indeterm.ma` contains a case study of the kernel functions on intervals involving the symbol `Indeterminate` and a detailed demonstration of the directed interval arithmetic on intervals involving the symbol `Indeterminate`.

Finally, implementing interval algorithms in computer algebra systems one have to keep in mind that not all simplification rules valid for numerical vectors and matrices are valid for interval ones too. The following example illustrates this.

```
In[106] := a = {-3, 1, 2};
In[107] := b = {Interval[{5, 7}], Interval[{5, 7}], Interval[{5, 7}]};
In[108] := a . b
Out[108] = 0
In[109] := a . Apply[Directed, b, 1]
Out[109] = 0
```

Multiplying the numerical vector `a` and the interval vector `b` *Mathematica* takes `Interval[{5, 7}]` or `Directed[{5, 7}]` out of brackets which leads to a wrong result. Fortunately, multiplication of two interval vectors is performed correctly.

```
In[110] := Interval /@ a . b
Out[110] = Interval[{-6, 6}]
In[111] := Directed /@ a . Apply[Directed, b, 1]
Out[111] = Directed[{-6, 6}]
```

The application of `Dot` function with numerical and interval arguments also should be avoided since

```
In[112] := Dot[{-3, 1, 2}, Table[Directed[{5, 7}], {3}]]
Out[112] = 0
```

Function `Inner` can be used instead.

```
In[113] := Inner[{-3, 1, 2}, Table[Directed[{5, 7}], {3}]]
Out[113] = Directed[{-6, 6}]
```

Obtaining an incorrect result from the *Mathematica* function `InterpolatingPolynomial` on the interval segments `segm` is due to the above reason.

```
In[114] := InterpolatingPolynomial[
  segm = Table[{x, Interval[{-1/2, 1/2}]}, {x, 4}], x]
Out[114] = Interval[{-1/2, 1/2}]

In[115] := InterpolatingPolynomial[
  Table[{x, Interval[{-1/2, 1/2}] + x/10000}, {x, 4}], x]
Out[115] = (-1 + x)*(Interval[{-9999/10000, 10001/10000}] +
      (-2 + x)*(Interval[{-1, 1}] +
  (-3 + x)*Interval[{-2/3, 2/3}])) +
    Interval[{-4999/10000, 5001/10000}]
```

4 Applications

Although developed in the late seventies [13], interval arithmetic structure $\{D, +, \times, \subseteq\}$ has remained an isolated mathematical theory for long. Only Gardenes et al. have tried to use it in solving some practical problems ([7, 8]). At the same time another extension of the conventional interval arithmetic have been proposed. Markov [21] extended the set of arithmetic operations on conventional (proper) intervals by four nonstandard (inner) interval operations. A number of numerical algorithms with result verification have been developed showing the advantages of the inner interval operations for a tighter inclusion of functional ranges. Inner operations have been also used for defining differential and integral calculus for interval functions [22].

One of the reasons for so little usage of the extension considered by Kaucher is that there has not been found an interpretation of the improper intervals and the corresponding extended arithmetic in terms of the normal interval arithmetic space. A recent work [2] and other papers by S. Markov [25, 26] found such a interpretation and showed a close relation between the two extensions of the conventional interval arithmetic — one obtained by extending the set of intervals, and the other obtained by extending the set of operations. The technique developed for "translation" of the results for directed intervals in the "language" of normal intervals has the advantage that we can easily perform computations in $\{D, +, \times, \subseteq\}$, where the operations possess group properties and good compatibility between each other, and interpret the results in the space IR of normal intervals. Directed interval arithmetic facilitates the extension of the calculus for interval-valued functions of real arguments [24] to the case of interval arguments.

In the next subsections we shall outline some applications of the directed interval arithmetic. Several *Mathematica* functions solving certain algebraic and numerical problems involving directed intervals were developed to illustrate these applications and to show how a user can build his own functions that use directed interval arithmetic. All the *Mathematica* functions presented in the next subsections were designed and work only for single directed intervals. All they require that the package `directed.m` has been initially loaded.

4.1 Directed Ranges

An interpretation of the elements of D as "directed" ranges of monotone functions [23] leads to an important application of the extended interval arithmetic providing sharp bounds for the range of monotone functions. Let f be a continuous and monotone function on the interval $T \in IR$ and its range be $f(T) = \{f(t) \mid t \in T\}$. The type of monotonicity of f determines the "direction" into which the range $f(T)$ is run through when the argument t runs through its interval domain T in a fixed direction, say from left to right. Indeed, if f is isotone (nondecreasing) in T, then the interval $f(t)$ is run through from its lower to its upper bound whenever t runs through T from left to right. Alternatively $f(T)$ is run through from its upper to its lower bound if f is antitone (nonincreasing) in T and t runs through T from left to right. That is why the interval $f[T] = [f(t^-), f(t^+)] \in D$ is called *directed range* of the function f over the interval T. The binary variable $\tau(A)$, defined by

(1), called direction of the interval $A \in D$, represents the type of the monotonicity of the corresponding monotone function which directed range is A.

Consider the function $f(x) = f_1(x) \circ f_2(x)$, where $\circ \in \{+, -, \times, /\}$. We seek the range of the function using the already known ranges $f_1(X)$, $f_2(X)$. Since f_1 and f_2 are continuous on R then the ranges $f_1(X)$ and $f_2(X)$ are intervals and for the range of f we have $f(X) = \{f_1(x) \circ f_2(x) \mid x \in X\} \subseteq f_1(X) \circ f_2(X)$. It is highly desirable to obtain an equality in the above relation. However, such a equality relation could be achieved by the conventional interval arithmetic only if both functions are equally monotone on the interval X. The familiar interval arithmetic can not provide an exact expression for $f(X)$ if f_1 and f_2 have different monotonicity on X. In [21] Markov defines an extension of the conventional interval arithmetic by introducing four special interval operations which provide equality relation for differently monotone functions. An analogue proposition in terms of directed intervals says:

Proposition *Assume $CM(T)$ is the set of continuous and monotone functions defined on $T \in IR$. For $f, g,\ f \circ g \in CM(D)$, $X \subseteq D$, $\ \circ \in \{+, -, \times, /\}$ and $g[X] \in D \setminus \mathcal{T}$ for $\circ = /$, we have*

$$(f \circ g)[X] \;=\; \begin{cases} f[X] + g[X], \\ f[X] - g[X]_-, \\ f[X]_{\sigma(g[X])} \ \times\ g[X]_{\sigma(f[X])}, \\ f[X]_{\sigma(g[X])} \ /\ g[X]_{-\sigma(f[X])}. \end{cases}$$

This proposition gives the direction of the resulting interval, and therefore supplies additional information for the type of monotonicity of the result. Based on this proposition, an algorithm for automatic computation of inner and outer inclusions of ranges of functions and their derivatives can be developed and implemented as a separate *Mathematica* package.

4.2 Solving Interval Algebraic Equations

Between the most important properties of the directed interval arithmetic is the existence of inverse elements with respect to the operations addition and multiplication (see property **K.4**, Section 2.2). For every $A \in D$ the corresponding inverse elements `-Dual[A]`, for the addition, and `1/Dual[A]`, for the multiplication, are such that

$$A - \mathrm{Dual}[A] = 0 \qquad \text{and} \qquad A/\mathrm{Dual}[A] = 1. \tag{11}$$

This property allows automatic simplification of interval expressions involving directed intervals. Doing computations with exact numbers, property **K.4** (11) will lead to the exact results 0 or 1. When A is an inexact number, however, the result of (11) will be a small interval around 0, resp. 1, which will blow up after each subsequent arithmetic operation. On the other hand, computer algebra systems like *Mathematica* allow that relations between abstract data objects be implemented as corresponding simplification equations. Thus, property (11) of the directed interval arithmetic was implemented in the *Mathematica* package `directed.m` by the following simplification equations for the operations addition and multiplication.

```
Directed /: Directed[{a_, b_}] + Directed[{c_, d_}] := 0 /;
    a == -c && b == -d ;
```

```
Directed /: Times[Directed[{a_, b_}], Directed[{c_, d_}]] := 1 /;
    a == 1/c && b == 1/d ;
```

Due to these equations exact algebraic simplification are possible even on interval expressions involving inexact numbers.

```
In[116] := a = Directed[{N[E], N[Pi]}];

In[117] := a - Dual[a]
Out[117] = 0
```

while

```
In[118] := b = -1 Dual[a];
   Directed[{First[a] + First[b], Second[a] + Second[b], Round}]
Out[118]   Directed[{-(4.440892098500628*10^-16), 4.440892098500628*10^-16}]
```

Due to the existence of inverse elements the solutions of the interval algebraic equations

$$\begin{array}{rcll} A + X & = & B & \qquad \text{and} \\ U \times Y & = & V & \end{array}$$

can be expressed in terms of the operations between directed intervals: $X = B - \mathtt{Dual}[A]$ and $Y = V/\mathtt{Dual}[U]$, if $U \in D \setminus \mathcal{T}$.

Mathematica function `Solve[eqns, vars]` attempts to solve an equation or set of equations for the variables `vars`. The rules that are built into *Mathematica* are intended to be appropriate for the broadest range of calculations. In specific cases, as directed or conventional interval arithmetic, some of the built-in rules are not valid.

```
In[119] := Solve[Interval[{2, 3}] + x == 0, x]
Out[119] = {{x -> Interval[{-3, -2}]}}

In[120] := Interval[{2, 3}] + x /. Out[119][[1]]
Out[120] = Interval[{-1, 1}]
```

Next *Mathematica* code illustrates how one can give his own transformation rules for a built-in *Mathematica* function to work correctly on directed intervals, too.

```
Solve::indet = "Solution is not defined. Zero in some divisor.";
Unprotect[Solve];
Solve[a_.  + b_.  x_Symbol ==  c_,  x_Symbol]:={} /;
     IntMemberQ[Directed[b], 0] && (Message[Solve::indet]; True);
Solve[a_. + b_. x_Symbol ==  c_, x_Symbol]:=
     {x -> (c - Dual[a]) / Dual[b]} /;
     Apply[And, (Head[#] == Directed || NumberQ[N[#]])& /@ {a, b, c}];
Protect[Solve];
```

Now, `Solve` can give the correct solution to a linear algebraic equation involving directed intervals

```
In[121] := Solve[Directed[{2, 3}]  - E x == Directed[{Sqrt[7], 12}], x]
Out[121] = {x -> Directed[{-9/E, -((-2 + 7^(1/2))/E)}]}

In[122] := Directed[{2, 3}]  - E x /. Out[121]
Out[122] = Directed[{7^(1/2), 12}]
```

or to warn that can not find the solution

```
In[123] := Solve[Directed[{2, 3}] - Directed[{-1, 1}] x == 0, x]
       Solve::indet:
    Solution is not defined. Zero in some divisor.
Out[123] = {}
```

When inexact numbers are involved in the directed intervals, `Solve` produces an outer inclusion of the corresponding solution.

```
In[124] := Solve[Directed[{5.1, 3, Round}] -
   Dual[Directed[{7.2, 11.5, Round}]] y == Directed[{1, 2}], y]
Out[124] = {y -> Directed[{0.0869565217391304, 0.5694444444444448}]}

In[125] := Directed[{5.1, 3, Round}] -
   Dual[Directed[{7.2, 11.5, Round}]] y /. Out[124]
Out[125] = Directed[{0.999999999999995, 2.000000000000001}]

In[126] := InclusionQ[Out[125], Directed[{1, 2}]]
Out[126] = True
```

An inner approximation of the solution can be obtained by solving the corresponding dual problem according to the inclusion assertions given in Section 2.3.

Linear algebraic equations are not the only equations that can be solved straightforward in directed interval arithmetic. We shall consider a simple practical example taken from [7] to illustrate the solution of a class of nonlinear algebraic equations.

Problem. Let $v = e \times r/(\rho + r + s)$ be the equation describing an electrical circuit, where e, r and ρ are given constants varying in prescribed intervals: $e \subseteq E$, $r \subseteq R$, $\rho \subseteq R_0$, $E, R, R_0 \in IR$. The goal is to determine an interval value S for the resistance, so that for any $s \in S$ the voltage v to be kept in prescribed bounds $V \in IR$, that is $v \subseteq V$.

We must solve S from the equation

$$\frac{E \times R}{R + R_0 + S} = V. \tag{12}$$

In directed interval arithmetic the solution S^* of (12) may be a proper interval showing the tolerance for the electrical circuit or an improper interval, that is a control band.

As in the linear case, the algebraic solution of (12) can be found by applying algebraic transformations on the equation and using properties (11) of the inverse additive and multiplicative elements. *Mathematica* function `Solve` was overloaded to find the solution of nonlinear algebraic equations of the form $a + b/(c + d \times x) = e$ involving directed intervals.

```
Unprotect[Solve];
Solve[a_. + b_. / (c_. + d_. x_Symbol) == e_, x_Symbol] := {} /;
     (IntMemberQ[Directed[d], 0] || IntMemberQ[Directed[e - Dual[a]], 0]) &&
     (Message[Solve::indet]; True);

Solve[a_. + b_. / (c_. + d_. x_Symbol) == e_, x_Symbol] :=
     {x -> Dual[(b/(Dual[e] - a) - c) / d]} /;
     Apply[And, (Head[#] == Directed || NumberQ[N[#]])& /@ {a, b, c, d, e}];
Protect[Solve];
```

Let we are given the following bounds for the input constants

```
In[127] := {e, r, r0} =
   {Directed[{2, 4}], Directed[{9, 11}], Directed[{1.5, 2.5}]};
```

and the voltage has to be kept inside the interval $[2.0, 4.0]$

```
In[128] := v = Directed[{2, 4}];
```

We have to compute an inner approximation $S^* \subseteq S$ in order to guarantee the inclusion $E \times R/(R + R_0 + S^*) \subseteq V$ (interval operations are inclusion isotone). According to the inclusion assertions from Section 2.3 we have to solve the the corresponding dual problem

```
In[129] := Solve[Dual[e] Dual[r] / (Dual[r] + R[Dual[r0]] + s) ==
 Dual[v], s] /. {Directed[t_] -> Dual[Directed[t]]}
Out[129] = {s -> Directed[{7.499999999999998, 2.500000000000003}]}
```

Since the resulting interval is improper, this is a control interval. In other words for any $s \in S$ there exist $e \in E$, $\rho \in R_0$ and $r \in R$ such that $e * r/(\rho + r + s) = v \in V$ or $E \times R/(R + R_0 + \mathtt{Dual}[S]) \supseteq V$. We can verify it by

```
In[130] := InclusionEQ[e r / (r + R[r0] + Dual[s]) /. Out[129], v]
Out[130] = True

In[131] := InclusionQ[v, e r / (r + R[r0] + s) /. Out[129]]
Out[131] = True
```

Allowing the voltage within the wider range $[2, 8]$

```
In[132] := v = Directed[{2, 8}];
```

the resulting resistance would be

```
In[133] := Solve[Dual[e] Dual[r] / (Dual[r] + R[Dual[r0]] + s) ==
 Dual[v], s] /. {Directed[t_] -> Dual[Directed[t]]}
Out[133] = {s -> Directed[{1.999999999999999, 2.500000000000003}]}
```

showing the tolerance for the electrical circuit, that is for any $s \in S$ and any $e \in E$, $\rho \in R_0$ and $r \in R$ $e * r/(\rho + r + s) = v \in V$ or $E \times R/(R + R_0 + S) \subseteq V$.

```
In[134] := InclusionQ[v, e r / (r + R[r0] + s) /. Out[133]]
Out[134] = True
```

4.3 Linear Algebraic Systems

Initiated by Gardenes et al. [7, 8] the algebraic completeness of the extended interval arithmetic structure $\{D, +, \times, \subseteq\}$ has been used by some Russian authors [19, 34, 36] to develop an *algebraic approach* to the solution of several problems related to interval linear algebraic systems.

Consider the system of linear algebraic equations

$$\mathbf{A}x = \mathbf{b}, \tag{13}$$

involving intervals in the $n \times n$-matrix $\mathbf{A}$ and/or in the right-hand side n-dimensional vector $\mathbf{b}$. Four different solution sets to the system (13) have been defined and studied:

- the *united solution set* is the set of solutions of all real (point) systems $Ax = b$ with $A \in \mathbf{A}$ and $b \in \mathbf{b}$, i. e.,

$$\begin{aligned} \Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b}) &= \{x \in R^n \mid (\exists A \in \mathbf{A})(\exists\, b \in \mathbf{b})(Ax = b)\} \\ &= \{\, x \in R^n \mid \mathbf{A}x \cap \mathbf{b} \neq \emptyset\,\}; \end{aligned} \tag{14}$$

- the *tolerable solution set* is the set of all real vectors x such that for every real $A \in \mathbf{A}$ the real vector Ax is contained in the interval vector $\mathbf{b}$, that is

$$\begin{aligned} \Sigma_{\forall\exists}(\mathbf{A}, \mathbf{b}) &= \{\, x \in R^n \mid (\forall A \in \mathbf{A})(\exists\, b \in \mathbf{b})(Ax = b)\,\} \\ &= \{\, x \in R^n \mid (\forall A \in \mathbf{A})(Ax \in \mathbf{b})\} = \{\, x \in R^n \mid \mathbf{A}x \subseteq \mathbf{b}\}; \end{aligned} \tag{15}$$

- the *controlled solution set* is the set of all real vectors $x \in R^n$, such that for any point $b \in \mathbf{b}$ we can find the corresponding $A \in \mathbf{A}$ satisfying $Ax = b$;

$$\begin{aligned} \Sigma_{\exists\forall}(\mathbf{A}, \mathbf{b}) &= \{\, x \in R^n \mid (\exists\, A \in \mathbf{A})(\forall\, b \in \mathbf{b})(Ax = b)\,\} \\ &= \{\, x \in R^n \mid (\forall\, b \in \mathbf{b})(\mathbf{A}x \ni b) = \{x \in R^n \mid \mathbf{A}x \supseteq \mathbf{b}\}; \end{aligned} \tag{16}$$

- the *interval algebraic solution* is an interval vector $\mathbf{x_a}$ which substituted in the expression $\mathbf{A} \times \mathbf{x}$, performing all interval operations, results in $\mathbf{b}$, that is

$$\mathbf{A}\mathbf{x_a} = \mathbf{b}. \tag{17}$$

The detailed description of each of the solution sets (14)–(16) is rather complex and grows exponentially with n. This is the reason why usually a simpler subset satisfying the definition of the corresponding solution set is looked for. Such a simpler solution set can be expressed in terms of intervals and the task then becomes: "Find an interval vector that is included in the corresponding solution set (if nonempty) of the interval linear system", or shortly, find an inner approximation for the corresponding solution set.

A key role in finding an inner approximation for the solution sets (14)–(16) plays the interval algebraic solution. This solution does not exists in general in the ordinary interval space and the directed interval arithmetic is a natural arithmetic for dealing with interval algebraic equations, since it is obtained from the arithmetic for normal intervals via algebraic completion.

It has been proved [19, 34] that if $\mathbf{x}_a$ is an algebraic interval solution to the system $\mathbf{A}x = \mathbf{B}_-$ in $\mathcal{K}$ and all its components are improper intervals, then $(\mathbf{x}_a)_- \subseteq \sum_{\exists\exists}$. That is $(\mathbf{x}_a)_-$ is an inner approximation of the united solution set of (13). Let $\mathbf{x}_a$ be the interval algebraic solution of (13) obtained in $\mathcal{K}$. If all components of $\mathbf{x}_a$ are proper intervals, then $\mathbf{x}_a \subseteq \sum_{\forall\exists}$; if all components of $\mathbf{x}_a$ are improper intervals, then $(\mathbf{x}_a)_- \subseteq \sum_{\exists\forall}$ [34]. The algebraic approach is proved to give almost always the inclusion-maximal inner estimations of the considered solution sets.

Shary [34] has investigated some properties of the algebraic interval solutions to the system (13) and proposed a numerical method for computation of $\mathbf{x}_a$. Constructing his method Shary has used the properties of the directed interval arithmetic and a bijective mapping from D^n into R^{2n} seeking thus the solution of the equation in R^{2n}, not in D^n. His numerical algorithm has nothing to do with directed interval arithmetic, possibly due to the lack of convenient tools for computation with directed intervals.

In [19] Kupriyanova has investigated the properties of the algebraic interval solution to the interval linear algebraic system $\mathrm{Dual}(\mathbf{A})x = \mathbf{b}$, where Dual performs the corresponding unary operation on the elements of $\mathbf{A}$. It is proven that the algebraic solution to this system is a maximal inner estimation for the united solution set of (13) in the sense of inclusion. An iterative two-sided method, originally proposed in [36] for the united solution set of interval linear systems in IR, has been generalized for interval linear systems in D. It is proven that the operator G, defined by

$$(Gx)_i \;:=\; \left(\mathbf{b}_i - \sum_{k=1,k\neq i}^{n} (a_{ik}x_k)_-\right) / (a_{ii})_-, \qquad i = 1,\ldots,n$$

is a continuous antitone operator that has at least one fixed point. If v_0 is an initial approximation of $\mathbf{x}_a$ such that $v_0 \subseteq \mathbf{x}_a = G\mathbf{x}_a$, then because of the antitonicity $v_0 \subseteq \mathbf{x}_a \subseteq Gv_0 = w_0$ and

$$v_0 \subseteq v_1 \subseteq \ldots v^* \subseteq \mathbf{x}_a \subseteq w^* \subseteq \ldots w_1 \subseteq w_0.$$

Each v_k is an inner estimation and each w_k is an outer estimation for the interval algebraic solution $\mathbf{x}_a$ of (13).

The proposed two-sided iterative algorithm was implemented in the *Mathematica* function `AlgebraicIntervalSolve` with directed interval arithmetic from the package `directed.m`.

```
AlgebraicIntervalSolve::diag = "Diagonal elements of the matrix
  contain zero. Rearrange the matrix.";

AlgebraicIntervalSolve::nosol = "Non contractive mapping.
  Try better initial approximation.";

Options[AlgebraicIntervalSolve] = {
 InitialApproximation -> Automatic,
 AccuracyGoal -> 10.^-6,
 IterativeList -> False,
 MaxIterations -> $IterationLimit};
```

```
AlgebraicIntervalSolve[m_, b_, opts___] := Module[
  {n = Length[b], operator, twostep, stopping,
   lst = IterativeList /. {opts} /. Options[AlgebraicIntervalSolve],
   eps = AccuracyGoal /. {opts} /. Options[AlgebraicIntervalSolve],
   it = MaxIterations /. {opts} /. Options[AlgebraicIntervalSolve],
   v = InitialApproximation /. {opts} /. Options[AlgebraicIntervalSolve],
  operator[x_] := ((b[[#]] - Dual[Delete[m[[#]], #] .
    Delete[x, #]]) / Dual[m[[#, #]]])& /@ Range[n];
  twostep[y_] := Module[{t = operator[y[[2]]]},
   it -= 1;  {t, operator[t]}];
  stopping[x_] := Max[Abs[N[Flatten[Apply[List,
    x[[1]] - Dual /@ x[[2]], 1]]]]] < eps;
  If[v == Automatic, v = Directed /@ LinearSolve[Map[MidPoint, m, {2}],
    MidPoint /@ b] ];
  If[lst, FixedPointList[twostep, {v, operator[v]},
    SameTest -> ((it < 0 || stopping[#])&)],
   FixedPoint[twostep, {v, operator[v]},
      SameTest -> ((it < 0 || stopping[#])&)]] /;
    Inner[InclusionEQ, operator[operator[v]], v, And] ||
    (Message[AlgebraicIntervalSolve::nosol]; False)
  ] /; Not[Apply[Or, IntMemberQ[m[[#, #]], 0]& /@ Range[Length[m]] ]] ||
     (Message[AlgebraicIntervalSolve::diag]; False) ;
```

`AlgebraicIntervalSolve[m_, b_, opts___]` gives inner and outer approximations of the algebraic interval solution to a given linear algebraic system with matrix `m` and right-hand-side vector `b` of directed intervals. In order to get started `AlgebraicIntervalSolve` has to be given an appropriate initial inner approximation of the algebraic interval solution. `AlgebraicIntervalSolve` automatically starts iterations from the solution to the point linear system `MidPoint`$(\mathbf{m})\mathbf{x} = \mathbf{b}$. This is a good initial approximation if the algebraic interval solution of the system `Dual`$(\mathbf{m})\mathbf{x} = \mathbf{b}$ is sought. If is possible to give the `AlgebraicIntervalSolve` another initial approximation of the algebraic interval solution by using the option `InitialApproximation`. If the inclusion $v_0 \subseteq Gw_0$ is fulfilled then the initial approximation is good and the operator G is a contractive one.

Several options can be also used to control the number of iterations performed by the function `AlgebraicIntervalSolve`. First, one can set `MaxIterations` to specify the maximum number of iterations that `AlgebraicIntervalSolve` should use. If `AlgebraicIntervalSolve` does not find a good solution in the number of steps that have been specified, it returns the last values that have been computed. These values can be used as `InitialApproximation` if one needs to continue the iteration. To control the accuracy of the solution `AlgebraicIntervalSolve` iterates and checks whether the differences in the end-points of the interval components between two successive approximations of the algebraic solution are within the accuracy specified by the option `AccuracyGoal`. For the purpose of debugging one can ask `AlgebraicIntervalSolve` to give the whole list of approximations obtained at each step of the iterative process by using the option `IterativeList`. Table 1 lists the options for `AlgebraicIntervalSolve` and their default values.

option name	*default value*	*function*
InitialApproximation	Automatic	approximate solution to the system using mid-points of the directed input intervals of matrix `m` and vector `b`
AccuracyGoal	10^-6	accuracy to which two successive approximations differ
MaxIterations	\$IterationLimit	maximum number of iterations to be performed in finding approximations to the algebraic solution
IterationList	False	delivers the list of approximations obtained at each step of the iterative process

Table 1. Options for `AlgebraicIntervalSolve`.

Let us find the algebraic interval solution to the system given by the matrix `A` and right-hand side vector `B`. Note, that the first vector is the inner approximation and the second vector is the outer approximation to the "true" solution.

```
In[135] := (A = {{Directed[{2, 4}], Directed[{-1, 1}]},
  {Directed[{-1, 1}], Directed[{2, 4}]}};
    B = {Directed[{0, 2}], Directed[{0, 2}]};)

In[136] := N[AlgebraicIntervalSolve[A, B]]
Out[136] = {{Directed[{0.200000007947286, 0.399999996026357}],
     Directed[{0.200000007947286, 0.399999996026357}]},
    {Directed[{0.1999999980131785, 0.4000000009934108}],
     Directed[{0.1999999980131785, 0.4000000009934108}]}}
```

The algebraic interval solution of the system with matrix `Dual(A)` and vector `B` is the maximum inner estimation of the united solution set of `Ax = B`. The united solution set, it maximum inner estimation and the algebraic solution of the linear algebraic system with matrix `A` and right-hand side vector `B` are presented on Fig. 1.

```
In[137] := AlgebraicIntervalSolve[Map[Dual, A, {2}], B]
Out[137] = {{Directed[{0, 1}], Directed[{0, 1}]},
    {Directed[{0, 1}], Directed[{0, 1}]}}
```

Now, let us find the algebraic interval solution of the system with matrix `A` and vector `G`.

```
In[138] := G = {Directed[{-3, 3}], Directed[{0, 0}]};
In[139] := AlgebraicIntervalSolve[A, G]
```

```
Out[139] = AlgebraicIntervalSolve::nosol:
    Non contractive mapping.
    Try better initial approximation.

   AlgebraicIntervalSolve[{{Directed[{2, 4}], Directed[{-1, 1}]},
      {Directed[{-1, 1}], Directed[{2, 4}]}},
     {Directed[{-3, 3}], Directed[{0, 0}]}]
```

Since we know that there exists an algebraic interval solution of this system, let us try another initial approximation of the algebraic solution.

```
In[140] := AlgebraicIntervalSolve[A, G, InitialApproximation ->
{Directed[{-0.3, 0.3}], Directed[{0.5, -0.5}]}]
Out[140] = {{Directed[{-3/4, 3/4}], Directed[{3/8, -3/8}]},
     {Directed[{-3/4, 3/4}], Directed[{3/8, -3/8}]}}
```

The algebraic interval solution of the algebraic interval system `Ax = G` does not belong to the set of proper intervals. Let us find an inner estimation of the united solution set of the same system. The united solution set and the algebraic solution to the linear algebraic system with matrix `A` and right-hand side vector `G` are presented on Fig. 2.

```
In[141] := AlgebraicIntervalSolve[Map[Dual, A, {2}], G]

Out[141] = {{Directed[{-3/2, 3/2}], Directed[{0, 0}]},
     {Directed[{-3/2, 3/2}], Directed[{0, 0}]}}
```

Out[141] shows that the solution of the point linear system `MidPoint[A]x = MidPoint[G]` may not be a good initial approximation of the algebraic interval solution. The following proposition [27] gives the exact solution of a linear algebraic system with a special type of point matrix.

Proposition *Let $A = (a_{i,k}) \in R^{n\times n}$ be a real matrix and let the numbers $a_{i,k}\Delta_{i,k}$, where $\Delta_{i,k}$ is the subdeterminant of $a_{i,k}$, have constant signs for all $i, k = 1, 2, ..., n$. Then for the solution of $A\mathbf{x} = \mathbf{b}$ the following Cramer-type formula holds:*

$$(\mathbf{x}_i)_{\sigma(\Delta)} = \frac{1}{\Delta}\sum_{i=1}^{n}(-1)^{i+k}\Delta_{ik}(\mathbf{b}_i)_{\lambda_{i,k}} \overset{Def}{=} \frac{1}{\Delta}\begin{vmatrix} a_{11} & \dots & \mathbf{b}_1 & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{1n} & \dots & \mathbf{b}_n & \dots & a_{nn} \end{vmatrix}, \tag{18}$$

where $\lambda_{i,k} = (-)^{i+k} = \{+,\ i+k \text{ even};\ -,\ i+k \text{ odd}\}$.

Mathematica function `DirectedCramer` implements formula (18) for directed intervals.

```
DirectedCramer::dim = "Not equal dimensions.";

DirectedCramer::nosol = "Zero Determinant.";

DirectedCramer[m_, b_] := Module[{n = Length[b], d, t},
  (Table[Inner[Times,
```

```
   Times[Det[Delete[Delete[m, Table[{k, i}, {k, n}]],
  #]]& /@ Range[n], t = Table[(-1)^(i+k), {k, n}]],
   MapAt[Dual, b, Flatten[{Position[t, -1]}]]], {i, n}]
   / d /. {x__} :> Dual /@ {x} /; Sign[d] == -1 ) /;
  UnsameQ[d = Det[m], 0] || (Message[DirectedCramer::nosol]; False)
] /; SameQ[Table[Length[b], {Length[b]}], Dimensions[m]] ||
     (Message[DirectedCramer::dim]; False);
```

A separate function `CamerTypeQ` tests the requirements of the formula.

```
CramerTypeQ[m_] := Sign[m[[1, 1]] m[[2, 2]]] == Sign[m[[1, 2]] m[[2, 1]]
 ] /; Length[m] == 2 && SameQ @@ Dimensions[m] ;

CramerTypeQ[m_] := With[{n = Length[m]},
     SameQ @@ (Sign /@ Thread[Times[
       Flatten[Table[m[[n-i+1, n-j+1]], {i, n}, {j, n}]],
       Flatten[Minors[m, n-1]] ]])
] /; SameQ @@ Dimensions[m] ;
```

A class of matrices satisfying the conditions of the above proposition is the class of Vandermonde matrices, appearing in interpolation theory. This makes the above formula suitable for the exact solution of identification problems in an interval interpolation setting [24].

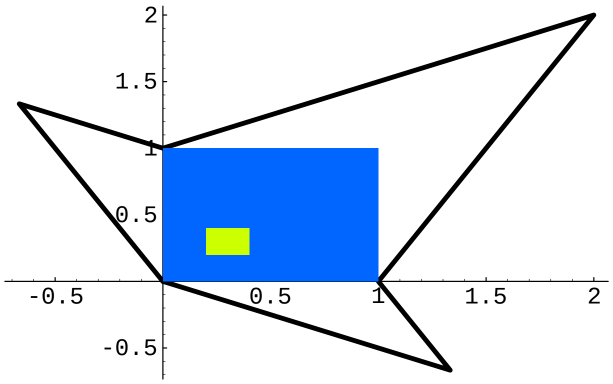

Figure 1: The united solution set (black solid line), its inner approximation (black rectangle) and the algebraic solution set (small grey rectangle inside) for the algebraic system with interval matrix A and interval vector B.

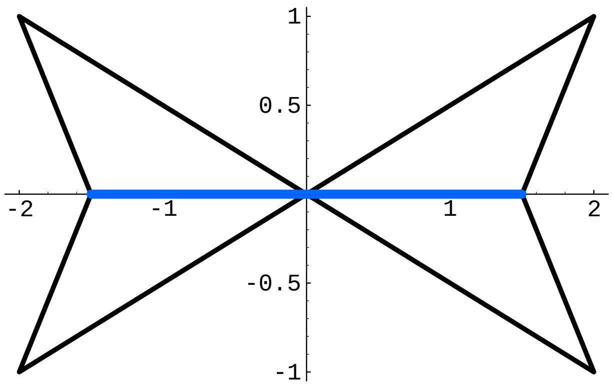

Figure 2: The united solution set (black solid line) and its inner approximation (grey section on the abscissa line) for the algebraic system with interval matrix A and interval vector G.

References

[1] Alefeld, G.; Herzberger, J.: *Einführung in die Intervallrechnung.* Bibliographisches Institut Mannheim, 1974.

[2] Dimitrova, N.; Markov, S. M.; Popova, E.: *Extended Interval Arithmetics: New Results and Applications.* In Atanassova, L.; Herzberger, J. (Eds.): *Computer Arithmetic and Enclosure Methods.* Elsevier Sci. Publishers B. V., 1992, pp. 225–232.

[3] Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., Watt, S. M.: *The Maple V language reference manual.* Springer-Verlag, New York, 1991.

[4] Collins, G. E.; Krandick, W.: *A Hybrid Method for High Precision Calculation of Polynomial Real Roots.* In Bronstein, M. (Ed.): Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, ACM Press, 1993, pp. 47-52.

[5] Connell, A. E.; Corless, R. M.: *An Experimental Interval Arithmetic Package in Maple.* Interval Computations, No. 2, 1993, pp. 120-134.

[6] Durst, E.: *Realisierung einer erweiterten Intervallrechnung mit Überlaufarithmetik.* Diplomarbeit, Universität Karlsruhe, 1975.

[7] Gardeñes, E.; Trepat, A.: *Fundamentals of SIGLA, an Interval Computing System over the Completed Set of Intervals.* Computing, **24**, 1980, pp. 161–179.

[8] Gardeñes, E.; Trepat, A.; Janer, J. M.: *Approaches to Simulation and to the Linear Problem in the SIGLA System.* Freiburger Interval-Berichte 81/8, 1981, pp. 1–28.

[9] Hong, H.: *Topology Analysis of Plane Algebraic Curves by Hybrid Methods: Numeric, Interval and Algebraic.* International Conference on Interval and Computer-Algebraic Methods in Science and Engineering. INTERVAL'94, St.-Petersburg, 1994, pp. 109-114.

[10] Hyvönen, E., DePascale, S.: *Complement and Discontinuous Interval Arithmetic.* Int. Conference on Interval and Computer-Algebraic Methods in Science and Engineering, Interval'94, St-Petersburg, 1994.

[11] Johnson, J. R., Krandick, W.: *A multiprecision floating point and interval arithmetic package for symbolic computation.* Technical Report RISC-Linz Report series number 93-20, research Institute for Symbolic Computation, RISC-Linz, Johanes Kepler University, A-4040 Linz, Austria, 1993.

[12] Kahan, W. M.: *A More Complete Interval Arithmetic.* Lecture Notes for a Summer Course at the University of Michigan, 1968.

[13] Kaucher, E.: *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Räume.* Dissertation, Universität Karlsruhe, 1973.

[14] Kaucher, E.: Algebraische Erweiterungen der Intervallrechnung unter Erhaltung der Ordnungs und Verbandstrukturen, Computing, Suppl. 1, 65–79, 1977.

[15] Kaucher, E.: *Über Eigenschaften und Anwendungsmöglichkeiten der erweiterten Intervallrechnung und des hyperbolischen Fastkörpers über R.* Computing Suppl. **1**, 1977, pp. 81–94.

[16] Kaucher, E.: *Interval Analysis in the Extended Interval Space IR.* Computing Suppl. **2**, 1980, pp. 33–49.

[17] Keiper, J. B.: *Interval Arithmetic in* Mathematica. Interval Computations, No. 3, 1993, pp. 76-87.

[18] Kulisch, U., Miranker, W. L.: *Computer Arithmetic in Theory and Practice.* Academic Press, New York 1981.

[19] Kupriyanova, L.: *A Method of Square Root for Solving Interval Linear Algebraic System.* Reliable Computing, 1(1), 1995, pp. 15-31.

[20] Laveuve, S. E.: *Definition einer Kahan-Arithmetik und ihre Implementierung.* In Nickel, K. (Ed.): *Interval Mathematics.* Lecture Notes in Computer Science, **29**, Springer, Berlin, 1975, pp. 236–245.

[21] Markov, S. M.: *Extended Interval Arithmetic.* Compt. Rend. Acad. Bulg. Sci., **30**, 9, 1977, pp. 1239–1242.

[22] Markov, S. M.: *Extended interval arithmetic and differential and integral calculus for interval functions of a real variable.* Ann. Univ. Sofia, Fac. Math., 71, Part I, 1976/77, pp. 131-168 (In Bulgarian).

[23] Markov, S. M.: *On the Presentation of Ranges of Monotone Functions using Interval Arithmetic.* Interval Computations, No 4(6), 1992, pp. 19–31.

[24] Markov, S. M.: *Some Interpolation Problems Involving Interval Data.* Interval Computations, 3, 1993, pp. 164-182.

[25] Markov, S. M.: *On Directed Interval Arithmetic and its Applications.* J. UCS, **1**, 7, 1995, pp. 510-521, (www server: http://hgiicm.tu-graz.ac.at/Cjucs_root).

[26] Markov, S. M.: *On the Foundations of Interval Arithmetic.* Proc. of SCAN-95, Alefeld, G. and Frommer, A. (Eds.): Akademie Verlag, Berlin pp. 507–513.

[27] Markov, S.; Popova, E.; Ullrich, Ch.: *On the Solution of Linear Algebraic Equations Involving Interval Coefficients.* In: S. Margenov, P. Vassilevski (Eds.): *Iterative Methods in Linear Algebra, II*, IMACS Series in Computational and Applied Mathematics, Vol. 3, 1996, pp. 216–225.

[28] Moore, R. E.: *Interval Analysis.* Prentice Hall, Englewood Cliffs, N. J., 1966.

[29] Ng, E.: *Symbolic-Numeric Interface: a Review.* Lecture Notes in Computer Science, **72**, 1979, pp. 330–345.

[30] Ortolf, H. -J.: *Eine Verallgemeinerung der Intervallarithmetik.* Geselschaft für Mathematik und Datenverarbeitung, Bonn **11**, 1969, pp. 1–71.

[31] Popova, E.: *Interval Operations Involving NaNs.* Reliable Computing, **2**, 2, 1996, pp. 161–166.

[32] Popova, E. D.; Ullrich, C. P.: *A Generalization of BIAS Specification.* Technical Report 95-8, Universität Basel, Dezember 1995, pp. 1-16.

[33] Semenov, A. L.; Petunin, D. V.: *The Use of Multi-Intervals in the UniCalc Solver.* IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, Wuppertal, September 26-29, 1995.

[34] Shary, S. P.: *Algebraic Approach to the Interval Linear Static Identification, Tolerance and Control Problems or One More Application of Kaucher Arithmetic.* Reliable Computing, **2**, 1, 1996, pp. 1–32.

[35] Wolfram, S.: Mathematica *A System for Doing Mathematics by Computer.* Addison-Wesley, 1991 (2nd ed.).

[36] Zyuzin, V. S.: *On a Way of Finding Two-Sided Approximation to the Solution of System of Linear Interval Equations.* Differential Equations and the Theory of Functions, Saratov State University, Saratov, 1987, pp. 28–32. (in Russian)