Chapter 2

Interval arithmetic

In this chapter, we will briefly describe the fundamentals of interval arithmetic. We will also discuss how to implement the arithmetic in a programming environment.

Simply put, interval arithmetic is an arithmetic for *inequalities*. To illustrate this point, let us assume that we want to compute the area of a rectangle with side-lengths ℓ_1 and ℓ_2. Given the measurements $\ell_1 = 10.3 \pm 0.1$ and $\ell_2 = 4.4 \pm 0.2$, what can we say about the area $A = \ell_1 \cdot \ell_2$? If we express our measurements in terms of the bounds $|\ell_1 - 10.3| \leq 0.1$ and $|\ell_2 - 4.4| \leq 0.2$, then (using the triangle inequality) all we can say is that $|\ell_1 \cdot \ell_2 - 10.3 \cdot 4.4| \leq 0.2 \cdot 10.3 + 0.1 \cdot 4.4 + 0.1 \cdot 0.2$, i.e., $|A - 45.32| \leq 2.52$. If, on the other hand, we view the measurements as the inequalities $10.2 \leq \ell_1 \leq 10.4$ and $4.2 \leq \ell_2 \leq 4.6$, the optimal answer is obvious: the area must satisfy $42.84 = 10.2 \cdot 4.2 \leq A \leq 10.4 \cdot 4.6 = 47.84$, which translates into the slightly improved bound $|A - 45.34| \leq 2.5$.

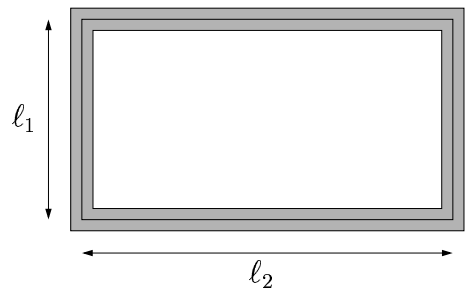

Figure 2.1: A rectangle with sides ℓ_1 and ℓ_2.

The calculations in the latter case can be summarized as a single multiplication of two intervals:

$$[10.2, 10.4] \times [4.2, 4.6] = [42.84, 47.84].$$

Interval arithmetic justifies this extension of the real arithmetic, and provides an elegant means of computing with inequalities. For a concise reference on this topic,

see e.g. [Mo66], [Mo79], or [AH83]. Early references are [Yo31], [Dw51], [Wa56], [Su58] and [Mo59].

2.1 Real intervals

In what follows, our basic elements will be closed and bounded intervals of the real line. We will adopt the short-hand notation

$$[a] = [\underline{a}, \bar{a}] = \{x \in \mathbb{R}\colon\ \underline{a} \leq x \leq \bar{a}\},$$

and consider the set of all such intervals of the real line:

$$\mathbb{IR} = \{[\underline{a}, \bar{a}]\colon\ \underline{a} \leq \bar{a};\quad \underline{a}, \bar{a} \in \mathbb{R}\}.$$

Note that we allow for degenerate intervals $[a]$ with $\underline{a} = \bar{a}$. We will refer to these intervals as being *thin*. A natural embedding of $\mathbb{IR}$ in $\mathbb{R}^2$ is given by the mapping $g\colon \mathbb{IR} \to \mathbb{R}^2$, defined by $[\underline{a}, \bar{a}] \mapsto (\underline{a}, \bar{a})$. Geometrically, this corresponds to viewing $\mathbb{IR}$ as the region in $\mathbb{R}^2$ above and on the diagonal $y = x$. Points in $\mathbb{R}^2$ lying on the diagonal correspond to thin intervals.

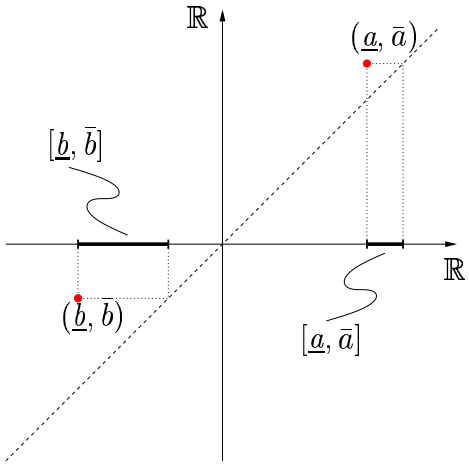

Figure 2.2: Identifying $\mathbb{IR}$ with $\{(x, y) \in \mathbb{R}^2 : y \geq x\}$.

Example 2.1.1 *The elements* $[-3, 4]$, $[1, 1]$, *and* $[\pi, 7]$ *all belong to* $\mathbb{IR}$, *whereas* $[2, -1]$ *and* $[-\infty, 0]$ *do not.*

Being sets, the elements of $\mathbb{IR}$ inherit the natural set relations, such as $=$, $\subseteq$, $\subset$, and $\overset{\circ}{\subset}$, defined by

$$\begin{aligned}
[a] = [b] \quad &\Leftrightarrow \quad \underline{a} = \underline{b} \text{ and } \bar{a} = \bar{b} \\
[a] \subseteq [b] \quad &\Leftrightarrow \quad \underline{b} \leq \underline{a} \text{ and } \bar{a} \leq \bar{b} \\
[a] \subset [b] \quad &\Leftrightarrow \quad [a] \subseteq [b] \text{ and } [a] \neq [b] \\
[a] \overset{\circ}{\subset} [b] \quad &\Leftrightarrow \quad \underline{b} < \underline{a} \text{ and } \bar{a} < \bar{b}
\end{aligned}$$

We can partially order[1] the set $\mathbb{IR}$ in several ways. Emphasizing the set-valued properties of $\mathbb{IR}$, we can use $\subseteq$ as a partial ordering. Preserving the the natural ordering of the real numbers, we may extend the relation $\leq$ to mean

$$[a] \leq [b] \quad \Leftrightarrow \quad \bar{a} \leq \underline{b}.$$

This also provides a partial ordering of $\mathbb{IR}$.

By somewhat abusing our interval notation, we often identify a real number a with the corresponding thin interval $[a, a]$. It then makes sense to define the relation

$$a \in [b] \quad \Leftrightarrow \quad \underline{b} \leq a \text{ and } a \leq \bar{b},$$

which is really a special case of $[a] \subseteq [b]$. All of these relations can be complemented by their logical opposites $\neq$, $\not\subseteq$, $\not\subset$, $\not\subset^{\circ}$, and $\notin$.

We can also equip $\mathbb{IR}$ with analogues to the set operations $\cup$ and $\cap$. Both operations, however, require minor adjustments. First, taking the union of two intervals may not result in a new interval. To overcome this problem, we introduce the notion of forming the *hull* of two intervals:

$$[a] \sqcup [b] = [\min\{\underline{a}, \underline{b}\}, \max\{\bar{a}, \bar{b}\}].$$

It is clear that the resulting interval contains the union of $[a]$ and $[b]$. Second, the intersection of two intervals $[a]$ and $[b]$ is empty if either $\bar{a} < \underline{b}$ or $\bar{b} < \underline{a}$. Because of this, we must add the empty set (denoted by $[\emptyset]$) to $\mathbb{IR}$ for the intersection operator to be well-defined. When the intervals $[a]$ and $[b]$ have at least one point in common, the intersection is the standard one. Thus we have

$$[a] \cap [b] = \begin{cases} [\emptyset] & : \text{ if } \bar{a} < \underline{b} \text{ or } \bar{b} < \underline{a}, \\ [\max\{\underline{a}, \underline{b}\}, \min\{\bar{a}, \bar{b}\}] & : \text{ otherwise.} \end{cases}$$

Example 2.1.2 *Let* $[a] = [1, 3]$, $[b] = [1, \pi]$, $[c] = [-2.3, 4]$, *and* $[d] = [4, 5]$. *Then* $[a] \subset^{\circ} [c]$, $[a] \subset [b]$, $[a] \sqcup [b] = [1, \pi]$, $[a] \sqcup [d] = [1, 5]$, $[a] \cap [d] = [\emptyset]$, *and* $[c] \cap [d] = [4, 4]$.

Given an interval $[a] \in \mathbb{IR}$, we define the following real-valued functions

$$\begin{aligned} \operatorname{rad}([a]) &= \tfrac{1}{2}(\bar{a} - \underline{a}) \qquad (\text{the } \textit{radius} \text{ of } [a]), \\ \operatorname{mid}([a]) &= \tfrac{1}{2}(\bar{a} + \underline{a}) \qquad (\text{the } \textit{midpoint} \text{ of } [a]). \end{aligned}$$

Thus we can write $[a] = [\operatorname{mid}([a]) - \operatorname{rad}([a]), \operatorname{mid}([a]) + \operatorname{rad}([a])]$, and it follows that

$$\xi \in [x] \quad \Leftrightarrow \quad |\xi - \operatorname{mid}([x])| \leq \operatorname{rad}([x]),$$

[1] A relation $\sim$ is a *partial order* on a set S if, for all $a, b, c \in S$, it satisfies: (1) Reflexivity: $a \sim a$. (2) Antisymmetry: $a \sim b$ and $b \sim a$ implies $a = b$. (3) Transitivity: $a \sim b$ and $b \sim c$ implies $a \sim c$.

for any interval $[x]$. Two additional real-valued functions which often come in handy are

$$\begin{aligned} \mathrm{mig}([a]) &= \min\{|a| : a \in [a]\} \qquad (\text{the } \textit{mignitude} \text{ of } [a]), \\ \mathrm{mag}([a]) &= \max\{|a| : a \in [a]\} \qquad (\text{the } \textit{magnitude} \text{ of } [a]). \end{aligned}$$

These functions provide us with the smallest resp. largest distance to the origin attained by elements of $[a]$. There are explicit, computable formulas for these functions:

$$\mathrm{mig}([a]) = \begin{cases} 0 & : \text{ if } 0 \in [a], \\ \min\{|\underline{a}|, |\bar{a}|\} & : \text{ otherwise;} \end{cases} \qquad \mathrm{mag}([a]) = \max\{|\underline{a}|, |\bar{a}|\}.$$

Combining the two functions, we can form the *absolute value* of an interval:

$$\mathrm{abs}([a]) = \{|a| : a \in [a]\} = [\mathrm{mig}([a]), \mathrm{mag}([a])].$$

In contrast to the previously defined functions, the absolute value of an interval is an interval.

Example 2.1.3 *Let* $[x] = [-2, 3]$ *and* $[y] = [1, \pi]$. *Then* $\mathrm{mag}([x]) = 3$, $\mathrm{mig}([x]) = 0$, $\mathrm{mag}([y]) = \pi$, $\mathrm{mig}([y]) = 1$, $\mathrm{abs}([x]) = [0, 3]$, *and* $\mathrm{abs}([y]) = [1, \pi]$.

Finally, we can turn $\mathbb{IR}$ into a metric space[2] by equipping it with the Hausdorff distance:

$$d([a], [b]) = \max\{|\underline{a} - \underline{b}|, |\bar{a} - \bar{b}|\}. \tag{2.1}$$

Note that, according to our definitions, it follows that $d([a], [b]) = 0$ if and only if $[a] = [b]$. Using the metric, we can define the notion of a convergent sequence of intervals:

$$\begin{aligned} \lim_{k\to\infty} [a_k] = [a] \quad &\Leftrightarrow \quad \lim_{k\to\infty} d([a_k], [a]) = 0 \\ &\Leftrightarrow \quad \left(\lim_{k\to\infty} \underline{a}_k = \underline{a}\right) \wedge \left(\lim_{k\to\infty} \bar{a}_k = \bar{a}\right) \wedge \left(\forall k \quad \underline{a}_k \le \bar{a}_k\right). \end{aligned}$$

Note that the last condition is necessary for the $\Leftarrow$ direction.

2.2 Real interval arithmetic

In addition to viewing the elements of $\mathbb{IR}$ as sets, we may consider them as generalized real numbers. As such, it makes sense to attempt to define arithmetic on $\mathbb{IR}$. We have already seen that a copy of $\mathbb{R}$ is represented in $\mathbb{IR}$ as the set of thin intervals. It is therefore desirable to demand that the extended arithmetic should coincide with the normal real arithmetic for thin intervals. The most natural approach is to define binary arithmetic operations on elements of $\mathbb{IR}$ in a set theoretic manner:

[2] See Definition A.3.2 for the definition of a metric space.

Definition 2.2.1 *If $\star$ is one of the operators $+, -, \times, \div$, we define arithmetic on the elements of $\mathbb{IR}$ by*

$$[a] \star [b] = \{a \star b\colon\ a \in [a], b \in [b]\},$$

with the exception that $[a] \div [b]$ is undefined if $0 \in [b]$.

From this definition it is not immediately clear that the resulting set always is an interval. As we are working exclusively with closed intervals, however, it turns out that we can describe the resulting set in terms of the endpoints of the operands:

Proposition 2.2.2 *Arithmetic on the elements of $\mathbb{IR}$ is given by*

$$\begin{aligned}
[a] + [b] &= [\underline{a} + \underline{b}, \bar{a} + \bar{b}] \\
[a] - [b] &= [\underline{a} - \bar{b}, \bar{a} - \underline{b}] \\
[a] \times [b] &= [\min\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\}, \max\{\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}\}] \\
[a] \div [b] &= [a] \times [1/\bar{b}, 1/\underline{b}], \quad \textit{if } 0 \notin [b].
\end{aligned}$$

Proof: The reason why the resulting set is an interval is due to the fact that any *real* operation $+, -, \times, \div$ is continuous in both of its arguments, with the exception of dividing by zero (this is why $[a] \div [b]$ is undefined[3] if $0 \in [b]$). If we fix one of the arguments, the real operations are monotone in the remaining argument. The monotonicity implies that extremal values are attained on the boundary of the domains, i.e., at the endpoints of the intervals. The proposition can thus be verified by examining a finite number of cases. □

As a consequence of Proposition 2.2.2, it follows that $\mathbb{IR}$ is an arithmetically closed subset of $\mathcal{P}(\mathbb{R})$ – the power set[4] of the real numbers.

From a computer programming point of view, this is good news indeed: using the formulas from Proposition 2.2.2, it is straight-forward to implement the datatype `interval` with its associated arithmetic, see Section 2.4. From a practical perspective, the formulas for multiplication and division can be made more efficient. As it stands, a single *interval* multiplication requires four *real* multiplications (as well as several comparisons). This number can be reduced by checking the sign of each endpoint of the two intervals. It is easy to see that interval multiplication can be divided into nine cases, as illustrated in Figure 2.3. Only one case requires four real multiplications; the other cases require just two.

As an example, assume that $0 \leq \underline{a} \leq \bar{a}$ and $\underline{b} \leq 0 \leq \bar{b}$. This situation corresponds to the square on the second row, third column in Figure 2.3. It is clear that the maximal element of $[a] \times [b] = \{a \times b\colon\ a \in [a], b \in [b]\}$ is given by choosing the largest

[3] We can allow for division by zero by extending the underlying set of real numbers to include the concept of infinity. We will address this topic in Section 2.3.

[4] The power set $\mathcal{P}(\mathbb{S})$ of a set $\mathbb{S}$ is the set of all subsets of $\mathbb{S}$.

elements from both $[a]$ and $[b]$. By the same token, the minimal element of $[a] \times [b]$ is given by choosing the largest element from $[a]$ and the smallest element from $[b]$. The resulting interval is thus given by $[a] \times [b] = [\bar{a}\underline{b}, \bar{a}\bar{b}]$, which only requires two real multiplications. In a similar fashion, the formula for interval division can be reduced to six simpler cases.

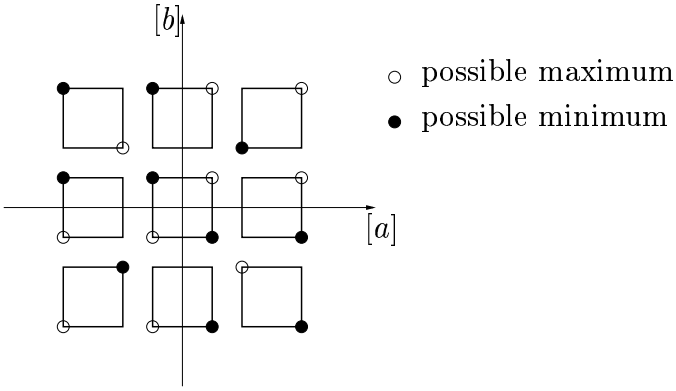

Figure 2.3: A more efficient interval multiplication scheme.

Example 2.2.3 *Using Proposition 2.2.2, we can compute*

$$\begin{aligned} [-1,0]+[0,\pi] &= [-1,\pi] & \qquad [-1,-1]\times[2,5] &= [-5,-2] \\ [1,4]-[1,4] &= [-3,3] & \qquad [-2,3]\times[-2,3] &= [-6,9] \\ [\tfrac{1}{2},1]-[0,\tfrac{1}{6}] &= [\tfrac{1}{3},1] & \qquad [1,\sqrt{2}]\times[-1,1] &= [-\sqrt{2},\sqrt{2}] \\ [2,4]-[3,3] &= [-1,1] & \qquad [1,2]\div[-2,-1] &= [-2,-\tfrac{1}{2}]. \end{aligned}$$

It follows from Definition 2.2.1 (or from Proposition 2.2.2) that addition and multiplication are both associative and commutative: for $[a], [b], [c] \in \mathbb{IR}$, we have

$$\begin{aligned} [a]+([b]+[c]) &= ([a]+[b])+[c]; & \qquad [a]+[b] &= [b]+[a], \\ [a]\times([b]\times[c]) &= ([a]\times[b])\times[c]; & \qquad [a]\times[b] &= [b]\times[a]. \end{aligned}$$

Also, it is clear that the elements $[0,0]$ and $[1,1]$ are the unique neutral elements with respect to addition and multiplication, respectively. Note, however, that in general an element in $\mathbb{IR}$ has no additive or multiplicative inverse. For example, we have $[1,2]-[1,2] = [-1,1] \neq [0,0]$, and $[1,2]\div[1,2] = [\frac{1}{2},2] \neq [1,1]$. As a consequence, the distributive law does *not* always hold. As an example[5], we have

$$[-1,1]([-1,0]+[3,4]) = [-1,1][2,4] = [-4,4],$$

[5]Here, and in what follows, we will often suppress the multiplication symbol $\times$.

whereas

$$[-1,1][-1,0] + [-1,1][3,4] = [-1,1] + [-4,4] = [-5,5].$$

This unusual property is important to keep in mind when representing functions as part of an interval calculation. Interval arithmetic satisfies a weaker rule than the distributive law, which we shall refer to as *sub-distributivity*:

$$[a]([b] + [c]) \subseteq [a][b] + [a][c]. \tag{2.2}$$

This is a set theoretical property that illustrates one of the fundamental differences between real- and interval arithmetic.

Exercise 2.2.4 *Prove that the space* $\mathbb{IR}$ *can be partially ordered by either relation* $\subseteq$ *or* $\leq$*, as described in Section 2.1.*

Exercise 2.2.5 *Prove that* $[a]([b] + [c]) = [a][b] + [a][c]$ *when either*

(1) $[a]$ *is thin.*

(2) all elements of $[b]$ *and* $[c]$ *have the same sign.*

Exercise 2.2.6 *Given an interval* $[a]$ *show that*

(1) $0 \in [a] - [a]$*, but that in general* $[a] - [a] \neq [0,0]$*,*

(2) $1 \in [a] \div [a]$*, but that in general* $[a] \div [a] \neq [1,1]$*. (Assume* $0 \notin [a]$*.)*

Another key feature of interval arithmetic is that of *inclusion isotonicity*:

Theorem 2.2.7 *If* $[a] \subseteq [a']$*,* $[b] \subseteq [b']$*, and* $\star \in \{+, -, \times, \div\}$*, then*

$$[a] \star [b] \subseteq [a'] \star [b'],$$

where we demand that $0 \notin [b']$ *for division.*

This is the single most important property of interval arithmetic: it allows us to accurately estimate the range of a large class of functions. This will be explained in a later section. Note that, in particular, Theorem 2.2.7 holds when $[a]$ and $[b]$ are thin intervals, i.e., real numbers.

Proof: It is somewhat amazing that this powerful theorem has a classical "one-line" proof: by an immediate application of Definition 2.2.1, we have

$$[a] \star [b] = \{a \star b\colon\ a \in [a], b \in [b]\} \subseteq \{a \star b\colon\ a \in [a'], b \in [b']\} = [a'] \star [b'].$$

□

2.3 Extended interval arithmetic

According to Definition 2.2.1, we cannot divide by an interval containing zero. Nevertheless, if we attempt to reinterpret the spirit of the formula

$$[a] \div [b] = \{a \div b\colon\ a \in [a], b \in [b]\},$$

as

$$[a] \div [b] = \{c \in \mathbb{R}\colon\ bc = a,\ a \in [a], b \in [b]\}, \tag{2.3}$$

there might be a way around this slight imperfection. The procedure, however, is quite delicate, and implementing it on a computer raises some subtle questions regarding how we choose to extend the real numbers to include the concept of infinity. Before going into details, let us illustrate the use of (2.3) in a simple setting.

Example 2.3.1 *If $a = [1,2]$ and $b = [-5,3]$, then according to (2.3), the quotient $[c] = [a] \div [b]$ is given by*

$$[c] = \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in [-5,3]\}.$$

Focusing on the particular value $b = 0$, we want to find all c such that $0 \cdot c \in [1,2]$. As the equation clearly has no solution, we loose no information by discarding this case. Hence

$$\begin{aligned}
[c] &= \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in [-5,0) \cup (0,3]\} \\
&= \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in [-5,0)\} \cup \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in (0,3]\} \\
&= \big([1,2] \div [-5,0)\big) \cup \big([1,2] \div (0,3]\big).
\end{aligned}$$

The first set may be interpreted as the limit

$$\begin{aligned}
[c]^- &= \lim_{\varepsilon \to 0^-} \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in [-5,\varepsilon)\} \\
&= \lim_{\varepsilon \to 0^-} [1,2] \div [-5,\varepsilon) = \lim_{\varepsilon \to 0^-} (\tfrac{2}{\varepsilon}, -\tfrac{1}{5}) = (-\infty, -\tfrac{1}{5}].
\end{aligned}$$

Similarly, the second set may be interpreted as the limit

$$\begin{aligned}
[c]^+ &= \lim_{\varepsilon \to 0^+} \{c \in \mathbb{R}\colon\ bc = a,\ a \in [1,2], b \in (\varepsilon,3]\} \\
&= \lim_{\varepsilon \to 0^+} [1,2] \div (\varepsilon,3] = \lim_{\varepsilon \to 0^+} [\tfrac{1}{3}, \tfrac{2}{\varepsilon}) = [\tfrac{1}{3}, \infty).
\end{aligned}$$

Combining the two results, we have the answer

$$[1,2] \div [-5,3] = (-\infty, -\tfrac{1}{5}] \cup [\tfrac{1}{3}, \infty) = \mathbb{R} \setminus (-\tfrac{1}{5}, \tfrac{1}{3}).$$

This example indicates that we need a notion of infinity in order to perform the extended interval division. There are several ways we can allow for "division by zero" – it all boils down to how we choose to extend the real numbers. From a mathematical point of view, there are more or less elegant extensions. We will acquaint ourselves with three variants, which are appropriately named: *the good*, *the bad*, and *the ugly*. Naturally, we shall stick to the ugly.

2.3.1 The good: projective extension

The projective extension of the real numbers, usually denoted $\mathbb{R}^*$, is formed by adding the unsigned "point at infinity" ∞ to the real line. This one-point compactification of the real line allows us to identify $\mathbb{R}^*$ with the closed circle $\mathbb{S}^1$ where the north-pole $\mathbf{N}$ plays the role of infinity, see Figure 2.4.

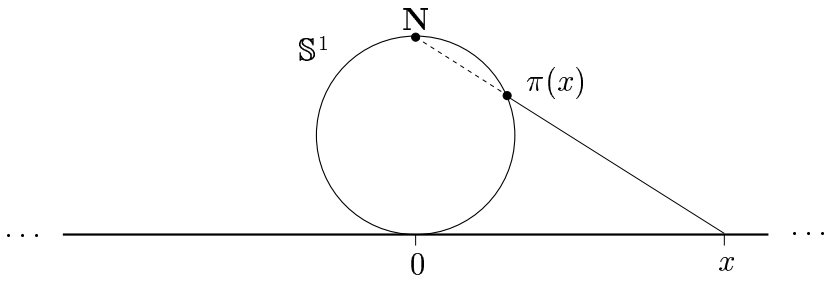

Figure 2.4: Identifying $\mathbb{R}^*$ and $\mathbb{S}^1$ via the projection $\pi\colon \mathbb{R}^* \to \mathbb{S}^1$.

We can partially extend the arithmetic operations from $\mathbb{R}$ to $\mathbb{R}^*$ in the following manner:

$$\begin{aligned} -(\infty) &= \infty, & x + \infty = \infty + x &= \infty \text{ if } x \neq \infty, \\ x \cdot \infty = \infty \cdot x &= \infty \text{ if } x \neq 0, & x/\infty &= 0 \text{ if } x \neq \infty, \\ x/0 &= \infty \text{ if } x \neq 0. \end{aligned}$$

The expressions $\infty \pm \infty$, ∞/∞ and $0 \cdot \infty$, however, are undefined.

In this setting there is no need for the interpretation (2.3). Instead, repeating the division performed in Example 2.3.1, we immediately have

$$\begin{aligned} [1,2] \div [-5,3] &= [1,2] \div \big([-5,0) \cup \{0\} \cup (0,3]\big) \\ &= \big([1,2] \div [-5,0)\big) \cup \big([1,2] \div 0\big) \cup \big([1,2] \div (0,3]\big) \\ &= \{x \in \mathbb{R}\colon x \leq -\tfrac{1}{5}\} \cup \{\infty\} \cup \{x \in \mathbb{R}\colon \tfrac{1}{3} \leq x\}. \end{aligned}$$

Note that we no longer can write $(-\infty, -\frac{1}{5})$ for $\{x \in \mathbb{R}\colon x \leq -\frac{1}{5}\}$. This is because, in the projective extension, we have $-\infty = \infty$, i.e., there is only one infinity, and it cannot be compared to the finite real numbers with any of the relations $\{<, \leq, >, \geq\}$. As a consequence, we cannot assign the value zero to an expression like $e^{-\infty}$, since in $\mathbb{R}^*$ the equality $e^{-\infty} = e^{\infty}$ must hold. On the other hand, it makes perfect sense to write $\tan(\pi/2) = \infty$. The beautiful part of the projective extension is the way we can represent the result of a "division by zero". So far, the outcome of performing $[1,2] \div [-5,3]$ appears to be a rather messy expression. Nevertheless, using the topology of the circle, we may adopt a short-hand notation for *extended intervals*:

$$[\tfrac{1}{3}, -\tfrac{1}{5}] = \{x \in \mathbb{R}\colon x \leq -\tfrac{1}{5}\} \cup \{\infty\} \cup \{x \in \mathbb{R}\colon \tfrac{1}{3} \leq x\}.$$

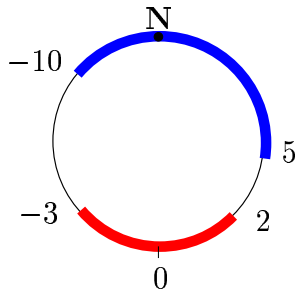

Figure 2.5: The two intervals $[-3, 2]$ and $[5, -10]$ in $\mathbb{R}^*$.

To motivate this notation, we refer to Figure 2.5, which shows two intervals in $\mathbb{R}^*$. The interval containing zero is simply the set $\{x \in \mathbb{R}^* : -3 \leq x \text{ and } x \leq 2\}$, which we denote $[-3, 2]$ as usual. The second interval, however, is different. It represents the set $\{x \in \mathbb{R}^* : x \leq -10 \text{ or } 5 \leq x \text{ or } x = \infty\}$, which we can write as $[5, -10]$. This should be interpreted on the circle as moving from the left endpoint, 5, counter-clockwise to the right endpoint, -10, just as with normal intervals. Unfortunately, there is no suitable way to represent the extended real line $\mathbb{R}^*$ in this manner.

Motivated by the preceeding discussion, we define the set $\mathbb{IR}^*$ of projectively extended intervals:

$$\mathbb{IR}^* = \{[\underline{a}, \bar{a}] : \quad \underline{a}, \bar{a} \in \mathbb{R}^*\},$$

where the case $\underline{a} > \bar{a}$ is interpreted as an extended interval.

2.3.2 The bad: affine extension

The affine extension of the real numbers, usually denoted $\bar{\mathbb{R}}$, is formed by adding the two signed infinities, $-\infty$ and $+\infty$, to the real line. This two-point compactification of the real line allows us to write $\bar{\mathbb{R}}$ as the closed interval $[-\infty, +\infty]$. The arithmetic operations can be partially extended to $\bar{\mathbb{R}}$ in the following manner:

$$\begin{aligned} -(+\infty) = -\infty \text{ and } -(-\infty) = +\infty, \qquad & x + (+\infty) = +\infty \text{ if } x \neq -\infty, \\ x + (-\infty) = -\infty \text{ if } x \neq +\infty, \qquad & x \cdot (\pm\infty) = \pm\infty \text{ if } x > 0, \\ x \cdot (\pm\infty) = \mp\infty \text{ if } x < 0, \qquad & x/(\pm\infty) = 0 \text{ if } x \neq \pm\infty. \end{aligned}$$

The expressions $+\infty + (-\infty)$, $-\infty + (+\infty)$, and $x/0$, however, are undefined. In contrast to the projective extension, the affine infinities can be compared in size: $-\infty < x < +\infty$ if $x \neq \pm\infty$ and $-\infty < +\infty$. Furthermore, the affine extension has the appealing property that it makes perfect sense to write statements like $e^{-\infty} = 0$, $e^{+\infty} = +\infty$, $\ln 0 = -\infty$, and $\ln(+\infty) = +\infty$. This property makes the affine extension the preferred choice among analysists.

It is now straight-forward to define the set $\mathbb{I}\bar{\mathbb{R}}$ of affinely extended intervals:

$$\mathbb{I}\bar{\mathbb{R}} = \{[\underline{a}, \bar{a}] : \underline{a} \leq \bar{a}; \quad \underline{a}, \bar{a} \in \bar{\mathbb{R}}\}.$$

Thus, apart from the elements of $\mathbb{IR}$, also intervals on the form $[-\infty, x]$, $[x, +\infty]$, and $[-\infty, +\infty]$ are valid elements of $\bar{\mathbb{IR}}$.

Since we cannot divide by zero in $\bar{\mathbb{R}}$, repeating the division performed in Example 2.3.1 requires the interpretation (2.3), and produces

$$[1, 2] \div [-5, 3] = [-\infty, -\tfrac{1}{5}] \cup [\tfrac{1}{3}, \infty].$$

As with the projective extension, we could simply introduce the notion of extended intervals (thus removing the demand $\underline{a} \leq \bar{a}$ for intervals), now with the meaning

$$[\tfrac{1}{3}, -\tfrac{1}{5}] = [-\infty, -\tfrac{1}{5}] \cup [\tfrac{1}{3}, \infty]. \tag{2.4}$$

Alternatively, we could accept the fact that some interval operations may produce a union of intervals. This line of action, however, leads to some tricky implementation issues. A yet simpler way of resolving the whole issue would be to return the entire line $[-\infty, +\infty]$ when dividing by zero, possibly with the exception that $[a]/[0] = [\emptyset]$ if $0 \notin [a]$. As this approach leads to an unneccessary loss of information, it is therefore not very widespead. We will use the definition

$$\bar{\mathbb{IR}} = \{[\underline{a}, \bar{a}] : \underline{a}, \bar{a} \in \bar{\mathbb{R}}\}$$

where the case $\underline{a} \geq \bar{a}$ corresponds to an extended interval of type (2.4).

2.3.3 The ugly: signed zero

While all elements of $\mathbb{R}^*$ have unique reciprocals, this is not the case for all members of $\bar{\mathbb{R}}$. Indeed, in $\bar{\mathbb{R}}$ we have $1/(-\infty) = 1/(+\infty) = 0$, whereas $1/0$ is undefined. In an attempt to resolve this problem, it is possible to equip $\bar{\mathbb{R}}$ with *signed zeroes*, satisfying $x/(+0) = \text{sign}(x) \cdot (+\infty)$ and $x/(-0) = \text{sign}(x) \cdot (-\infty)$ for $x \neq \pm 0$. In effect, this gives both infinites and both signed zeroes unique reciprocals, just like all other elements of $\bar{\mathbb{R}}$:

$$1/(+\infty) = +0, \qquad 1/(+0) = +\infty, \qquad 1/(-\infty) = -0, \qquad 1/(-0) = -\infty.$$

As an illustration, we have $1 \div [+0, 2] = [\frac{1}{2}, +\infty]$, whereas $1 \div [-0, 2] = [-\infty, +\infty]$. Unfortunately, there is no natural way of propagating the sign of the zero under addition and subtraction: what sign should $(+0) + (-0)$ or even $x - x$ have?

The IEEE standard incorporates signed infinities as well as signed zeroes. The signs appear naturally within the actual sign-exponent-mantissa encoding of the floating point numbers. Even though -0 and $+0$ are distinct values, they both compare as equal, and are only distinguishable by comparing their sign bits. Regarding addition and subtraction, the standard [IE85] states

> When the sum of two operands with opposite signs (or the difference of two operands with equal signs) is exactly zero, the sign of that sum (or difference) shall be + in all rounding modes except round toward $-\infty$, in which mode that sign shall be $-$. However, $x + x = x - (-x)$ retains the same sign as x even when x is zero.

Thus, on any computer compliant with the IEEE standard, we have $(+0) + (-0) = x - x = +0$, unless we are rounding with $\bigtriangledown$, answering the question posed above.

The signed zeroes were not introduced for their mathematical elegance: their presence is due to computer manufacturers' desire to reduce the number of fatal floating point errors. Instead of having to abort a computation that happens to perform a division by zero, it is much more desirable to produce a well-defined result of the division. Without the signed zero, this is simply not possible.

2.3.4 The extended interval division

In light of the previous discussion, the extended interval division is defined over the space $\overline{\mathbb{IR}}$ (equipped with signed zeros), where we allow for extended intervals of the form (2.4). Following [Ra96], we define division over $\overline{\mathbb{IR}}$ as follows

$$[a] \div [b] = \begin{cases} [a] \times [1/\overline{b}, 1/\underline{b}] & \text{if } 0 \notin [b], \\ [-\infty, +\infty] & \text{if } 0 \in [a] \text{ and } 0 \in [b], \\ [\overline{a}/\underline{b}, +\infty] & \text{if } \overline{a} < 0 \text{ and } \underline{b} < \overline{b} = 0, \\ [\overline{a}/\underline{b}, \overline{a}/\overline{b}] & \text{if } \overline{a} < 0 \text{ and } \underline{b} < 0 < \overline{b}, \\ [-\infty, \overline{a}/\overline{b}] & \text{if } \overline{a} < 0 \text{ and } 0 = \underline{b} < \overline{b}, \\ [-\infty, \underline{a}/\underline{b}] & \text{if } 0 < \underline{a} \text{ and } \underline{b} < \overline{b} = 0, \\ [\underline{a}/\overline{b}, \underline{a}/\underline{b}] & \text{if } 0 < \underline{a} \text{ and } \underline{b} < 0 < \overline{b}, \\ [\underline{a}/\overline{b}, +\infty] & \text{if } 0 < \underline{a} \text{ and } 0 = \underline{b} < \overline{b}, \\ [\emptyset] & \text{if } 0 \notin [a] \text{ and } [b] = [0, 0]. \end{cases} \tag{2.5}$$

Case 1 deals with non-zero divisors, although it now incorporates quotients such as $[6, 8] \div [2, +\infty] = [+0, 4]$. Cases 4 and 7 yield extended intervals, i.e., these particular results actually consist of a union of two infinite intervals. In [Ra96], it is proved that the division defined by (2.5) is inclusion isotonic, i.e., if $[a] \subseteq [a']$, and $[b] \subseteq [b']$, then $[a] \div [b] \subseteq [a'] \div [b']$, which generalizes Theorem 2.2.7.

It is worth pointing out that, although signed zeros are not explicitly present in (2.5), their properities are mimicked in the formulas. As an example, let us consider case 5, where the condition is stated as "if $\overline{a} < 0$ and $0 = \underline{b} < \overline{b}$". For all practical purposes, this can be interpreted as "if $\overline{a} < 0$ and $+0 = \underline{b} < \overline{b}$".

There are two main advantages of having access to an extended interval division in a computing environment. First, all run-time errors of the type *division by zero*

are immediately avoided. Usually, an error of this type will cause a program to crash, unless some serious error-handling capabilities have been provided by the programmer. Second, it is actually desirable, from a mathematical point of view, to be able to perform extended division. Later on, we will see a striking example of this when we study the interval Newton method.

2.4 Floating point interval arithmetic

When implementing interval arithmetic on a computer, we no longer work over the space $\mathbb{R}$, but rather $\mathbb{F}$ - the floating point numbers of the computer. This is a finite set, and so is $\mathbb{IF}$ – the set of all intervals whose endpoints belong to $\mathbb{F}$:

$$\mathbb{IF} = \{[\underline{a}, \bar{a}] : \underline{a} \leq \bar{a}; \quad \underline{a}, \bar{a} \in \mathbb{F}\}.$$

As discussed earlier, $\mathbb{F}$ is not arithmetically closed. Thus, when performing arithmetic on intervals in $\mathbb{F}$ we must round the resulting interval *outwards* to guarantee inclusion of the true result. By this, we mean that the lower bound is rounded down, and the upper bound is rounded up. For $[a], [b] \in \mathbb{IF}$, we define

$$\begin{aligned}
[a] + [b] &= [\bigtriangledown(\underline{a} + \underline{b}), \bigtriangleup(\bar{a} + \bar{b})] \\
[a] - [b] &= [\bigtriangledown(\underline{a} - \bar{b}), \bigtriangleup(\bar{a} - \underline{b})] \\
[a] \times [b] &= [\min\{\bigtriangledown(\underline{a}\underline{b}), \bigtriangledown(\underline{a}\bar{b}), \bigtriangledown(\bar{a}\underline{b}), \bigtriangledown(\bar{a}\bar{b})\}, \\
&\qquad \max\{\bigtriangleup(\underline{a}\underline{b}), \bigtriangleup(\underline{a}\bar{b}), \bigtriangleup(\bar{a}\underline{b}), \bigtriangleup(\bar{a}\bar{b})\}] \\
[a] \div [b] &= [\min\{\bigtriangledown(\underline{a}/\underline{b}), \bigtriangledown(\underline{a}/\bar{b}), \bigtriangledown(\bar{a}/\underline{b}), \bigtriangledown(\bar{a}/\bar{b})\}, \\
&\qquad \max\{\bigtriangleup(\underline{a}/\underline{b}), \bigtriangleup(\underline{a}/\bar{b}), \bigtriangleup(\bar{a}/\underline{b}), \bigtriangleup(\bar{a}/\bar{b})\}], \quad \text{if } 0 \notin [b].
\end{aligned}$$

Recall that $\bigtriangledown(x)$ and $\bigtriangleup(x)$ were defined in Section 1.3.2. The resulting type of arithmetic is called interval arithmetic with *directed rounding*. As we shall see, this is easily implemented on a computer that supports the directed roundings.

With regards to efficiency, a single $\mathbb{IF}$-multiplication requires eight $\mathbb{F}$-multiplications: four products must be computed under two different rounding modes. As before, it is customary to break the formula for multiplication into nine cases (depending of the signs of the operands' endpoints). Out of these nine cases, only one will involve four $\mathbb{F}$-multiplications; the remaining eight will need just two. In a similar manner, the (non-extended) $\mathbb{IF}$-division can be split into six cases.

Exercise 2.4.1 *Derive the optimal formulas for division in $\mathbb{IF}$, assuming that a floating point comparison is much faster than a floating point division.*

Extending the floating point interval arithmetic via (2.5) is straight-forward, and yields the set

$$\mathbb{I}\bar{\mathbb{F}} = \{[\underline{a}, \bar{a}] : \underline{a}, \bar{a} \in \bar{\mathbb{F}}\},$$

where the case $\underline{a} \geq \bar{a}$ corresponds to an extended interval of type (2.4).

2.4.1 A MATLAB implementation of interval arithmetic

To illustrate how easy it is to get started, we present a simple MATLAB implementation of (non-extended) interval arithmetic with directed rounding. A corresponding implementation in the C++ programming language is listed in Appendix B.

As most modern programming languages, MATLAB uses *classes* to define new data types, and *methods* to define the functionality of a user-defined class. A new class can be added to the MATLAB environment by specifying a structure that provides data storage for the object, and creating a class directory containing m-files[6] that operate on the object. These m-files contain the methods for the class. MATLAB is somewhat peculiar in that it demands a certain file hierarchy associated with each class. In Figure 2.6 we illustrate a simple setup for our `interval` class. We will explain the purpose of the different m-files as we go along.

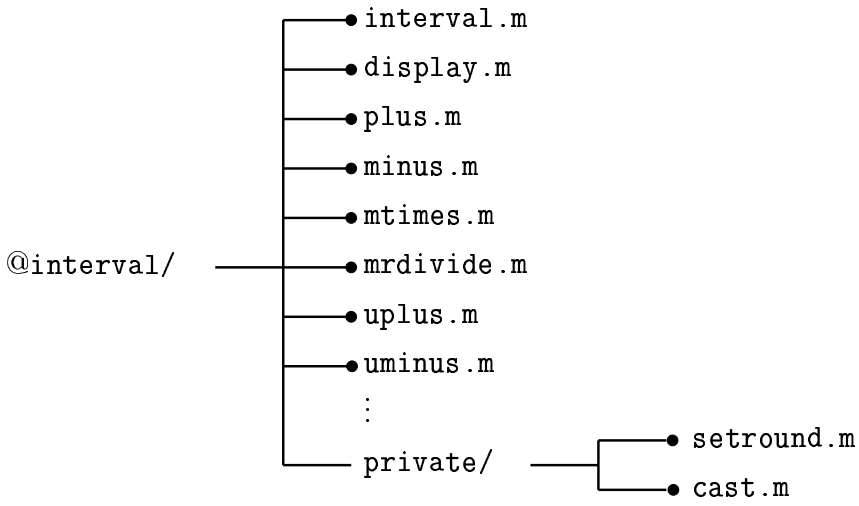

Figure 2.6: The hierarchy of the MATLAB `interval` class.

As we want to build an `interval` class, we begin by creating a directory called `@interval` where all m-files associated to the `interval` class will reside. Having done this, we create the m-file `interval.m`, in which we define what is meant by an interval. It is natural to implement an interval as a class consisting of two numbers – the endpoints of the interval:

```
01 function iv = interval(lo, hi)
02 % A naive interval class constructor.
03 if nargin == 1
04     hi = lo;
05 elseif ( hi < lo )
06     error('The endpoints do not define an interval.');
07 end
```

[6] An m-file is simply a text file `filename.m` containing a sequence of MATLAB statements to be executed. The file extension of `.m` makes this a MATLAB m-file.

```
08 iv.lo = lo; iv.hi = hi;
09 iv = class(iv,'interval');
```

By including lines 03 and 04, we allow the constuctor to automatically convert a single number x to a thin interval $[x, x]$. Note that, as opposed to most programming languages, MATLAB only supports the `double` format, which means that no explicit type declarations have to (or can!) be made.

We must also inform MATLAB how to display `interval` objects. This is achieved via the m-file `display.m`:

```
01 function display(iv)
02 % A simple output formatter for the interval class.
03 disp([inputname(1), ' = ']);
04 fprintf('  [%17.17f, %17.17f]\n', iv.lo, iv.hi);
```

We can now input/output intervals within the MATLAB environment:

```
>> a = interval(3, 4), b = interval(2, 5), c = interval(1)
a =
  [3.00000000000000000, 4.00000000000000000]
b =
  [2.00000000000000000, 5.00000000000000000]
c =
  [1.00000000000000000, 1.00000000000000000]
```

When creating user-defined classes, it is often desirable to change the behavior of the MATLAB operators and functions[7] for cases when the arguments are user-defined classes. This can be accomplished by *overloading* the relevant functions. Overloading enables a function to handle different types and numbers of input arguments, and perform whatever operation is appropriate for the situation at hand.

Each native MATLAB operator has an associated function name (e.g., the + operator has an associated `plus.m` function). Any such operator can be overloaded by creating an m-file with the appropriate name in the class directory. In Table 2.4.1, we list the operators we intend to overload in our `interval` class.

Here, another feature of MATLAB becomes evident: the MATLAB-engine regards all numeric elements as matrices, even if they are single numbers. Indeed, a single number can be viewed as a 1×1-matrix.

Let us begin by writing a function that returns the sum of two intervals:

```
01 function result = plus(a, b)
02 % Overloading the '+' operator for intervals.
03 [a, b] = cast(a, b);
04 setround(-inf);
05 lo = a.lo + b.lo;
```

[7] In what follows, we will not distinguish between *functions* and *methods*.

Operation	m-file	New description
a + b	plus(a,b)	Interval addition
a - b	minus(a,b)	Interval subtraction
a * b	mtimes(a,b)	Interval multiplication
a / b	rmdiv(a,b)	Interval division
+a	uplus(a)	Unary plus
-a	uminus(a)	Unary minus

Table 2.1: Overloaded MATLAB arithmetic methods.

```
06 setround(+inf);
07 hi = a.hi + b.hi;
08 setround(0.5);
09 result = interval(lo, hi);
```

Let us examine this small piece of code: First, the function `cast`, appearing on line 03, makes sure that the inputs `a` and `b` are intervals. If one of them is not an interval, it is converted to an interval by a call the interval constructor, see the listing below.

```
01 function [a, b] = cast(a, b)
02 % Casts non-intervals to intervals.
03 if ~isa(a, 'interval')
04     a = interval(a);
05 end
06 if ~isa(b, 'interval')
07     b = interval(b);
08 end
```

Casting[8] allows for expressions like $[1,2]+3$, which is converted to $[1,2]+[3,3]$, and evaluated to $[4,5]$.

Second, the function `setround`, appearing on lines 04, 06, and 08 of the file `plus.m`, instructs the MATLAB-engine to switch the rounding direction before performing an arithmetic operation. This function is implemented in the auxiliary file `setround.m`, presented below:

```
01 function setround(rnd)
02 % A switch for changing rounding mode. The arguments
03 % {+inf, -inf, 0.5, 0} correspond to the roundings
04 % {upward, downward, to nearest, to zero}, respectively.
05 system_dependent('setround',rnd);
```

We consider both functions `cast` and `setround` to be intrinsic to the `interval` class. By placing their m-files in the `private` subdirectory, these functions are hidden from non-interval classes.

[8] In the programming language C++, casting is implicitly performed at the compilation stage. This simplifies the actual programming, but can also produce hard-to-find bugs.

Carrying on, it is straight-forward to write functions that overload the remaining arithmetic operations $-$, $\times$, and $\div$. Below, we present a MATLAB listing of the division algorithm:

```
01 function result = mrdivide(a, b)
02 % A non-optimal interval division algorithm.
03 [a, b] = cast(a, b);
04 if ( (b.lo <= 0.0) & (0.0 <= b.hi) )
05     error('Denominator straddles zero.');
06 else
07     setround(-inf);
08     tmp1 = min(a.lo / b.lo, a.lo / b.hi);
09     tmp2 = min(a.hi / b.lo, a.hi / b.hi);
10     lo = min(tmp1, tmp2);
11     setround(+inf);
12     tmp1 = max(a.lo / b.lo, a.lo / b.hi);
13     tmp2 = max(a.hi / b.lo, a.hi / b.hi);
14     hi = max(tmp1, tmp2);
15     setround(0.5);
16     result = interval(lo, hi);
17 end
```

Performing some simple interval calculations, we have:

```
>> a+b, a-b, a*b, a/b
ans =
  [5.00000000000000000, 9.00000000000000000]
ans =
  [-2.00000000000000000, 2.00000000000000000]
ans =
  [6.00000000000000000, 20.00000000000000000]
ans =
  [0.59999999999999998, 2.00000000000000000]
```

The outward rounding is apparent in the left endpoint of the last result. All other endpoints were computed exactly. We should point out that our interval constructor is still very rudimentary, and does not handle user-input adequately. As an example, suppose we would like to generate the smallest interval containing 1/10. As a first attempt, we may try something like

```
>> interval(1/10)
ans =
  [0.10000000000000001, 0.10000000000000001]
```

which is *not* what we wanted. The problem here is that the quotient 1/10 is *first* rounded to a single floating point number, which is *then* converted to a thin interval. Since 1/10 has no exact representation in the floating point format, we obtain an interval that does not contain 1/10. A way to work around this is to declare either the nominator or denominator as an interval. Since integers have exact representations, no rounding takes place at this stage. It is only when the division takes place that the directed rounding kicks in, producing a non-thin interval straddling 1/10:

```
>> interval(1)/10
ans =
  [0.09999999999999999, 0.10000000000000001]
```

More sophisticated interval libraries provide a means for entering strings of numbers[9], such as

```
>> interval('1/10')
ans =
  [0.09999999999999999, 0.10000000000000001]
```

Nevertheless, this has a cost in programming effort which we are not willing to pay at the moment.

Continuing our calculations, we can now illustrate the sub-distributive property of interval arithmetic:

```
>> c = interval(0.25, 0.50)
c =
  [0.25000000000000000, 0.50000000000000000]
>> a*(b+c), a*b+a*c
ans =
  [5.25000000000000000, 22.00000000000000000]
ans =
  [5.00000000000000000, 22.00000000000000000]
```

Notice the differing lower endpoints; clearly the expression $ab + ac$ produces a wider result than $a(b + c)$. Since all computations in this example are exact, the outward rounding does not affect the result.

Exercise 2.4.2 *Modify the appropriate m-files so they perform multiplication and division by checking the signs of the operands' endpoints.*

Exercise 2.4.3 *Add some interval functions (e.g.* `sin(x)` *and* `pow(x,n)`*) to the* `interval` *class. Note that this requires some knowledge of how accurate the corresponding real-valued functions are in the underlying programming environment.*

Exercise 2.4.4 *Do you know any other programming language that supports operator overloading? If so, try to implement a rudimentary interval arithmetic library whose syntax permits expressions like* `x + y` *and* `z = sin(pow(x,2))`*, where* `x`*,* `y`*, and* `z` *are of type* `interval`*.*

Now that we have all arithmetic operations in place, let us consider the built-in relational operators provided by `MATLAB`. Some of these are listed in Table 2.4.1.

[9] The `MATLAB` package `IntLab` has this functionality, as does the `C++` toolbox `CXSC`, see [INv4] and [CXSC], respectively.

Operation	m-file	New description
`a == b`	`eq(a,b)`	Equal to
`a ~= b`	`ne(a,b)`	Not equal to
`a <= b`	`le(a,b)`	Subset of
`a < b`	`lt(a,b)`	Proper subset of
`a & b`	`and(a,b)`	Intersection
`a | b`	`or(a,b)`	Interval hull

Table 2.2: Overloaded `MATLAB` set-relation methods.

When overloading these methods, we will give them new, interval-based, meanings. Let us begin with the simplest of them all: the *equality* relation. Two intervals are *equal* exactly when their endpoints agree. Analogously, two intervals are *not equal* if at least one of their endpoints differ. Both functions can be implemented in a few lines.

```
01 function result = eq(a, b)
02 % The '(e)qual' operator '=='.
03 [a, b] = cast(a, b);
04 result = ( (a.lo == b.lo) & (a.hi == b.hi) );
```

```
01 function result = ne(a, b)
02 % The '(n)ot (e)qual' operator '~='.
03 [a, b] = cast(a, b);
04 result = ( (a.lo ~= b.lo) | (a.hi ~= b.hi) );
```

Turning to the order-relations *less or equal* and *less than*, we will interpret them as the set-relations *inclusion* $\subseteq$ and *proper inclusion* $\overset{\circ}{\subset}$, respectively.

```
01 function result = le(a, b)
02 % The '(l)ess or (e)qual' operator '<='. Means 'a inside b'.
03 [a, b] = cast(a, b);
04 result = ( (b.lo <= a.lo) & ( a.hi <= b.hi) );
```

```
01 function result = lt(a, b)
02 % The '(l)ess (t)han' operator '<'. Means 'a inside int(b)',
03 [a, b] = cast(a, b);
04 result = ( (b.lo < a.lo) & ( a.hi < b.hi) );
```

The four functions we have defined so far are all *boolean*, i.e., their return-values come from the set {`true`, `false`}. In `MATLAB` (and most other programming languages), these alternatives are coded as '1' and '0', respectively.

```
>> a = interval(1, 10); b = interval(-2, 3); c = interval(3, 5);
>> [a==b, a==c, a~=b, a~=c, a<=b, b<=a, a<c, c<a]
ans =
     0     0     1     1     0     0     0     1
```

Finally, we will implement the logical operatiors *and* and *or*, but we will re-define them as the set-operations *intersection* $\cap$, and *hull* $\sqcup$, respectively. One complication here is that two intervals may have an empty intersection. Seeing that our simple interval constructor does not accommodate empty intervals, we will return the MATLAB version of the empty set, accompanied by a warning[10] whenever this situation occurs.

```
01 function result = and(a, b)
02 % The 'and' operator '&'. Means 'a intersected with b'.
03 [a, b] = cast(a, b);
04 if ( (a.hi < b.lo) | (b.hi < a.lo) )
05     warning('The intervals do not intersect.');
06     result = [];
07 else
08     result = interval(max(a.lo, b.lo), min(a.hi, b.hi));
09 end
```

Since we have not defined the interval methods to operate on empty sets, it is vital that we have a means for detecting an empty set. Fortunately, MATLAB has a built-in function `isempty` that can reveal whether the outcome of an interval-intersection is an empty set or not.

```
>> a=interval(1,3); b=interval(4,5); c=interval(2,5);
>> aANDb = a & b; aANDc = a & c;
Warning: The intervals do not intersect.
> In /matlab/@interval/and.m at line 5
aANDb =
     []
aANDc =
  [2.00000000000000000, 3.00000000000000000]
>> [isempty(aANDb), isempty(aANDc)]
ans =
     1     0
```

The hull of two intervals is always well-defined, and thus straight-forward to implement.

```
01 function result = or(a, b)
02 % The 'or' operator '|'. Means 'hull of a and b'.
03 [a, b] = cast(a, b);
04 result = interval(min(a.lo, b.lo), max(a.hi, b.hi));
```

```
>> aORb = a | b, aORc = a | c
aORb =
  [1.00000000000000000, 5.00000000000000000]
aORc =
  [1.00000000000000000, 5.00000000000000000]
```

[10] It may be desirable to comment out the warning in order to minimize unnecessary output.

Exercise 2.4.5 *Make the necessary modifications to the m-files* `interval.m` *and* `display.m` *to accommodate input/output of empty intervals. As a first step you must find a good representation for the empty interval.*

Exercise 2.4.6 *How would you modify the remaining m-files to fully incorporate empty intervals?*

Exercise 2.4.7 *Implement a new class* `xinterval` *for extended intervals. Try to extend all associated interval methods described in this section. [Warning: this takes some effort!]*

Exercise 2.4.8 *An interval can also be represented by the two numbers* $< m, r >$*, where* m *is the midpoint, and* r *is the radius of the interval. Derive the interval-arithmetic rules using only these two quantities. From a numerical point of view, what are the merits/drawbacks of this (*`midrad`*) representation compared to the end-point (*`infsup`*) representation?*