Computing under Interval Uncertainty: Computational Complexity

In this chapter, we will briefly describe the computational complexity of the range estimation problem under interval uncertainty.

**Linear case.** Let us start with the simplest case of a linear function

$$y = f(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i.$$  

In this case, substituting the (approximate) measured values $\tilde{x}_i$, we get the approximate value

$$\tilde{y} = a_0 + \sum_{i=1}^{n} a_i \cdot \tilde{x}_i$$

for $y$.

The approximation error $\Delta y = \tilde{y} - y$ of this approximation can be described as

$$\Delta y = \sum_{i=1}^{n} a_i \cdot \Delta x_i,$$

where each input error $\Delta x_i$ can take any value from $-\Delta_i$ to $\Delta_i$.

The sum $\sum_{i=1}^{n} a_i \cdot \Delta x_i$ attains its largest possible value if each term $a_i \cdot \Delta x_i$ in this sum attains the largest possible value:

- If $a_i \geq 0$, then this term is a monotonically non-decreasing function of $\Delta x_i$, so it attains its largest value at the largest possible value $\Delta x_i = \Delta_i$; the corresponding largest value of this term is $a_i \cdot \Delta_i$.
- If $a_i < 0$, then this term is a decreasing function of $\Delta x_i$, so it attains its largest value at the smallest possible value $\Delta x_i = -\Delta_i$; the corresponding largest value of this term is $-a_i \cdot \Delta_i = |a_i| \cdot \Delta_i$.

In both cases, the largest possible value of this term is $|a_i| \cdot \Delta_i$, so, the largest possible value of the sum $\Delta y$ is
\[ \Delta = |a_1| \cdot \Delta_1 + \ldots + |a_n| \cdot \Delta_n. \]

Similarly, the smallest possible value of \( \Delta y \) is \(-\Delta\).

Hence, the interval of possible values of \( \Delta y \) is \([-\Delta, \Delta]\), and the interval of possible values of the actual value \( y \) is \([\tilde{y} - \Delta, \tilde{y} + \Delta]\).

The corresponding range can be computed in linear time, i.e., efficiently.

**Quadratic case.** Already for quadratic functions

\[
y = f(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i \cdot x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot x_i \cdot x_j,
\]

the problem of computing the exact range

\[
y = f(x_1, \ldots, x_n) = \{ f(x_1, \ldots, x_n) : x_1 \in x_1, \ldots, x_n \in x_n \}
\]

down over interval inputs \( x_i \in x_i = [\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i] \) is, in general, NP-hard; see, e.g., [182, 334].

**What is NP-hard? A brief description.** NP-hard means, crudely speaking, that no feasible (polynomial time) algorithm can compute the exact endpoints of the range \( y \) for all possible intervals \( x_1, \ldots, x_n \). (Strictly speaking, this interpretation is only true under the widely believed but still unproven hypothesis that P\( \neq \)NP).

**Towards a more precise description of NP-hardness: the notion of a feasible algorithm.** The notion of NP-hardness is related to the fact that some algorithms take so much computation time that even for inputs of reasonable size, the computation time exceeds the lifetime of the Universe – and thus, cannot be practically computed. For example, if for \( n \) inputs, the algorithm takes time \( 2^n \), then for \( n \approx 300-400 \), the resulting computation time is unrealistically large. How can we separate “realistic” (“feasible”) algorithms from non-feasible ones?

The running time of an algorithm depends on the size of the input. In the computer, every object is represented as a sequence of bits (0s and 1s). Thus, for every computer-represented object \( x \), it is reasonable to define its size (or length) \( \text{len}(x) \) as the number of bits in this object’s computer representation.

It is known that in most feasible algorithms, the running time on an input \( x \) is bounded either by the size of the input, or by the square of the size of the input, or, more generally, by a polynomial of the size of the input. It is also known that in most non-feasible algorithms, the running time grows exponentially (or even faster) with the size, so it cannot be bounded by any polynomial. In view of this fact, in theory of computation, an algorithm is usually called feasible if its running time is bounded by a polynomial of the size of the input. This definition is not perfect: e.g., if the running time on input of size \( n \) is \( 10^{40} \cdot n \), then this running time is bounded by a polynomial