How to Deal with Uncertainties in Computing: from Probabilistic and Interval Uncertainty to Combination of Different Approaches, with Applications to Engineering and Bioinformatics

Vladik KREINOVICH [a,1]

[a] *Department of Computer Science, University of Texas at El Paso, USA*

Abstract. Most data processing techniques traditionally used in scientific and engineering practice are statistical. These techniques are based on the assumption that we know the probability distributions of measurement errors etc.

In practice, often, we do not know the distributions, we only know the bound Δ on the measurement accuracy – hence, after the get the measurement result $\widetilde{x}$, the only information that we have about the actual (unknown) value x of the measured quantity is that x belongs to the interval $[\widetilde{x}-\Delta,\widetilde{x}+\Delta]$. Techniques for data processing under such interval uncertainty are called interval computations; these techniques have been developed since 1950s.

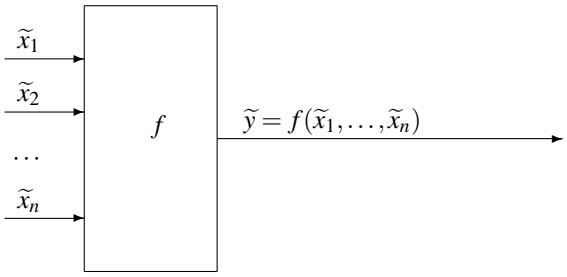
In many practical problems, we have a combination of different types of uncertainty, where we know the probability distribution for some quantities, intervals for other quantities, and expert information for yet other quantities. The purpose of this paper is to describe the theoretical background for interval and combined techniques and to briefly describe the existing practical applications.

Keywords. uncertainty, interval computations, probabilistic uncertainty

1. Data Processing under Uncertainty: A General Problem

Need for indirect measurements. In many practical situations, we are interested in the value of a quantity y which is difficult (or even impossible) to measure directly, such as the amount of oil in a given area or tomorrow's temperature. Since we cannot measure y directly, a natural idea is to measure y *indirectly*, i.e., find some easier-to-measure quantities $x_1,\ldots,x_n$ which are related to y by a known dependence $y = f(x_1,\ldots,x_n)$, and use the results $\widetilde{x}_i$ of measuring x_i to estimate y as $f(\widetilde{x}_1,\ldots,\widetilde{x}_n)$.

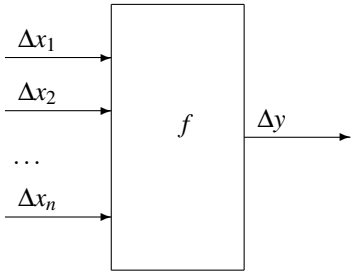
[1] Author: Vladik Kreinovich, Department of Computer Science, University of Texas at El Paso, 500 W. University, El Paso, TX 79968, USA; E-mail: vladik@utep.edu.



Indirect measurements are also known as *data processing*.

Need to take uncertainty into account. Measurements are never 100% accurate: the measurement result $\widetilde{x}_i$ is, in general, somewhat different from the actual (unknown) values x_i, and we have a non-zero *measurement error* $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$. Hence, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of indirect measurement is, in general, different, from the actual value $y = f(x_1, \ldots, x_n)$.

How big is the resulting uncertainty $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$?



Probabilistic and interval uncertainty. Traditional approach to uncertainty assumes that we know the probability distributions for Δx_i (usually, they are assumed to be Gaussian). In many practical situations, we can indeed determine these probability distributions by *calibrating* the corresponding measuring instrument (MI), i.e., to comparing its result with the results of measuring the same quantity by a much more accurate ("standard") MI.

However, there are two case when the calibration is not possible:

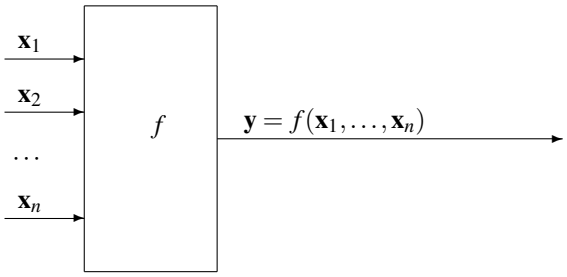
- *state-of-the-art measurements*, e.g., in fundamental science, when we use the best MI available – and so, no more-accurate MIs are available for comparison, and
- *manufacturing*, when calibration of each MI is possible, but would be prohibitively expensive.

In such cases, all we know is the upper bound Δ_i on the (absolute value of the) measurement error $|\Delta x_i|$, the bound provided by the manufacturer of the MI; see, e.g., [13]. In this case, once we have the measurement result $\widetilde{x}_i$, the only thing we know about the actual (unknown) value x_i is that x_i belongs to the *interval* $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. Algorithms for processing such interval uncertainty are known as *interval computation*; see, e.g., [3,8].

2. Interval Computations

Interval computations: a problem. We know an algorithm $y = f(x_1, \ldots, x_n)$ and we know n intervals $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$. We want to compute the range of possible values of y:

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



In general, this problem is NP-hard even for quadratic f; see, e.g., [5]. So, from the practical viewpoint, we face the following challenges: when are feasible algorithms possible? and when computing $\mathbf{y}$ is not feasible, how can we find a good approximation $\mathbf{Y} \supseteq \mathbf{y}$.

Interval computations: a brief history. Interval computations can be traced to Archimedes who used lower and upper bounds to estimate π. In modern times, this area was revived by three 1950s pioneers: M. Warmus (Poland), T. Sunaga (Japan), and R. Moore (USA). The early 1960s witnessed the first boom, including taking interval uncertainty into account when planning spaceflights to the Moon. Current applications range from designing elementary particle colliders to checking whether a comet will hit the Earth to robotics and chemical engineering; see, e.g., [3,8].

But why not use Maximum Entropy (MaxEnt) approach? Often in practice, many different probability distributions are consistent with the same observations. Instead of considering *all* possible distributions – as interval computations do – why not select *one* of these distributions – e.g., the one with the largest entropy. For example, if all we know is that x belongs to an interval, then this MaxEnt approach leads to a uniform distribution on this interval. For several variables, if we have no information about their dependence, this approach concludes that these variables are independent.

This idea is reasonable, but it has a serious limitation. As an example, let us take the simplest possible data proceeding algorithm $y = x_1 + \ldots + x_n$. In this case, $\Delta y = \Delta x_1 + \ldots + \Delta x_n$. Let us assume that all measurement errors Δx_i are limited to the interval $[-\Delta, \Delta]$. Then, the worst case situation is when $\Delta x_i = \Delta$ and thus, $\Delta y = n \cdot \Delta$.

On the other hand, if we use the MaxEnt approach, then for large n, due to Central Limit Theorem, Δy is close to normal, with $\sigma = \Delta \cdot \frac{\sqrt{n}}{\sqrt{3}}$. So, if we, as usual, ignore deviations exceeding 6σ, we get an upper bound $\sim \sqrt{n}$ – but it is possible that $\Delta = n \cdot \Delta \sim n \gg \sqrt{n}$.

Conclusion: using a single distribution can be very misleading, especially if we want guaranteed results – and this is very important in high-risk application areas such as space exploration or nuclear engineering.

Interval arithmetic: foundations of interval techniques. The general problem of interval computations is to compute the range of a given function $f(x_1,\ldots,x_n)$ on given intervals $\mathbf{x}_i$. For arithmetic operations $f(x_1,x_2)$ (and for elementary functions), we have explicit formulas for the range; e.g., when $x_1 \in \mathbf{x}_1 = [\underline{x}_1,\overline{x}_1]$ and $x_2 \in \mathbf{x}_2 = [\underline{x}_2,\overline{x}_2]$, then:

- The range $\mathbf{x}_1 + \mathbf{x}_2$ for $x_1 + x_2$ is $[\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2]$.
- The range $\mathbf{x}_1 - \mathbf{x}_2$ for $x_1 - x_2$ is $[\underline{x}_1 - \overline{x}_2, \overline{x}_1 - \underline{x}_2]$.
- The range $\mathbf{x}_1 \cdot \mathbf{x}_2$ for $x_1 \cdot x_2$ is $[\underline{y},\overline{y}]$, where

$$\underline{y} = \min(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2) \text{ and } \overline{y} = \max(\underline{x}_1 \cdot \underline{x}_2, \underline{x}_1 \cdot \overline{x}_2, \overline{x}_1 \cdot \underline{x}_2, \overline{x}_1 \cdot \overline{x}_2).$$

The range $1/\mathbf{x}_1$ for $1/x_1$ is $[1/\overline{x}_1, 1/\underline{x}_1]$ (if $0 \notin \mathbf{x}_1$).

Straightforward interval computations: example. Inside the computer, each computation is a sequence of arithmetic operations. For example, computing $f(x) = (x-2)\cdot(x+2)$ means that we compute $r_1 := x - 2$, then $r_2 := x + 2$, and finally, $r_3 := r_1 + r_2$.

To find the range of function on a given interval, a seemingly natural idea is to perform the same operations, but with *intervals* instead of *numbers*. For example, for $x \in [1,2]$, we compute $\mathbf{r}_1 := [1,2] - [2,2] = [-1,0]$, $\mathbf{r}_2 := [1,2] + [2,2] = [3,4]$, and $\mathbf{r}_3 := [-1,0]\cdot[3,4] = [-4,0]$.

However, the actual range is narrower: $f(\mathbf{x}) = [-3,0]$. Thus, we need to come up with more efficient ways of computing an enclosure $\mathbf{Y} \supseteq \mathbf{y}$.

First idea: use of monotonicity. For arithmetic operations, we had exact ranges – because $+$, $-$, and $\cdot$ are monotonic in each variable. In general, if $f(x_1,\ldots,x_n)$ is (non-strictly) increasing ($f \uparrow$) in each x_i, then $f(\mathbf{x}_1,\ldots,\mathbf{x}_n) = [f(\underline{x}_1,\ldots,\underline{x}_n), f(\overline{x}_1,\ldots,\overline{x}_n)]$. We can get similar simplifying formulas if $f \uparrow$ for some x_i and $f \downarrow$ for other x_j.

How can we check whether $f \uparrow$? It is well known that $f \uparrow$ in x_i if $\dfrac{\partial f}{\partial x_i} \geq 0$ for all values $x = (x_1,\ldots,x_n)$. So, to check for monotonicity, it is sufficient to check whether the range $[\underline{r}_i,\overline{r}_i]$ of $\dfrac{\partial f}{\partial x_i}$ on $\mathbf{x}_i$ has $\underline{r}_i \geq 0$. Differentiation can easily done, e.g., by Automatic Differentiation (AD) tools. Estimating ranges of $\dfrac{\partial f}{\partial x_i}$ can be done, e.g., by straightforward interval computations. So, if the range $[\underline{r}_i,\overline{r}_i]$ of each $\dfrac{\partial f}{\partial x_i}$ on $\mathbf{x}_i$ has $\underline{r}_i \geq 0$, then

$$f(\mathbf{x}_1,\ldots,\mathbf{x}_n) = [f(\underline{x}_1,\ldots,\underline{x}_n), f(\overline{x}_1,\ldots,\overline{x}_n)].$$

In the above example of $f(x) = (x-2)\cdot(x+2)$ on $\mathbf{x} = [1,2]$, we have $\dfrac{df}{dx} = 1\cdot(x+2) + (x-2)\cdot 1 = 2x$. The range of this derivative is $[\underline{r},\overline{r}] = [2,4]$, with $2 \geq 0$. Thus, we get $f([1,2]) = [f(1), f(2)] = [-3,0]$, which is the exact range.

Second idea: centered form. Not all functions are monotonic. In such cases, we can use another idea from calculus – the Intermediate Value Theorem, according to which

$$f(x_1,\ldots,x_n) = f(\widetilde{x}_1,\ldots,\widetilde{x}_n) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\chi)\cdot(x_i-\widetilde{x}_i)$$

for some $\chi_i \in \mathbf{x}_i$. We can thus conclude that $f(x_1,\ldots,x_n) \in \mathbf{Y}$, where

$$\mathbf{Y} = \widetilde{y} + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(\mathbf{x}_1,\ldots,\mathbf{x}_n)\cdot[-\Delta_i,\Delta_i].$$

Here, differentiation can be performed by the AD tools, and the range of the derivatives can be estimated, if appropriate, by monotonicity, or by straightforward interval computations – or itself by centered form (this will take more time but lead to a more accurate estimation).

As an example, we can take a non-monotonic function $f(x) = x\cdot(1-x)$ on the interval $\mathbf{x} = [0,1]$. Computing this function means computing $r_1 := 1-x$ and $r_2 := x\cdot r_1$. Thus, straightforward interval computations lead to $\mathbf{r}_1 := [1,1]-[0,1] = [0,1]$ and $\mathbf{r}_2 := [0,1]\cdot[0,1] = [0,1]$, while the actual range is $[0,0.25]$.

Here, $\mathbf{x} = [\widetilde{x}-\Delta,\widetilde{x}+\Delta]$, with $\widetilde{x} = 0.5$ and $\Delta = 0.5$, and $\dfrac{df}{dx} = 1\cdot(1-x)+x\cdot(-1) = 1-2x$, thus $\dfrac{df}{dx}(\mathbf{x}) = 1-2\cdot[0,1] = [-1,1]$. So, we get the estimate

$$\mathbf{Y} = f(\widetilde{x}) + \frac{df}{dx}(\mathbf{x})\cdot[-\Delta,\Delta] = 0.5\cdot(1-0.5)+[-1,1]\cdot[-0.5,0.5] =$$

$$0.25+[-0.5,0.5] = [-0.25,0.75].$$

The upper bound is better than for straightforward interval computations, but the lower bound is worse. This example shows that we need other ideas.

Third idea: bisection. The centered form is based on the first order formula whose accuracy is $O(\Delta_i^2)$. Thus, if he intervals are too wide, we can split one of them in half – decreasing the error from Δ_i^2 to $\Delta_i^2/4$ – and take the union of the resulting ranges.

For example, for $f(x) = x\cdot(1-x)$ on $x \in \mathbf{x} = [0,1]$, we split into $\mathbf{x}' = [0,0.5]$ and $\mathbf{x}'' = [0.5,1]$. On both subintervals, $f(x)$ is monotonic: e.g., on $\mathbf{x}''$, $1-2\cdot\mathbf{x} =$ $1-2\cdot[0.5,1] = [-1,0]$ with $\underline{r} \geq 0$. So, we get the exact bounds $f(\mathbf{x}') = [f(0),f(0.5)] =$ $[0,0.25]$ and $f(\mathbf{x}'') = [f(1),f(0.5)] = [0,0.25]$, and the exact overall estimate $f(\mathbf{x}')\cup f(\mathbf{x}'') = [0,0.25]$.

Of course, this is a toy example; in practice, we do not always get the exact range, but by combining the above three ideas, we get reasonable estimates for the range.

Alternative approach: affine arithmetic. So far, we compute the range of $x\cdot(1-x)$ by multiplying ranges of x and $1-x$. We ignored the fact that both factors depend on x and are, thus, dependent.

A natural idea is thus, for each intermediate result a, to keep an explicit dependence on $\Delta x_i = \widetilde{x}_i - x_i$ (at least its linear terms). In other words, we represent each intermediate result as $a = a_0 + \sum\limits_{i=1}^{n} a_i\cdot\Delta x_i + [\underline{a},\overline{a}]$. We start with $x_i = \widetilde{x}_i - \Delta x_i$, i.e., with

$$\widetilde{x}_i + 0 \cdot \Delta x_1 + \ldots + 0 \cdot \Delta x_{i-1} + (-1) \cdot \Delta x_i + 0 \cdot \Delta x_{i+1} + \ldots + 0 \cdot \Delta x_n + [0,0],$$

where $a_0 = \widetilde{x}_i$, $a_i = -1$, $a_j = 0$ for $j \neq i$, and $[\underline{a}, \overline{a}] = [0,0]$.

Then, for $a = a_0 + \sum_{i=1}^{n} a_i \cdot \Delta x_i + \mathbf{a}$ and $b = b_0 + \sum_{i=1}^{n} b_i \cdot \Delta x_i + \mathbf{b}$, we get:

- *Addition:* $c_0 = a_0 + b_0$, $c_i = a_i + b_i$, $\mathbf{c} = \mathbf{a} + \mathbf{b}$.
- *Subtraction:* $c_0 = a_0 - b_0$, $c_i = a_i - b_i$, $\mathbf{c} = \mathbf{a} - \mathbf{b}$.
- *Multiplication:* $c_0 = a_0 \cdot b_0$, $c_i = a_0 \cdot b_i + b_0 \cdot a_i$, and

$$\mathbf{c} = a_0 \cdot \mathbf{b} + b_0 \cdot \mathbf{a} + \sum_{i \neq j} a_i \cdot b_j \cdot [-\Delta_i, \Delta_i] \cdot [-\Delta_j, \Delta_j] + \sum_i a_i \cdot b_i \cdot [-\Delta_i, \Delta_i]^2 +$$

$$\left(\sum_i a_i \cdot [-\Delta_i, \Delta_i] \right) \cdot \mathbf{b} + \left(\sum_i b_i \cdot [-\Delta_i, \Delta_i] \right) \cdot \mathbf{a} + \mathbf{a} \cdot \mathbf{b}.$$

For example, for $f(x) = x \cdot (1-x)$ on $x \in [0,1]$, $n = 1$, $\widetilde{x} = 0.5$, and $\Delta = 0.5$. The computer computes $f(x)$ by computing $r_1 := 1 - x$ and $r_2 := x \cdot r_1$. In affine arithmetic, we start with $x = 0.5 - \Delta x + [0,0]$, then compute $\mathbf{r}_1 := 1 - (0.5 - \Delta x) = 0.5 + \Delta x$ and $\mathbf{r}_2 := (0.5 - \Delta x) \cdot (0.5 + \Delta x)$, i.e., $\mathbf{r}_2 = 0.25 + 0 \cdot \Delta x - [-\Delta, \Delta]^2 = 0.25 + [-\Delta^2, 0]$. The resulting range is $\mathbf{y} = 0.25 + [-0.25, 0] = [0, 0.25]$, which is the exact range.

In our simple example, we got the exact range. However, in general, range estimation is NP-hard. This means that any feasible (polynomial-time) algorithm will sometimes lead to excess width: $\mathbf{Y} \supset \mathbf{y}$. In particular, affine arithmetic may lead to excess width. How to get more accurate estimates? One idea is to use bisection. Another idea is that, in addition to linear terms in Δx_i, we also keep quadratic and higher order terms in the Taylor expansion. This *Taylor arithmetic* has indeed been successful, e.g., in collider design.

Interval computations vs. affine arithmetic: comparative analysis. Affine arithmetic is usually more accurate, but it is also slower.

Indeed, in interval computations, for each intermediate result a, we compute two values: endpoints $\underline{a}$ and $\overline{a}$ of $[\underline{a}, \overline{a}]$. In affine arithmetic, for each a, we compute $n+3$ values: a_0, a_1, …, a_n, $\underline{a}$, and $\overline{a}$. Thus, affine arithmetic is $\sim n$ times slower.

Solving systems of equations: extending known algorithms to situations with interval uncertainty. In many practical situations, we need to find the unknowns y_i from the system of equations $g_i(y_1, \ldots, y_n) = a_i$.

Many algorithms $y_j = f_j(a_1, \ldots, a_m)$ are known for the cases when we know the values a_i. In practice, however, we often a_i with interval uncertainty: we only know that $a_i \in [\underline{a}_i, \overline{a}_i]$. In this case, we want to find the corresponding ranges of y_j. A natural idea is to apply interval computations techniques to find the range $f_j([\underline{a}_1, \overline{a}_1], \ldots, [\underline{a}_n, \overline{a}_n])$.

For specific equations, we often already know which ideas work best. For example, for linear equations $Ay = b$, we can use the known fact that y is monotonic in b.

Solving systems of equations when no algorithm is known. The main idea is that we parse each equation into elementary constraints, and then use interval computations to improve original ranges until we get a narrow range (= solution).

First example: the equation $x - x^2 = 0.5$ for $x \in [0,1]$ (this equation has no solution). Parsing leads to $r_1 = x^2$, and $0.5 (= r_2) = x - r_1$.

From $r_1 = x^2$, we extract two rules: (1) $x \to r_1 = x^2$ and (2) $r_1 \to x = \sqrt{r_1}$. From $0.5 = x - r_1$, we extract two more rules: (3) $x \to r_1 = x - 0.5$ and (4) $r_1 \to x = r_1 + 0.5$. We will apply these rules one by one, then again, until we converge.

- We start with $\mathbf{x} = [0,1]$, $\mathbf{r} = (-\infty, \infty)$.
- Rule (1) leads to $\mathbf{r} = [0,1]^2 = [0,1]$, so the new range for r is $\mathbf{r}_{\text{new}} = (-\infty,\infty) \cap [0,1] = [0,1]$.
- Now, Rule (2) leads to $\mathbf{x}_{\text{new}} = \sqrt{[0,1]} \cap [0,1] = [0,1]$ – no change.
- Rule (3) leads to $\mathbf{r}_{\text{new}} = ([0,1] - 0.5) \cap [0,1] = [-0.5, 0.5] \cap [0,1] = [0, 0.5]$.
- Rule (4) leads to $\mathbf{x}_{\text{new}} = ([0, 0.5] + 0.5) \cap [0,1] = [0.5, 1] \cap [0,1] = [0.5, 1]$.
- Rule (1) lead to $\mathbf{r}_{\text{new}} = [0.5, 1]^2 \cap [0, 0.5] = [0.25, 0.5]$.
- Rule (2) leads to $\mathbf{x}_{\text{new}} = \sqrt{[0.25, 0.5]} \cap [0.5, 1] = [0.5, 0.71]$; here, we round $\underline{a}$ down $\downarrow$ and $\overline{a}$ up $\uparrow$, to guarantee enclosure.
- Rule (3) leads to $\mathbf{r}_{\text{new}} = ([0.5, 0.71] - 0.5) \cap [0.25, 5] = [0.0.21] \cap [0.25, 0.5]$, i.e., $\mathbf{r}_{\text{new}} = \emptyset$.

We conclude that the original equation has no solutions.

Another example: equation $x - x^2 = 0$ for $x \in [0,1]$. Parsing leads to $r_1 = x^2$, and $0 = x - r_1$. So, we get the following rules: (1) $r = x^2$; (2) $x = \sqrt{r}$; (3) $r = x$; (4) $x = r$.

We start with $\mathbf{x} = [0,1]$ and $\mathbf{r} = (-\infty, \infty)$. The problem is that after Rule (1), we're stuck with $\mathbf{x} = \mathbf{r} = [0,1]$. A natural solution is thus to bisect $\mathbf{x} = [0,1]$ into $[0, 0.5]$ and $[0.5, 1]$. For the 1st subinterval:

- Rule (1) leads to $\mathbf{r}_{\text{new}} = [0, 0.5]^2 \cap [0, 0.5] = [0, 0.25]$;
- Rule (4) leads to $\mathbf{x}_{\text{new}} = [0, 0.25]$;
- Rule (1) leads to $\mathbf{r}_{\text{new}} = [0, 0.25]^2 = [0, 0.0625]$;
- Rule (4) leads to $\mathbf{x}_{\text{new}} = [0, 0.0625]$; etc.
- we converge to $x = 0$.

For the second subinterval, we converge to $x = 1$.

Optimization: extending known algorithms to situations with interval uncertainty. In optimization, we need to find $y_1, \ldots, y_m$ for which $g(y_1, \ldots, y_m, a_1, \ldots, a_m) \to \max$. In many practical situations, we know a_i with interval uncertainty: $a_i \in [\underline{a}_i, \overline{a}_i]$. We want to find the corresponding ranges of y_j.

Often, for the case of exactly known a_i, we have an algorithm $y_j = f_j(a_1, \ldots, a_n)$ for solving the optimization problem. This is true, e.g., for a quadratic objective function g. In this case, we can apply interval computations techniques to find the range $f_j([\underline{a}_1, \overline{a}_1], \ldots, [\underline{a}_n, \overline{a}_n])$. For specific f, we often already know which ideas work best, and thus, we get an even better solution.

Optimization when no algorithm is known. The main idea is to divide the original box $\mathbf{x}$ into subboxes $\mathbf{b}$. Then, if $\max\limits_{x \in \mathbf{b}} g(x) < g(x')$ for a known x', we dismiss the subbox $\mathbf{b}$.

As an example, let us take $g(x) = x \cdot (1 - x)$ for $\mathbf{x} = [0,1]$. Let's divide $[0,1]$ into 10 subboxes $\mathbf{b} = [0, 0.1], [0.1, 0.2], \ldots$ For each $\mathbf{b}$, we find $g(\widetilde{b})$; the largest is $0.45 \cdot 0.55 = 0.2475$. Then, we compute $G(\mathbf{b}) = g(\widetilde{b}) + (1 - 2 \cdot \mathbf{b}) \cdot [-\Delta, \Delta]$, and dismiss subboxes for which $\overline{Y} < 0.2475$. For example, for $[0.2, 0.3]$, we have

$$0.25 \cdot (1 - 0.25) + (1 - 2 \cdot [0.2, 0.3]) \cdot [-0.05, 0.05].$$

Here, $\overline{Y} = 0.2175 < 0.2475$, so we dismiss $[0.2, 0.3]$. As a result, we keep only boxes $\subseteq [0.3, 0.7]$. Further subdivision will get us closer and closer to $x = 0.5$.

3. An Example Where We Need to Combine Different Approaches to Uncertainty: Chip Design

Formulation of the problem. One of the main problems of computer chip design is to estimate the clock cycle on the design stage. The clock cycle of a chip is constrained by the maximum path delay over all the circuit paths $D \stackrel{\text{def}}{=} \max(D_1, \ldots, D_N)$. The path delay D_i along the i-th path is the sum of the delays corresponding to the gates and wires along this path. Each of these delays, in turn, depends on several factors such as the variation caused by the current design practices, environmental design characteristics (e.g., variations in temperature and in supply voltage), etc.; see, e.g., [12].

Traditional (interval) approach to estimating the clock cycle. The traditional approach assumes that each factor takes the worst possible value. The resulting time delay corresponds to the case when all the factors are at their worst. This is not be a very realistic case: different factors are usually independent, and thus, combination of many worst cases is improbable. As a result, the worst-case estimates are 30% above the observed clock time. Since the clock time is set too high, chips are over-designed and under-performing.

Robust statistical methods are needed. In the ideal case, when we know all the probability distributions, we can use Monte-Carlo simulations to estimate the corresponding uncertainty. In practice, we only have *partial* information about the distributions of some of the parameters. Usually, we know the mean, and we know some characteristic of the deviation from the mean – e.g., the interval that is guaranteed to contain possible values of this parameter.

A possible approach would be to apply Monte-Carlo methods with several possible distributions. The problem is that there are infinitely many possible distributions, and so there is no guarantee that the result of using finitely many of them is a valid bound for all possible distributions. It is therefore desirable to provide *robust* bounds, i.e., bounds that work for all possible distributions.

Towards a mathematical formulation of the problem. Each gate delay d depends on the difference $x_1, \ldots, x_n$ between the actual and the nominal values of the parameters. These differences are usually small, so we can safely ignore terms which are quadratic and higher order in x_i and thus assume that d is a linear function of x_i. Each path delay D_i is the sum of gate delays, thus D_i is also a linear function of x_j: $D_i = a_i + \sum_{j=1}^{n} a_{ij} \cdot x_j$ for some a_i and a_{ij}. The desired maximum delay $D = \max_i D_i$ is thus the maximum of linear functions.

It is known that every maximum of linear functions is convex. Thus, the dependence $D = F(x_1, \ldots, x_n)$ is convex.

We know that the factors x_i are independent. We know the distributions of some of the factors. For others, we know ranges $[\underline{x}_j, \overline{x}_j]$ and means E_j. We are given the largest allowed probability $\varepsilon > 0$ of the fault. We need to find the smallest y_0 for which, for all possible distributions, we have $y \le y_0$ with the probability $\ge 1 - \varepsilon$.

Additional property: dependency is non-degenerate. Sometimes, we learn additional information about one of the factors x_j. For example, we learn that x_j actually belongs to a proper subinterval of the original interval $[\underline{x}_j, \overline{x}_j]$. As a result, the original class $\mathscr{P}$ of possible distributions is replaced with a smaller one $\mathscr{P}' \subset \mathscr{P}$. In this case, the new value y_0' can only decrease: $y_0' \le y_0$.

If x_j is irrelevant for y, then $y_0' = y_0$. Our assumption is that irrelevant variables have been weeded out. In this case, if we narrow down one of the intervals $[\underline{x}_j, \overline{x}_j]$, the resulting value y_0 decreases: $y_0' < y_0$.

Formulation of the problem in precise terms.

GIVEN:

- n, $k \le n$, $\varepsilon > 0$;
- a convex function $y = F(x_1, \ldots, x_n) \ge 0$;
- $n - k$ cdfs $F_j(x)$, $k+1 \le j \le n$;
- intervals $\mathbf{x}_1, \ldots, \mathbf{x}_k$, values $E_1, \ldots, E_k$,

TAKE: all joint probability distributions on R^n for which:

- all x_i are independent,
- $x_j \in \mathbf{x}_j$, $E[x_j] = E_j$ for $j \le k$, and
- x_j have distribution $F_j(x)$ for $j > k$.

FIND: the smallest y_0 s.t. for all such distributions, $F(x_1, \ldots, x_n) \le y_0$ with probability $\ge 1 - \varepsilon$.

WHEN: the problem is *non-degenerate* – if we narrow down one of the intervals $\mathbf{x}_j$, y_0 decreases.

Main result and how we can use it. Our main result is that y_0 is attained when for each j from 1 to k, we take $x_j = \underline{x}_j$ with probability $\underline{p}_j \stackrel{\text{def}}{=} \dfrac{\overline{x}_j - E_j}{\overline{x}_j - \underline{x}_j}$, and $x_j = \overline{x}_j$ with the remaining probability $\overline{p}_j \stackrel{\text{def}}{=} \dfrac{E_j - \underline{x}_j}{\overline{x}_j - \underline{x}_j}$. (The proof is given in the Appendix.)

Thus, we arrive at the following algorithm:

- simulate these distributions for x_j, $j < k$;
- simulate known distributions for $j > k$;
- use the simulated values $x_j^{(s)}$ to find $y^{(s)} = F(x_1^{(s)}, \ldots, x_n^{(s)})$;
- sort N values $y^{(s)}$: $y_{(1)} \le y_{(2)} \le \ldots \le y_{(N_i)}$;
- take $y_{(N_i \cdot (1-\varepsilon))}$ as y_0.

Comment about Monte-Carlo techniques. It is usually believed that Monte-Carlo methods are inferior to analytical: they are approximate, they require large computation time; and simulations for *several* distributions, may mis-calculate the (desired) maxi-

mum over *all* distributions. In our case, however, the value corresponding to the selected distributions indeed provide the desired maximum value y_0. This is a particular case of the general fact that justified Monte-Carlo methods often lead to *faster* computations than analytical techniques a good example is multi-D integration, where Monte-Carlo methods were originally invented.

4. Combining Interval and Probabilistic Uncertainty: General Case

Need for such a combination. Chip design is just one of the examples of the practical situations in which some information comes in probabilistic form and some in interval form. So, we need to learn how to combine these two types of uncertainty.

How to represent a probability distribution. There are many ways to represent a probability distribution. A natural idea is to look for an objective. The objective of processing uncertain data is to make decisions, and it is known that rational decision making is equivalent to maximizing the expected values of an appropriate function $u(x,a)$ called *utility*: $E_x[u(x,a)] \to \max_a$; see, e.g., [2,7,9,14].

In many practical situations, the function $u(x)$ is smooth. We can therefore expand $u(x)$ in Taylor series and keep only the first few terms: we have $u(x) = u(x_0) + (x-x_0)\cdot u'(x_0) + \ldots$ Then, to find the expected value, it is enough to know the moments $E[(x-x_0)^k]$. If we do not know the exact moments, then we should know (interval) bounds on moments.

Sometimes, we have a threshold-type $u(x)$: e.g., when a fine is imposed if the concentration of a certain pollutants exceeds a certain threshold. In this case, to compute $E[u]$, we need to know cdf $F(x) = \text{Prob}(\xi \le x)$. In the case of uncertainty, we have bounds $[\underline{F}(x), \overline{F}(x)]$; such bounds are known as *p-boxes*.

Extension of interval arithmetic to probabilistic case: successes and challenges. To process the corresponding uncertainty, it is reasonable to parse the original algorithm into a sequence of elementary operations $+$, $-$, $\cdot$, $1/x$, max, min, and then use the formulas for these elementary operations.

Explicit formulas are known for intervals, for p-boxes, and for intervals + 1st moments E_i. In some cases, these formulas are easy to get, e.g., for $+$, $-$, and for product of independent x_i; see, e.g., [1,6,10]. For multiplication $y = x_1 \cdot x_2$, when we have no information about the correlation, we get non-trivial formulas:

- $\underline{E} = \max(p_1 + p_2 - 1, 0)\cdot \overline{x}_1 \cdot \overline{x}_2 + \min(p_1, 1-p_2)\cdot \overline{x}_1 \cdot \underline{x}_2 + \min(1-p_1, p_2)\cdot \underline{x}_1 \cdot \overline{x}_2 + \max(1-p_1-p_2, 0)\cdot \underline{x}_1 \cdot \underline{x}_2$;
- $\overline{E} = \min(p_1, p_2)\cdot \overline{x}_1 \cdot \overline{x}_2 + \max(p_1 - p_2, 0)\cdot \overline{x}_1 \cdot \underline{x}_2 + \max(p_2 - p_1, 0)\cdot \underline{x}_1 \cdot \overline{x}_2 + \min(1-p_1, 1-p_2)\cdot \underline{x}_1 \cdot \underline{x}_2$,

where $p_i \stackrel{\text{def}}{=} (E_i - \underline{x}_i)/(\overline{x}_i - \underline{x}_i)$.

The main remaining challenges are to deal with situations when we know intervals + 1st and 2nd moments, or when we now moments + p-boxes. In some cases, such formulas are known. Let us describe one of these cases.

5. Case Study: Bioinformatics

Practical problem. It is important to find genetic difference between cancer cells and healthy cells. Ideal solution to this problem is to directly measure concentrations c of the gene in cancer cells and h in healthy cells. In reality, such cells are difficult to separate. Thus, what we really measure is not c, but $y_i \approx x_i \cdot c + (1-x_i) \cdot h$, where x_i is the percentage of cancer cells in i-th sample. This amount can be described as $a \cdot x_i + h \approx y_i$, where $a \stackrel{\text{def}}{=} c - h$.

Ideal case. If we know x_i exactly, we can use the Least Squares Method $\sum_{i=1}^{n} (a \cdot x_i + h - y_i)^2 \to \min_{a,h}$. We then find $a = \dfrac{C(x,y)}{V(x)}$ and $h = E(y) - a \cdot E(x)$, where $E(x) = \dfrac{1}{n} \cdot \sum_{i=1}^{n} x_i$, $V(x) = \dfrac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - E(x))^2$, and $C(x,y) = \dfrac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - E(x)) \cdot (y_i - E(y))$.

Real-life case. In practice, experts manually count x_i, and only provide interval bounds $\mathbf{x}_i$, e.g., $x_i \in [0.7, 0.8]$. We then need to find the range of a and h corresponding to all possible values $x_i \in [\underline{x}_i, \overline{x}_i]$.

Solving the problem. In general, we know intervals $\mathbf{x}_1 = [\underline{x}_1, \overline{x}_1], \ldots, \mathbf{x}_n = [\underline{x}_n, \overline{x}_n]$, and we need to compute the range of $E(x)$, $V(x)$, etc. In general, this problem is NP-hard even for the population variance $V(x)$ [10]. However, there are efficient algorithms for $\overline{V}$ and $C(x,y)$ for reasonable situations; see, e.g., [10]. So, to solve our problem, we use these algorithms to find intervals for $C(x,y)$ and for $V(x)$ and then divide these intervals.

6. Another Case Study: Detecting Outliers

Outlier detection is important. In many application areas, it is important to detect *outliers*, i.e., unusual, abnormal values: in *medicine*, unusual values may indicate disease; in *geophysics*, abnormal values may indicate a mineral deposit (or an erroneous measurement result); in *structural integrity* testing, abnormal values may indicate faults in a structure, etc.

The traditional engineering approach is that a new measurement result x is classified as an outlier if $x \notin [L, U]$, where $L \stackrel{\text{def}}{=} E - k_0 \cdot \sigma$ and $U \stackrel{\text{def}}{=} E + k_0 \cdot \sigma$, and $k_0 > 1$ is preselected. Most frequently, $k_0 = 2$, 3, or 6.

Outlier detection under interval uncertainty. In many practical situations, we only have intervals $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$. Different values $x_i \in \mathbf{x}_i$ lead to different intervals $[L, U]$. We can therefore distinguish between *possible* and *guaranteed* outliers. A possible outlier is when x is outside *some* k_0-sigma interval. Finding possible outliers is important in structural integrity – so as not to miss a fault. A *guaranteed* outlier is a value which is outside *all* k_0-sigma intervals. For example, before a breast surgery, we want to make sure that there is a micro-calcification.

One can easily check that x is a possible outlier if $x \notin [\overline{L}, \underline{U}]$, and x is a guaranteed outlier if $x \notin [\underline{L}, \overline{U}]$. So, to detect outliers, we must find the ranges $[\underline{L}, \overline{L}]$ and $[\underline{U}, \overline{U}]$ of $L = E - k_0 \cdot \sigma$ and $U = E + k_0 \cdot \sigma$. Algorithms for computing such ranges have indeed been proposed; see, e.g., [10].

7. Fuzzy Uncertainty

Need for fuzzy uncertainty. Knowledge often comes from experts, and expert usually describe their knowledge by using imprecise ("fuzzy") words from natural language such as small. To describe this knowledge in precise terms, a special technique called *fuzzy* was invented; see, e.g., [4,11,15]. In this techniques, for each possible value x of the corresponding quantity, we ask the expert to estimate the degree $\mu(x)$ to which this value satisfies the appropriate property (i.e., to what extent this value is small). The expert can do it, e.g., by marking a value on a scale from 0 to 10; if the expert marks 7, we take $\mu(x) = 7/10$. The resulting function $\mu(x)$ is known as a *membership function* or a *fuzzy set*.

Data processing under fuzzy uncertainty. We have a data processing algorithm $y = f(x_1, \ldots, x_n)$ and n fuzzy sets $\mu_i(x_i)$. We need to come up with a fuzzy set $\mu(y)$ that describes the resulting uncertainty about y.

To come up with the appropriate formulas, let us take into account that y is a possible value of the possible value of the corresponding quantity Y if and only if there exist numbers $x_1, \ldots, x_n$ such that each x_i is a possible value of X_i and $f(x_1, \ldots, x_n) = y$. We know the degrees $\mu_i(x_i)$ to which each x_i is a possible value of X_i.

So, if use $\min(\mu(A), \mu(B))$ to describe our degree of belief in $A \& B$, and $\max(d(A), d(B))$ for $A \vee B$, we conclude that

$$\mu(y) = \max_{x_1, \ldots, x_n : f(x_1, \ldots, x_n) = y} \min(\mu_1(x_1), \ldots, \mu_n(x_n)).$$

This formula is known as *Zadeh's extension principle.*

Reduction to interval computations. How can we compute $\mu(y)$"? A direct optimization would be computationally expensive. Good new is that such computations can be simplified if we represent each fuzzy set μ_i by its *α-cuts* $X_i(\alpha) \stackrel{\text{def}}{=} \{x_i : \mu_i(x_i) \geq \alpha\}$. It turns out that for continuous f, for every α, the α-cut $Y(\alpha)$ is equal to the range of f on α-cuts of X_i: $Y(\alpha) = f(X_1(\alpha), \ldots, X_n(\alpha))$.

Thus, to compute $\mu(y)$, we can simply, for $\alpha = 0, 0.1, 0.2, \ldots, 1$ apply interval computations techniques to compute $Y(\alpha)$.

References

[1] S. Ferson, V. Kreinovich, L. Ginzburg, D.S. Myers, and K. Sentz, *Constructing Probability Boxes and Dempster-Shafer Structures*, Sandia National Laboratories, Report SAND2002-4015, January 2003.

[2] P.C. Fishburn, *Utility Theory for Decision Making*, John Wiley & Sons Inc., New York, 1969.

[3] L. Jaulin, M. Kiefer, O. Dicrit, and E. Walter, *Applied Interval Analysis*, Springer, London, 2001.

[4] G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.

[5] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[6] V. Kreinovich, Interval computations and interval-related statistical techniques: tools for estimating uncertainty of the results of data processing and indirect measurements, In: F. Pavese and A.B. Forbes (eds.), *Data Modeling for Metrology and Testing in Measurement Science*, Birkhauser-Springer, Boston, 2009, pp. 117-145.

[7] R.D. Luce and R. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.
[8] R.E. Moore, R.B. Kearfott, and M.J. Cloud, *Introduction to Interval Analysis*, SIAM, Philadelphia, 2009.
[9] H.T. Nguyen, O. Kosheleva, and V. Kreinovich, Decision making beyond Arrow's 'impossibility theorem', with the analysis of effects of collusion and mutual attraction, *International Journal of Intelligent Systems* **24**(1) (2009) 27–47.
[10] H.T. Nguyen, V. Kreinovich, B. Wu, and G. Xiang, *Computing Statistics under Interval and Fuzzy Uncertainty*, Springer Verlag, Berlin, Heidelberg, 2012.
[11] H.T. Nguyen and E.A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.
[12] M. Orshansky, W.-S. Wang, G. Xiang, and V. Kreinovich, Interval-based robust statistical techniques for non-negative convex functions, with application to timing analysis of computer chips", *Proceedings of the Second International Workshop on Reliable Engineering Computing REC'2006*, Savannah, Georgia, February 22-24, 2006, pp. 197-212.
[13] S.G. Rabinovich, *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, Berlin, 2005.
[14] H. Raiffa, *Decisio n Analysis*, Addison-Wesley, Reading, Massachusetts, 1970.
[15] L.A. Zadeh, Fuzzy sets, *Information and Control* **8**(1965) 338–353.

A. Proof of the Result about Chips: Main Idea

Let us fix the optimal distributions for $x_2,\dots,x_n$; then,

$$\text{Prob}(D \le y_0) = \sum_{(x_1,\dots,x_n):D(x_1,\dots,x_n)\le y_0} p_1(x_1)\cdot p_2(x_2)\cdot\dots$$

So, $\text{Prob}(D \le y_0) = \sum\limits_{i=0}^{N} c_i \cdot q_i$, where v_i are possible values of x_1, p_i is the probability of v_i, and c_i do not depend on x_1. The restrictions on q_i are: $q_i \ge 0$, $\sum\limits_{i=0}^{N} q_i = 1$, and $\sum\limits_{i=0}^{N} q_i \cdot v_i = E_1$. Thus, the worst-case distribution for x_1 is a solution to the following linear programming (LP) problem:

$$\text{Minimize } \sum_{i=0}^{N} c_i \cdot q_i \text{ under the constraints } \sum_{i=0}^{N} q_i = 1 \text{ and } \sum_{i=0}^{N} q_i \cdot v_i = E_1,$$
$$q_i \ge 0, \quad i = 0,1,2,\dots,N.$$

It is known that in LP with $N+1$ unknowns $q_0, q_1, \dots, q_N$, $\ge N+1$ constraints are equalities. In our case, we have 2 equalities, so at least $N-1$ constraints $q_i \ge 0$ are equalities. Hence, no more than 2 values q_i are non-0.

If the corresponding values v or v' of x_1 – for which the probability is non-zero – are in $(\underline{x}_1, \overline{x}_1)$, then for $[v,v'] \subset \mathbf{x}_1$ we get the same y_0 – in contradiction to non-degeneracy. Thus, the worst-case distribution is located at $\underline{x}_1$ and $\overline{x}_1$. The condition that the mean of x_1 is E_1 leads to the desired formulas for $\underline{p}_1$ and $\overline{p}_1$. Similarly, we get formulas for all other values $\underline{p}_i$ and $\overline{p}_i$. The statement is proven.