9-1-2011

No-Free-Lunch Result for Interval and Fuzzy Computing: When Bounds Are Unusually Good, Their Computation is Unusually Slow

Ildar Batyrshin
Instituto Mexicano de Petróleo

Martine Ceberio
University of Texas at El Paso, mceberio@utep.edu

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Comments:
Technical Report: UTEP-CS-11-31b
To appear in: I. Batyrshin and G. Sidorov (eds.), *Proceedings of the 10th Mexican International Conference on Artificial Intelligence MICAI'2011*, Puebla, Mexico, November 26 - December 4, 2011, Springer Lecture Notes in Artificial Intelligence, Vol. 7095, pp. 13-23.

No-Free-Lunch Result for Interval and Fuzzy Computing: When Bounds Are Unusually Good, Their Computation is Unusually Slow

Martine Ceberio and Vladik Kreinovich

University of Texas at El Paso, Computer Science Dept., El Paso, TX 79968, USA
mceberio@utep.edu, vladik@utep.edu

Abstract. On several examples from interval and fuzzy computations and from related areas, we show that when the results of data processing are unusually good, their computation is unusually complex. This makes us think that there should be an analog of Heisenberg's uncertainty principle well known in quantum mechanics: when we an unusually beneficial situation in terms of results, it is not as perfect in terms of computations leading to these results. In short, nothing is perfect.

1 First Case Study: Interval Computations

Need for data processing. In science and engineering, we want to *understand* how the world works, we want to *predict* the results of the world processes, and we want to *design* a way to control and change these processes so that the results will be most beneficial for the humankind.

For example, in meteorology, we want to know the weather now, we want to predict the future weather, and – if, e.g., floods are expected, we want to develop strategies that would help us minimize the flood damage.

Usually, we know the equations that describe how these systems change in time. Based on these equations, engineers and scientists have developed algorithms that enable them to predict the values of the desired quantities – and find the best values of the control parameters. As input, these algorithms take the current and past values of the corresponding quantities.

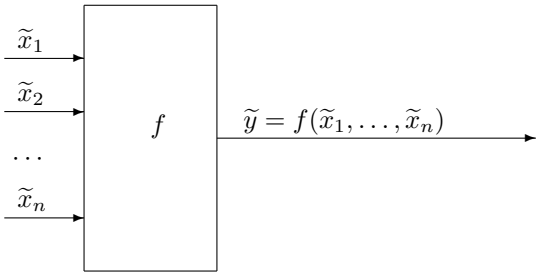
For example, if we want to predict the trajectory of the spaceship, we need to find its current location and velocity, the current position of the Earth and of the celestial bodies, then we can use Newton's equations to find the future locations of the spaceship.

In many situations – e.g., in weather prediction – the corresponding computations require a large amount of input data and a large amount of computations steps. Such computations (*data processing*) are the main reason why computers were invented in the first place – to be able to perform these computations in reasonable time.

Need to take input uncertainty into account. In all the data processing tasks, we start with the current and past values $x_1, \ldots, x_n$ of some quantities, and we use a known algorithm $f(x_1, \ldots, x_n)$ to produce the desired result $y = f(x_1, \ldots, x_n)$.

The values x_i come from measurements, and measurements are never absolutely accurate: the value $\widetilde{x}_i$ that we obtained from measurement is, in general, different from the actual (unknown) value x_i of the corresponding quantity. For example, if the clock shows 12:20, it does not mean that the time is *exactly* 12 hours, 20 minutes and 00.0000 seconds: it may be a little earlier or a little later than that.

As a result, in practice, we apply the algorithm f not to the actual values x_i, but to the *approximate* values $\widetilde{x}_i$ that come from measurements:



So, instead of the ideal value $y = f(x_1, \ldots, x_n)$, we get an approximate value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$. A natural question is: how do approximation errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ affect the resulting error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$? Or, in plain words, how to take input uncertainty into account in data processing?

From probabilistic to interval uncertainty. [18] Manufacturers of the measuring instruments provide us with bounds Δ_i on the (absolute value of the) measurement errors: $|\Delta x_i| \le \Delta_i$. If now such upper bound is known, then the device is *not* a measuring instrument.

For example, a street thermometer may show temperature that is slightly different from the actual one. Usually, it is OK if the actual temperature is $+24$ but the thermometer shows $+22$ – as long as the difference does not exceed some reasonable value Δ. But if the actual temperature is $+24$ but the thermometer shows -5, any reasonable person would return it to the store and request a replacement.

Once we know the measurement result $\widetilde{x}_i$, and we know the upper bound Δ_i on the measurement error, we can conclude that the actual (unknown) value x_i belongs to the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. For example, if the measured temperature is $\widetilde{x}_i = 22$, and the manufacturer guarantees the accuracy $\Delta_i = 3$, this means that the actual temperature is somewhere between $\widetilde{x}_i - \Delta_i = 22 - 3 = 19$ and $\widetilde{x}_i + \Delta_i = 22 + 3 = 25$.

Often, in addition to these bounds, we also know the *probabilities* of different possible values Δx_i within the corresponding interval $[-\Delta_i, \Delta_i]$. This is how

uncertainty is usually handled in engineering and science – we assume that we know the probability distributions for the measurement errors Δx_i (in most cases, we assume that this distribution is normal), and we use this information to describe the probabilities of different values of Δy. However, there are two important situations when we do not know these probabilities:

- cutting-edge measurements, and
- cutting-cost manufacturing.

Indeed, how do we determine the probabilities? Usually, to find the probabilities of different values of the measurement error $\Delta x_i = \widetilde{x}_i - x_i$, we bring our measuring instrument to a lab that has a "standard" (much more accurate) instrument, and compare the results of measuring the same quantity with two different instruments: ours and a standard one. Since the standard instrument is much more accurate, we can ignore its measurement error and assume that the value X_i that it measures is the actual value: $X_i \approx x_i$. Thus, the difference $\widetilde{x}_i - X_i$ between the two measurement results is practically equal to the measurement error $\Delta x_i = \widetilde{x}_i - x_i$. So, when we repeat this process several times, we get a histogram from which we can find the probability distribution of the measurement errors.

However, in the above two situations, this is not done. In the case of cutting-edge measurements, this is easy to explain. For example, if we want to estimate the measurement errors of the measurement performed by a Hubble space telescope (or by the newly built CERN particle collider), it would be nice to have a "standard", five times more accurate telescope floating nearby – but Hubble is the best we have. In manufacturing, in principle, we can bring every single sensor to the National Institute of Standards and determine its probability distribution – but this would cost a lot of money: most sensors are very cheap, and their "calibration" using the expensive super-precise "standard" measuring instruments would cost several orders of magnitude more. So, unless there is a strong need for such calibration – e.g., if we manufacture a spaceship – it is sufficient to just use the upper bound on the measurement error.

In both situations, after the measurements, the only information that we have about the actual value of x_i is that this value belongs to the interval $[\underline{x}_i, \overline{x}_i] = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

Different possible values x_i from the corresponding intervals lead, in general, to different values of $y = f(x_1, \ldots, x_n)$. It is therefore desirable to find the range of all possible values of y, i.e., the set

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

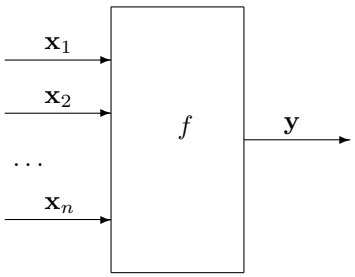
(Since the function $f(x_1, \ldots, x_n)$ is usually continuous, its range is the interval.) Thus, we arrive at the same *interval computations* problem; see, e.g., [6, 7, 15].

The main problem. We are given:

- an integer n;
- n intervals $\mathbf{x}_1 = [\underline{x}_1, \overline{x}_1]$, ..., $\mathbf{x}_n = [\underline{x}_n, \overline{x}_n]$, and
- an algorithm $f(x_1, \ldots, x_n)$ which transforms n real numbers into a real number $y = f(x_1, \ldots, x_n)$.

We need to compute the endpoints $\underline{y}$ and $\overline{y}$ of the interval

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



In general, the interval computations problem is NP-hard. It is known that in general, the problem of computing the exact range $\mathbf{y}$ is NP-hard; see, e.g., [13]. Moreover, it is NP-hard even if we restrict ourselves to quadratic functions $f(x_1, \ldots, x_n)$ – even to the case when we only consider a very simple quadratic function: a sample variance [2, 3]:

$$f(x_1, \ldots, x_n) = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n} \cdot \sum_{i=1}^{n} x_i \right)^2.$$

NP-hard means, crudely speaking, that it is not possible to have an algorithm that would always compute the exact range in reasonable time.

Case of small measurement errors. In many practical situations, the measurement errors are relatively small, i.e., we can safely ignore terms which are quadratic or higher order in terms of these errors. For example, if the measurement error is 10%, its square is 1% which is much smaller than 10%. In such situations, it is possible to have an efficient algorithm for computing the desired range.

Indeed, in such situations, we can simplify the expression for the desired error

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n) =$$

$$f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n)$$

if we expand the function f in Taylor series around the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and restrict ourselves only to linear terms in this expansion. As a result, we get the expression

$$\Delta y = c_1 \cdot \Delta x_1 + \ldots + c_n \cdot \Delta x_n,$$

where by c_i, we denoted the value of the partial derivative $\partial f/\partial x_i$ at the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$:

$$c_i = \frac{\partial f}{\partial x_i}|_{(\widetilde{x}_1, \ldots, \widetilde{x}_n)}.$$

In the case of interval uncertainty, we do not know the probability of different errors Δx_i; instead, we only know that $|\Delta x_i| \le \Delta_i$. In this case, the above sum attains its largest possible value if each term $c_i \cdot \Delta x_i$ in this sum attains the largest possible value:

- If $c_i \ge 0$, then this term is a monotonically non-decreasing function of Δx_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 $c_i \cdot \Delta_i$.
- If $c_i < 0$, then this term is a decreasing function of Δx_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 $-c_i \cdot \Delta_i = |c_i| \cdot \Delta_i$.

In both cases, the largest possible value of this term is $|c_i| \cdot \Delta_i$, so, the largest possible value of the sum Δy is

$$\Delta = |c_1| \cdot \Delta_1 + \ldots + |c_n| \cdot \Delta_n.$$

Similarly, the smallest possible value of Δy is $-\Delta$.

Hence, the interval of possible values of Δy is $[-\Delta, \Delta]$, with Δ defined by the above formula.

How do we compute the derivatives? If the function f is given by its analytical expression, then we can simply explicitly differentiate it, and get an explicit expression for its derivatives. This is the case which is typically analyzed in textbooks on measurement theory; see, e.g., [18].

In many practical cases, we do not have an explicit analytical expression, we only have an *algorithm* for computing the function $f(x_1, \ldots, x_n)$, an algorithm which is too complicated to be expressed as an analytical expression.

When this algorithm is presented in one of the standard programming languages such as Fortran or C, we can apply one of the existing analytical differentiation tools (see, e.g., [5]), and automatically produce a program which computes the partial derivatives c_i. These tools analyze the code and produce the differentiation code as they go.

In many other real-life applications, an algorithm for computing $f(x_1, \ldots, x_n)$ may be written in a language for which an automatic differentiation tool is not available, or a program is only available as an executable file, with no source code at hand. In such situations, when we have no easy way to analyze the code, the only thing we can do is to take this program as a *black box*: i.e., to apply it

to different inputs and use the results of this application to compute the desired value Δ. Such black-box methods are based on the fact that, by definition, the derivative is a limit:

$$c_i = \frac{\partial f}{\partial x_i} =$$

$$\lim_{h \to 0} \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)}{h}.$$

By definition, a limit means that when h is small, the right-hand side expression is close to the derivative – and the smaller h, the closer this expression to the desired derivative. Thus, to find the derivative, we can use this expression for some small h:

$$c_i \approx \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)}{h}.$$

To find all n partial derivatives c_i, we need to call the algorithm for computing the function $f(x_1, \ldots, x_n)$ $n + 1$ times:

- one time to compute the original value $f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$ and
- n times to compute the perturbed values $f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$ for $i = 1, 2, \ldots, n$.

So:

- if the algorithm for computing the function $f(x_1, \ldots, x_n)$ is feasible, finishes its computations in polynomial time T_f, i.e., in time which is bounded by a polynomial of the size n of the input,
- then the overall time needed to compute all n derivatives c_i is bounded by $(n + 1) \cdot T_f$ and is, thus, also polynomial – i.e., feasible.

Cases when the resulting error is unusually small. In general, the resulting approximation error Δ is a linear function of the error bounds $\Delta_1, \ldots, \Delta_n$ on individual (direct) measurements. In other words, the resulting approximation error is of the same order as the original bounds Δ_i. In this general case, the above technique (or appropriate faster techniques; see, e.g., [9, 19]) provide a good estimate for Δ, an estimate with an absolute accuracy of order Δ_i^2 and thus, with a relative accuracy of order Δ_i.

There are usually good cases, when all (or almost all) linear terms in the linear expansion disappear: when the derivatives $c_i = \dfrac{\partial f}{\partial x_i}$ are equal to 0 (or close to 0) at the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$. In this case, to estimate Δ, we must consider next terms in Taylor expansion, i.e., terms which are quadratic in Δ_i:

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n) =$$

$$f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n) =$$

$$f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - \left(f(\widetilde{x}_1, \ldots, \widetilde{x}_n) + \frac{1}{2} \cdot \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{\partial^2 f}{\partial x_i \partial x_j} \cdot \Delta x_i \cdot \Delta x_j + \ldots \right) =$$

$$-\frac{1}{2}\cdot\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial^2 f}{\partial x_i\partial x_j}\cdot\Delta x_i\cdot\Delta x_j+\ldots$$

As a result, in such situations, the resulting approximation error is unusually small – it is proportional to Δ_i^2 instead of Δ_i. For example, when the measurement accuracy is $\Delta_i\approx 10\%$, usually, we have Δ of the same order 10%, but in this unusually good case, the approximation accuracy is of order $\Delta_i^2\approx 1\%$ – an order of magnitude better.

When bounds are unusually good, their computation is unusually slow. In the above case, estimating Δ means solving an interval computations problem (of computing the range of a given function on given intervals) for a quadratic function $f(x_1,\ldots,x_n)$. We have already mentioned that, in contrast to the linear case when we have an efficient algorithm, the interval computation problem for quadratic functions is NP-hard. Thus, when bounds are unusually small, their computation is an unusually difficult task.

Discussion. The above observation us think that there should be an analog of Heisenberg's uncertainty principle (well known in quantum mechanics):

- when we an unusually beneficial situation in terms of results,
- it is not as perfect in terms of computations leading to these results.

In short, nothing is perfect.

Comment. Other examples – given below – seem to confirm this conclusion.

2 Second Case Study: Fuzzy Computations

Need for fuzzy computations. In some cases, in addition to (and/or instead of) measurement results x_i, we have expert estimates for the corresponding quantities. These estimates are usually formulated by using words from natural language, like "about 10". A natural way to describe such expert estimates is to use fuzzy techniques (see, e.g., [8, 17]), i.e., to describe each such estimate as a *fuzzy number* X_i – i.e., as a function $\mu_i(x_i)$ that assigns, to each possible value x_i, a degree to which the expert is confident that this value is possible. This function is called a *membership function*.

Fuzzy data processing. When each input x_i is described by a fuzzy number X_i, i.e., by a membership function $\mu_i(x_i)$ that assigns, to every real number x_i, a degree to which this number is possible as a value of the i-th input, we want to find the fuzzy number Y that describes $f(x_1,\ldots,x_n)$. A natural way to define the corresponding membership function $\mu(y)$ leads to Zadeh's extension principle:

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

Fuzzy data processing can be reduced to interval computations. It is known that from the computational viewpoint, the application of this formula can be reduced to interval computations.

Specifically, for each fuzzy set with a membership function $\mu(x)$ and for each $\alpha \in (0, 1]$, we can define this set's *α-cut* as $\mathcal{X}(\alpha) \stackrel{\text{def}}{=} \{x : \mu(x) \geq \alpha\}$. Vice versa, if we know the α-cuts for all α, we, for each x, can reconstruct the value $\mu(x)$ as the largest value α for which $x \in \mathcal{X}(\alpha)$. Thus, to describe a fuzzy number, it is sufficient to find all its α-cuts.

It is known that when the inputs $\mu_i(x_i)$ are fuzzy numbers, and the function $y = f(x_1, \ldots, x_n)$ is continuous, then for each α, the α-cut $\mathcal{Y}(\alpha)$ of y is equal to the range of possible values of $f(x_1, \ldots, x_n)$ when $x_i \in \mathcal{X}_i(\alpha)$ for all i:

$$\mathcal{Y}(\alpha) = f(\mathcal{X}_1(\alpha), \ldots, \mathcal{X}_n(\alpha)) = \{f(x_1, \ldots, x_n) : x_1 \in \mathcal{X}_1(\alpha), \ldots, x_n \in \mathcal{X}_n\};$$

see, e.g., [1, 8, 16, 17]. So, if we know how to solve our problem under interval uncertainty, we can also solve it under fuzzy uncertainty – e.g., by repeating the above interval computations for $\alpha = 0, 0.1, \ldots, 0.9, 1.0$.

When bounds are unusually good, their computation is unusually slow. Because of the above reduction, the conclusion about interval computations can be extended to fuzzy computations:

- when the resulting bounds are unusually good,
- their computation is unusually difficult.

3 Third Case Study: When Computing Variance under Interval Uncertainty Is NP-Hard

Computing the range of variance under interval uncertainty is NP-hard: reminder. The above two examples are based on the result that computing the range of a quadratic function under interval uncertainty is NP-hard. Actually, as we have mentioned, even computing the range $[\underline{V}, \overline{V}]$ of the variance $V(x_1, \ldots, x_n)$ on given intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is NP-hard [2, 3]. Specifically, it turns out that while the lower endpoint $\underline{V}$ can be computed in polynomial time, computing the upper endpoint $\overline{V}$ is NP-hard.

Let us move analysis deeper. Let us check when we should expect the most beneficial situation – with small $\overline{V}$ – and let us show that in this case, computing $\overline{V}$ is the most difficult task.

When we can expect the variance to be small. By definition, the variance $V = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - E)^2$ describes the average deviation of its values from the mean $E = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$. The smallest value of the variance V is attained when all the

values from the sample are equal to the mean E, i.e., when all the values in the sample are equal $x_1 = \ldots = x_n$.

In the case of interval uncertainty, it is thus natural to expect that the variance is small if it is possible that all values x_i are equal, i.e., if all n intervals $\mathbf{x}_1$, $\ldots$, $\mathbf{x}_n$ have a common point.

In situations when we expect small variance, its computation is unusually slow. Interestingly, NP-hardness is proven, in [2, 3], exactly on the example of n intervals that all have a common intersection – i.e., on the example when we should expect the small variance.

Moreover, if the input intervals do not have a common non-empty intersection – e.g., if there is a value C for which every collection of C intervals have an empty intersection – then it is possible to have a feasible algorithm for computing the range of the variance [2–4, 10–12].

Discussion. Thus, we arrive at the same conclusion as in the above cases:

- when we an unusually beneficial situation in terms of results,
- it is not as perfect in terms of computations leading to these results.

4 Fourth Case Study: Kolmogorov Complexity

Need for Kolmogorov complexity. In many application areas, we need to compress data (e.g., an image). The original data can be, in general, described as a string x of symbols. What does it mean to compress a sequence? It means that instead of storing the original sequence, we store a compressed data string and a program describing how to un-compress the data. The pair consisting of the data and the un-compression program can be viewed as a single program p which, when run, generates the original string x. Thus, the quality of a compression can be described as the length of the shortest program p that generates x. This shortest length is known as *Kolmogorov complexity* $K(x)$ of the string x; see, e.g., [14]:

$$K(x) \stackrel{\text{def}}{=} \min\{\text{len}(p) : p \text{ generates } x\}.$$

In unusually good situations, computations are unusually complex. The smaller the Kolmogorov complexity $K(x)$, the more we can compress the original sequence x. It turns out (see, e.g., [14]) that, for most strings, the Kolmogorov complexity $K(x)$ is approximately equal to their length – and can, thus, be efficiently computed (as long as we are interested in the approximate value of $K(x)$, of course). These strings are what physicists would call *random*.

However, there are strings which are not random, strings which can be drastically compressed. It turns out that computing $K(x)$ for such strings is difficult: there is no algorithm that would, given such a string x, compute its Kolmogorov complexity (even approximately) [14]. This result confirms our general conclusion that:

- when situations are unusually good,
- computations are unusually complex.

Acknowledgments. This work was supported in part by the National Science Foundation grants HRD-0734825 and DUE-0926721 and by Grant 1 T36 GM078000-01 from the National Institutes of Health.

The authors are thankful to Didier Dubois for valuable discussions, and to the anonymous referees for valuable suggestions.

References

1. Dubois, D., Prade, H.: Operations on fuzzy numbers, International Journal of Systems Science, 9, 613–626 (1978)
2. Ferson, S., Ginzburg, L., Kreinovich, V., Longpré, L., Aviles, M.: Computing variance for interval data is NP-hard, ACM SIGACT News, 33(2), 108–118 (2002)
3. Ferson, S., Ginzburg, L., Kreinovich, V., Longpré, L., Aviles, M.: Exact bounds on finite populations of interval data, Reliable Computing, 11(3), 207–233 (2005)
4. Ferson, S., Kreinovich, V., Hajagos, J., Oberkampf, W., Ginzburg, L.: Experimental Uncertainty Estimation and Statistics for Data Having Interval Uncertainty, Sandia National Laboratories, Report SAND2007-0939, May 2007 (2007)
5. Griewank, A., and Walter, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM Publ., Philadelphia, PA (2008)
6. Interval computations website http://www.cs.utep.edu/interval-comp
7. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics, Springer, London (2001)
8. Klir, G., Yuan, B.: Fuzzy Sets and Fuzzy Logic, Prentice Hall, Upper Saddle River, NJ (1995)
9. Kreinovich, V., Ferson, S.: A new Cauchy-based black-box technique for uncertainty in risk analysis, Reliability Engineering and Systems Safety, 85(1–3), 267–279 (2004)
10. Kreinovich, V., Longpré, L., Starks, S. A., Xiang, G., Beck, J., Kandathi, R., Nayak, A., Ferson, S., Hajagos, J.: Interval versions of statistical techniques, with applications to environmental analysis, bioinformatics, and privacy in statistical databases, Journal of Computational and Applied Mathematics, 199(2), 418–423 (2007)
11. V. Kreinovich, G. Xiang, S. A. Starks, L. Longpré, M. Ceberio, R. Araiza, J. Beck, R. Kandathi, A. Nayak, R. Torres, and J. Hajagos, Towards combining probabilistic and interval uncertainty in engineering calculations: algorithms for computing statistics under interval uncertainty, and their computational complexity, Reliable Computing, 12(6), 471–501 (2006)
12. Kreinovich, V., Xiang, G.: Fast algorithms for computing statistics under interval uncertainty: an overview, In: Huynh, V.-N., Nakamori, Y., Ono, H., Lawry, J., Kreinovich, V., Nguyen, H. T. (eds.), Interval/Probabilistic Uncertainty and Non-Classical Logics, Springer-Verlag, Berlin-Heidelberg-New York, 19–31 (2008)
13. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht (1997)
14. Li, M., Vitanyi, P.: *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, Berlin, Heidelberg, New York (2008)
15. Moore, R. E., Kearfott, R. B., Cloud, M. J.: Introduction to Interval Analysis, SIAM Press, Philadelphia, Pennsylvania (2009)

16. Nguyen, H. T., Kreinovich, V.: Nested intervals and sets: concepts, relations to fuzzy sets, and applications, In: Kearfott, R. B., Kreinovich, V., eds., Applications of Interval Computations, Kluwer, Dordrecht, 245–290 (1996)
17. Nguyen, H. T., Walker, E. A.: A First Course in Fuzzy Logic, Chapman & Hall/CRC, Boca Raton, Florida (2006)
18. Rabinovich, S. Measurement Errors and Uncertainties: Theory and Practice, Springer Verlag, New York (2005)
19. Trejo, R., Kreinovich, V.: Error estimations for indirect measurements: randomized vs. deterministic algorithms for 'black-box' programs, In: Rajasekaran, S., Pardalos, P., Reif, J., Rolim, J. (eds.), *Handbook on Randomized Computing*, Kluwer, 673–729 (2001)