Uncertainty Quantification for Results of AI-Based Data Processing: Towards More Feasible Algorithms

C. Q. Lauter, M. Ceberio, and V. Kreinovich*

Department of Computer Science, University of Texas at El Paso
El Paso, Texas 79968, USA
** E-mail: vladik@utep.edu, www.cs.utep.edu/vladik*

O. M. Kosheleva

Department of Teacher Education, University of Texas at El Paso
El Paso, Texas 79968, USA
E-mail: olgak@utep.edu

AI techniques have been actively and successfully used in data processing. This tendency started with fuzzy techniques, now neural network techniques are actively used. With each new technique comes the need for the corresponding uncertainty quantification (UQ). In principle, for both fuzzy and neural techniques, we can use the usual UQ methods – however, these techniques often require an unrealistic amount of computation time. In this paper, we show that in both cases, we can use specific features of the corresponding techniques to drastically speed up the corresponding computations.

1. Formulation of the problem

Need for inirect measurements. In many practical situations, we are interested in the value of a quantity y that is difficult (or even impossible) to measure directly. Since we cannot measure y directly, we measure it *indirectly*, i.e.:

- we find easier-to-measure quantities $x_1, \ldots, x_n$ that are related to y by a known dependence $y = f(x_1, \ldots, x_n)$,
- we measure these quantities, and
- we use the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ to estimate y as

$$\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n).$$

Computing $\widetilde{y}$ is an important case of what is called *data processing*.

Similar data processing is also used for prediction, to use the current values $x_1, \ldots, x_n$ of different physical quantities to compute estimates y for

future values of quantities of interest.

Need for uncertainty quantification (UQ). Measurements are never 100% accurate. Each measurement result $\widetilde{x}_i$ is, in general, different from the actual (unknown) value x_i; see, e.g.,[12]. As a result, even if the dependence $y = f(x_1, \ldots, x_n)$ is exact, the estimate $\widetilde{y}$ is, in general, different from the actual value y.

From the practical viewpoint, it is important to know how big can be the difference $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$. For example, if we are estimating the amount of water in an underground location as 100 units, then:

- if it is 100 ± 20, this is worth exploring, while
- if it is 100 ± 150 then maybe there is no water at all, so we better perform more measurements before starting the expensive drilling.

Typical types of uncertainty. To estimate Δy, we need to have information about measurement errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$. Usually (see, e.g.,[12]):

- we either know the probability distribution; this is known as *probabilistic uncertainty*;
- or we know the upper bound Δ_i on the absolute value $|\Delta x_i|$.

Let us describe the two corresponding UQ problems.

Comment about probabilistic uncertainty. In the probabilistic cases, usually, the mean values of the measurement error is 0. Indeed, if the mean is not 0, i.e., if the measuring instrument has bias, then we can re-calibrate all the measurement results by subtracting this known bias – and thus, get the mean to be 0.

Also, usually, the measurement error is caused by many relatively small effects. It is known that the joint effect of a large number of small errors leads to an approximately normal distribution; this is known as the Central Limit Theorem (see, e.g.,[13]). It is known that to uniquely describe a normal distribution with 0 mean, it is sufficient to describe its standard deviation σ_i. Thus, in the case of probabilistic uncertainty, we usually know the standard deviation σ_i of the corresponding measurement error.

Measurement errors of different measurements are usually caused by different factor and are, therefore, independent.

Comment about the second type of uncertainty. In the second case, once we get the measurement result $\widetilde{x}_i$, all we know about the actual value x_i of the corresponding quantity is that this value is in the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

The value Δ_i is then equal to the half-width of this interval. Because of this fact, this case is known as *interval uncertainty*; see, e.g., [3,6,7,9,12].

UQ: formulation of the problem in the case of probabilistic uncertainty. We know:

- the data processing algorithm $f(x_1, \ldots, x_n)$;
- the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$; and
- the standard deviations $\sigma_1, \ldots, \sigma_n$ of the corresponding measurement errors $\Delta x_i = \widetilde{x}_i - x_i$; the mean values of these errors are 0s, and different measurement errors are independent.

Based on this information, we need to compute the standard deviation σ of the resulting estimation error $\Delta y = \widetilde{y} - y$, where $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $y = f(x_1, \ldots, x_n)$.

UQ: formulation of the problem in the case of interval uncertainty. We know:

- the data processing algorithm $f(x_1, \ldots, x_n)$;
- the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$; and
- the upper bounds $\Delta_1, \ldots, \Delta_n$ of the absolute values of the corresponding measurement errors $\Delta x_i = \widetilde{x}_i - x_i$: $|\Delta x_i| \le \Delta_i$.

Based on this information, we need to compute the largest possible value Δ of the absolute value of the resulting estimation error $\Delta y = \widetilde{y} - y$, where $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $y = f(x_1, \ldots, x_n)$.

Possibility of linearization. Measurement errors Δx_i are usually reasonable small, in the sense that terms quadratic in terms of these errors are much smaller than the errors themselves – and can, thus be safely ignored. For example, if $\Delta x_i \approx 10\%$, then $(\Delta x_i)^2 \approx 1\%$, which is much smaller than 1%. In such situations, we can expand the dependence of Δy on Δx_i in Taylor series and keep only terms which are linear in Δx_i:

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

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

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

where $f_{,x_i}$ denotes the partial derivative of the function $f(x_1, \ldots, x_n)$ with respect to x_i:

$$f_{,x_i} \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}_{|x_1=\widetilde{x}_1,\ldots,x_n=\widetilde{x}_n}.$$

Thus, we have

$$\Delta y = \sum_{i=1}^{n} f_{,x_i} \cdot \Delta x_i. \tag{1}$$

Comment. In the following text, we will use the above simplifying notation for partial derivatives in other cases as well, i.e., we will denote

$$a_{,b} \stackrel{\text{def}}{=} \frac{\partial a}{\partial b}.$$

Linearization helps to solve both UQ problems. If we know all the partial derivatives $f_{,x_i}$, then we can use the known formulas for solving both UQ problems:

- in the probabilistic case, we have

$$\sigma^2 = \sum_{i=1}^{n} (f_{,x_i})^2 \cdot \sigma_i^2; \tag{2}$$

- in the interval case, we have

$$\Delta = \sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i. \tag{3}$$

How can we compute the partial derivatives? Sometimes, the data processing algorithm $f(x_1, \ldots, x_n)$ is straightforward, so partial derivatives can be computed by explicit differentiation. In most other cases, a natural idea is to use sensitivity analysis (i.e., in effect, numerical differentiation) to find the derivatives $f_{,x_i}$ of f with respect to the inputs x_i :

$$f_{,x_i} \approx \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}}{h_i}. \tag{4}$$

In this approach, we need several calls to the function $f(x_1, \ldots, x_n)$:

- we need one call to compute the value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, and

- we need n calls to compute n values

$$f\left(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n\right)$$

needed to compute n derivatives.

So overall, we need $n+1$ calls to the program that computes $f(x_1, \ldots, x_n)$.

Comment. When the number of inputs n is large, this idea – of computing all the derivatives and then applying formulas (2) and (3) – is not always practical. Indeed, in some such cases, computing f often takes hours on a high-performance computer. So we do not have the luxury of calling the function $f(x_1, \ldots, x_n)$ for each of thousands of variables. In such cases, we can use Monte-Carlo (MC) techniques:

- traditional MC techniques for probabilistic uncertainty (see, e.g., [13]), and
- Cauchy-based MC techniques for the case of interval uncertainty; see, e.g., [5].

AI-related challenges. AI techniques are becoming ubiquitous in data processing. Their results are good, but in terms of uncertainty quantification (UQ) they lead to new challenges.

First challenge: the use of fuzzy techniques. The first challenge is that, for some inputs x_i, we only have expert estimates. These estimates are often formulated by using imprecise ("fuzzy") words from natural language. There are special techniques – known as fuzzy techniques [1,4,8,10,11,14] – for translating these estimates into numerical machine-understandable form. In this techniques, an expert information about the quantity x_i is described by assigning, to each possible value of this quantity, a degree $\mu_i(x_i) \in [0,1]$ with which this value is possible. The corresponding function $\mu_i(x_i)$ is called a *membership function* or, alternatively, a *fuzzy set*.

This is largely equivalent to describing, for several degrees of confidence α, the interval $\mathbf{x}_i(\alpha)$ of all the values which are possible with at least this degree of confidence:

- for $\alpha > 0$, we have

$$\mathbf{x}_i(\alpha) = \{x_i : \mu_i(x_i) \geq \alpha\},$$

and

- for $\alpha = 0$, we have

$$\mathbf{x}_i = \overline{\{x_i : \mu_i(x_i) > 0\}},$$

where $\overline{S}$ means the closure of the set S.

In principle, there are UQ techniques for processing such fuzzy inputs.

In effect, we need to:

- select several certainty levels α, e.g., the values $\alpha = 0, 0.1, 0.2, \ldots, 0.9, 1$, and
- on each level, perform interval estimations, i.e., compute the range

$$\mathbf{y}(\alpha) = \{f(x_1, \ldots, x_n) : x_i \in \mathbf{x}_i(\alpha)\}. \tag{5}$$

The challenge is that this multiplies the UQ time – which is often already long – by a factor of 11. This often makes it not feasible.

Second challenge: the use of neural techniques. Second, data processing is now often performed by a deep neural network (NN), with billions of parameters; see, e.g.,[2]. In this case, each computation already requires a significant amount of computation time. So, for large number of inputs n, repeating the computations $n+1$ times – as in the traditional UQ techniques – is not possible.

What we do in this paper. In this paper, we provide possible solutions to both challenges.

2. What we propose: fuzzy case

Idea. For fuzzy techniques, we can use the fact that all input fuzzy sets usually have similar shapes: e.g., they are all triangular or they are all trapezoid.

Case of triangular membership functions. A (symmetric) triangular membership function $\mu_i(x_i)$ is different from 0 on some interval $[\underline{x}_i, \overline{x}_i] = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$, in which its value is equal:

- to $\mu_i(x_i) = (\overline{x}_i - x_i)/\Delta_i$ for $x_i \in [\widetilde{x}_i, \overline{x}_i]$,
- to $\mu_i(x_i) = (x_i - \underline{x}_i)/\Delta_i$ for $x_i \in [\underline{x}_i, \widetilde{x}_i]$, and
- to 0 when $x_i \notin [\underline{x}_i, \overline{x}_i]$.

For this membership function, the half-width $\Delta_i(\alpha)$ of the α-cut interval is equal to $(1 - \alpha) \cdot \Delta_i$. By plugging in this expression into the formula (3),

we conclude that

$$\Delta(\alpha) = \sum_{i=1}^{n} |f_{,x_i}| \cdot (1-\alpha) \cdot \Delta_i = (1-\alpha) \cdot \sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i.$$

So, in this case, there is no need to apply the formula (3) 11 times:

- we can apply the formula (3) once, for $\alpha = 0$, when $\Delta_i(0) = \Delta_i$, and
- then, to find the values $\Delta(\alpha)$ corresponding to all other α, we can simply multiply $\Delta(0)$ by $1 - \alpha$.

Case of trapezoid membership functions. Similarly, a (symmetric) trapezoid membership function $\mu_i(x_i)$ is different from 0 on some interval $[\underline{x}_i, \overline{x}_i] = [\widetilde{x}_i - \Delta_i^{(1)} - \Delta^{(2)}, \widetilde{x}_i + \Delta_i^{(1)} + \Delta_i^{(2)}]$, in which its value is equal:

- to $\mu_i(x_i) = 1$ when $x_i \in [\widetilde{x}_i - \Delta_i^{(1)}, \widetilde{x}_i + \Delta_i^{(1)}]$,
- to $\mu_i(x_i) = (\overline{x}_i - x_i)/\Delta_i^{(2)}$ for $x_i \in [\widetilde{x}_i + \Delta_i^{(1)}, \overline{x}_i]$,
- to $\mu_i(x_i) = (x_i - \underline{x}_i)/\Delta_i^{(2)}$ for $x_i \in [\underline{x}_i, \widetilde{x}_i - \Delta_i^{(1)}]$, and
- to 0 when $x_i \notin [\underline{x}_i, \overline{x}_i]$.

For this membership function, the half-width $\Delta_i(\alpha)$ of the α-cut interval is equal to $\Delta_i^{(1)} + (1-\alpha) \cdot \Delta_i^{(2)}$. By plugging in this expression into the formula (3), we conclude that

$$\Delta(\alpha) = \sum_{i=1}^{n} |f_{,x_i}| \cdot [\Delta_i^{(1)} + (1-\alpha) \cdot \Delta_i^{(2)}] = \sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i^{(1)} + (1-\alpha) \cdot \sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i^{(2)}.$$

So, in this case, there is no need to apply the formula (3) 11 times:

- we can apply the formula (3) twice, first for the values $\Delta_i^{(1)}$, and then for the values $\Delta_i^{(2)}$, resulting in the values $\Delta^{(1)}$ and $\Delta^{(2)}$;
- then, to find the values $\Delta(\alpha)$ corresponding to all other α, we can simply compute $\Delta^{(1)} + (1-\alpha) \cdot \Delta^{(2)}$.

More general case. We can have other cases in which the half-width $\Delta(\alpha)$ of the α-cut interval has the form

$$\Delta_i(\alpha) = \sum_{j=1}^{k} f_j(\alpha) \cdot \Delta_i^{(j)}, \tag{6}$$

for some $k \ll 11$. Here, the functions $f_j(\alpha)$ are common for all membership functions of this type, while the values $\Delta_i^{(j)}$ depend on the specific membership function from this family. For example:

- for triangular membership functions, we have $k = 1$ and $f_1(\alpha) = 1 - \alpha$;
- for trapezoid membership functions, we have $k = 2$, $f_1(\alpha) = 1$, and $f_2(\alpha) = 1 - \alpha$.

In general, substituting the expression (6) into the formula (3), we get

$$\Delta(\alpha) = \sum_{j=1}^{k} f_j(\alpha) \cdot \left(\sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i^{(j)} \right).$$

Here too, there is no need to apply the formula (3) 11 times:

- we can apply the formula (3) k times, each time $j = 1, \ldots, k$ for the values $\Delta_i^{(j)}$, resulting in the values $\Delta^{(1)}, \ldots, \Delta^{(k)}$;
- then, to find the values $\Delta(\alpha)$ corresponding to all other α, we can simply multiply compute

$$\Delta(\alpha) = \sum_{j=1}^{k} f_j(\alpha) \cdot \Delta^{(j)}.$$

Thus, we arrive at the following algorithm.

Formulation of the problem. We know:

- the data processing algorithm $f(x_1, \ldots, x_n)$;
- the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$; and
- for different values $\alpha \in [0, 1]$, the half-widths $\Delta_1(\alpha), \ldots, \Delta_n(\alpha)$ of the corresponding *alpha*-cut intervals, for which, for some small value k and for some known functions $f_1(\alpha), \ldots, f_k(\alpha)$, we have

$$\Delta_i(\alpha) = \sum_{j=1}^{k} f_j(\alpha) \cdot \Delta_i^{(j)},$$

for some values $\Delta_i^{(j)}$.

Based on this information, we need to compute the half-width $\Delta(\alpha)$ of the α-cut intervals for y.

Resulting algorithm.

- First, for each j from 1 to k, we compute

$$\Delta^{(j)} = \sum_{i=1}^{n} |f_{,x_i}| \cdot \Delta_i^{(j)}.$$

- Then, for each α, we compute

$$\Delta(\alpha) = \sum_{j=1}^{k} f_j(\alpha) \cdot \Delta^{(j)}.$$

Discussion. By using this algorithm, instead of 11 applications of the time-consuming formula (3), we only apply it $k \ll 11$ times:

- $k = 1$ time for triangular membership functions,
- $k = 2$ times for trapezoid membership functions, etc.

Thus, we get a significant 5-10 times speedup.

3. What we propose: neural case

Idea. For neural data processing, we can use the fact that:

- when a neural network is trained,
- it already computes a lot of partial derivatives.

Our idea us to use these partial derivatives to estimate the desired derivatives $f_{,x_i} = y_{,x_i}$.

How neural networks are trained: reminder. To describe how to do it, let us recall how neural networks are trained.

- In the first layer, the inputs x_i are transformed into

$$y_j = s\left(\sum_{i=1}^{n} w_{ji} \cdot x_i + w_{j0}\right),$$

  for some nonlinear function $s(z)$ that is known as the *activation function*.
- Outputs of each layer serve as inputs for the next layer, etc.

At the end of the forward step, we get the final result y.

Then, we form a measure of accuracy $J(y)$ – e.g., $J(y) = (y - \widetilde{y})^2$ – that we will minimize. After that, we use a special backpropagation algorithm to compute the partial derivatives $J_{,w}$ of the objective function J with respect to each weight w.

As intermediate results in these computations:

- we compute the values $s'\left(\sum_{i=1}^{n} w_{ji} \cdot x_i + w_{j0}\right)$, where $s'(z)$, as usual, means the derivative of the function $s(z)$; we will denote these values by y'_j; and

- we compute the derivatives $J_{,y_j}$ of the objective function J with respect to all the intermediate results y_j.

Then, we use gradient descent, i.e., we update each weight w by using the gradient descent formula:

$$w \mapsto w - \beta \cdot J_{,w}.$$

Analysis of the problem. As part of usual training-related these computations, we compute the values $J_{,y_j}$, where y_j are outputs of the first layer. We know that $J_{,y_j} = J_{,y} \cdot f_{,y_j}$, so we can compute $y_{,y_j} = J_{,y_j}/J_{,y}$. Here, $f_{,x_i} = \sum\limits_j y_{,y_j} \cdot y_{j,x_i}$, where $y_{j,x_j} = y'_j \cdot w_{ji}$. Thus, $f_{,x_i} = \sum\limits_j (J_{,y_j}/J_{,y}) \cdot y'_j \cdot w_{ji}$. Here:

- We know the weights w_{ji}.
- We can easily compute $J_{,y}$.
- From the backpropagation step, we know the values $J_{,y_j}$ and y'_j.

Thus, we can indeed easily compute all the derivatives $f_{,x_i}$, without any extra calls to f.

Once we have all the derivatives $f_{,x_i}$, we can use the formulas (2) and (3) to propagate uncertainty. So, we arrive at the following algorithm.

Formulation of the problem. We know:

- the function $f(x_1, \ldots, x_n)$ computed by a trained neural network; and
- the measurement results $\widetilde{x}_1, \ldots, \widetilde{x}_n$.

We want to compute the partial derivatives $f_{,x_i}$ of the function $f(x_1, \ldots, x_n)$ with respect to all its inputs $x_1, \ldots, x_)n$. Once we know these derivatives, we can use the formulas (2) or (3) to propagate uncertainty.

Resulting algorithm. From the neural network, we know all the weights w_{ji}. During the training, we have computed:

- the values $J_{,y_j}$ – the partial derivatives of the objective function with respect to the outputs y_j of the first data processing layer, and
- the values $y'_j = s'\left(\sum\limits_{i=1}^{n} w_{ji} \cdot x_i + w_{j0}\right)$ corresponding to this layer.

We know the objective function $J(y)$, so we can easily compute the derivative $J_{,y}$. Then, we can compute the desired partial derivatives by using the following formula:

$$f_{,x_i} = \sum_j \frac{J_{,y_j}}{J_{,y}} \cdot y'_j \cdot w_{ji}.$$

Comment. This algorithm implicitly assumes that the neural continues to learn as new data come in.

It is, of course, not applicable to the cases when the network is trained only once, after which its weights are frozen and it no longer learns. For such neural networks, to estimate the derivatives, we may want:

- to find, among all the inputs $x = (x_1, \ldots, x_n)$ used for training, an input which is the closest to the given values $\widetilde{x} = (\widetilde{x}_1, \ldots, \widetilde{x}_n)$, and
- to use the derivatives $f_{,x_i}$ corresponding to the closest input x in the given case as well.

Acknowledgments

This work was supported in part by the National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science), HRD-1834620 and HRD-2034030 (CAHSI Includes), EAR-2225395 (Center for Collective Impact in Earthquake Science C-CIES), and by the AT&T Fellowship in Information Technology.

It was also supported by the program of the development of the Scientific-Educational Mathematical Center of Volga Federal District No. 075-02-2020-1478, and by a grant from the Hungarian National Research, Development and Innovation Office (NRDI).

The authors are greatly thankful to Franco Pavese for his encouragement and to all the participants of the International Conference on Advanced Mathematical Tools in Metrology and Testing AMCTM 2023 (September 26–28, 2023) for valuable discussions.

References

1. R. Belohlavek, J. W. Dauben, and G. J. Klir, *Fuzzy Logic and Mathematics: A Historical Perspective*, Oxford University Press, New York, 2017.

2. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, Massachusetts, 2016.
3. L. Jaulin, M. Kiefer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control, and Robotics*, Springer, London, 2012.
4. G. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic*, Prentice Hall, Upper Saddle River, New Jersey, 1995.
5. V. Kreinovich and S. Ferson, "A new Cauchy-based black-box technique for uncertainty in risk analysis", *Reliability Engineering and Systems Safety*, 2004, Vol. 85, No. 1–3, pp. 267–279.
6. B. J. Kubica, *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization, and Similar Problems: from Inequalities Systems to Game Solutions*, Springer, Cham, Switzerland, 2019.
7. G. Mayer, *Interval Analysis and Automatic Result Verification*, de Gruyter, Berlin, 2017.
8. J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, Springer, Cham, Switzerland, 2017.
9. R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM, Philadelphia, 2009.
10. H. T. Nguyen, C. L. Walker, and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2019.
11. V. Novák, I. Perfilieva, and J. Močkoř, *Mathematical Principles of Fuzzy Logic*, Kluwer, Boston, Dordrecht, 1999.
12. S. G. Rabinovich, *Measurement Errors and Uncertainty: Theory and Practice*, Springer Verlag, New York, 2005.
13. D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.
14. L. A. Zadeh, "Fuzzy sets", *Information and Control*, 1965, Vol. 8, pp. 338–353.