# ON HARDWARE SUPPORT FOR INTERVAL COMPUTATIONS AND FOR SOFT COMPUTING: THEOREMS

Hung T. Nguyen, Vladik Kreinovich, *Member, IEEE*, Vyacheslav Nesterov, and Mutsumi Nakamura

**Abstract.** *This paper provides a rationale for providing hardware supported functions of more than two variables for processing incomplete knowledge and fuzzy knowledge. The result is in contrast to Kolmogorov's theorem in numerical (non-fuzzy) case.*

## 1. INTRODUCTION

In this paper, we show that for interval computations and for processing fuzzy data (i.e., for soft computing), it is desirable to have hardware supported operations with more than two operands.

Before we formulate the problem and go into technical details, we would like to emphasize the importance of this problem by briefly describing in Subsection 1.1 the practical origin (and practical necessity) of interval computations and soft computing.

### 1.1. Estimating accuracy of the results of data processing: crisp and fuzzy cases

**Data processing: why?** To make decisions, we must have some information about the values of the physical quantities. For example, in order to decide whether to approve the scheduled launch of a Space Shuttle, we must know the characteristics of the Shuttle (to make sure that all its systems work), and weather conditions around the launch site during the launch time. Some of these characteristics can be measured directly: e.g., characteristics of the Shuttle's electric systems can be measured by testers. Some of the desired characteristics can be estimated by experts: e.g., some experts meteorologists can provide us with reasonably good short-time weather predictions for a given area.

---

Hung T. Nguyen is with the Department of Mathematical Sciences, New Mexico State University, Las Cruces, NM 88003, USA, email `hunguyen@nmsu.edu`

Vladik Kreinovich is with the Department of Computer Science Department, University of Texas at El Paso, El Paso, TX 79968, USA, email `vladik@cs.utep.edu`

Vyacheslav Nesterov is with the Institute of New Technologies, P. O. Box 52, St. Petersburg 256, 195256 Russia, email `nest@into.nit.spb.su`

Mutsumi Nakamura is with the Department of Mathematics, University of Texas at Austin, Austin, TX 78712, USA, email `mutsumin@math.utexas.edu`

In some cases, however, it is very difficult (or even impossible) to measure the characteristic $y$ that we are interested in, and there are no experts who can predict the values of these characteristics. For example, it is very difficult to directly measure the temperature inside the jet chamber (because this temperature is extremely high); if we are planning a mission to a new planet, it is simply impossible to directly measure the characteristics of the new environment before the mission actually gets there, and often, no expert can help.

If we are interested in the value of such a characteristic $y$, and we cannot estimate $y$ *directly* (either by measurement, or by using experts), then a natural idea is to estimate $y$ *indirectly*, i.e.:

- to estimate some other (easier to estimate) quantities $x_1, ..., x_n$ that are related to $y$, and then
- to compute the estimate $\tilde{y}$ for $y$ based on the estimates $\tilde{x}_1, .., \tilde{x}_n$ for $x_1, ..., x_n$.

This process is called *data processing,* and this is what super-computers are doing most of the time: from measured characteristics $\tilde{x}_i$ of observed collisions in the accelerators, they reconstruct the (directly unobservable) properties of the elementary particles; from the results $\tilde{x}_i$ of geophysical measurements, computers predict the amount $y$ of oil (or other mineral) in a given area, etc.

In this paper, we will assume that we already know what characteristics $x_i$ to measure, and how to reconstruct $y$ from $x_i$. In other words, we will assume that we know an *algorithm* $f(x_1, ..., x_n)$ that transforms the values of $x_i$ into an estimate for $y$. This algorithm is not necessarily simple: e.g., in geophysics, it may involve solving a complicated non-linear integral equation ("inverse problem"); in elementary particle physics, it may involve solving a system of non-linear operator quantum equations, etc.

**The result of data processing is never absolutely accurate.** The assumption that we know the algorithm $f$ means that if we know the *exact* values of the variables $x_1, ..., x_n$, we can then apply the algorithm $f$ and compute the exact value of $y$. In reality, however, we only know some estimates $\tilde{x}_i$ for $x_i$ that are obtained either by measurements or by an expert estimation. Measurements are never 100% precise; expert estimates are not absolutely precise either. As a result, the available values $\tilde{x}_i$ differ from the actual (unknown) values $x_i$; therefore, the estimate $\tilde{y} = f(\tilde{x}_1, ..., \tilde{x}_n)$ that we obtain by processing the available data may differ from the desired value $y = f(x_1, ..., x_n)$.

**In real-life applications, we must know the accuracy of the result of data processing.** For practical purposes, it is important to know how different the result $\tilde{y}$ of data processing can be from the actual value $y$.

For example, if we want to decide whether a particular well is worth drilling, and the estimate for the amount of oil is $\tilde{y} = 100$ mln. tons, then before we start drilling, we would like to know whether this is, say, $100 \pm 1$, in which case, we should probably start drilling, or it is $100 \pm 100$ (maybe 100, maybe 0, maybe 200), in which case we would rather undertake further (and more accurate) measurements.

In this paper, we consider the problem of finding this accuracy.

**Simplest case: measurements only.** Before we start analyzing the general case, where both measurement results and expert estimates are present, let us consider the simplest case, where there is no expert knowledge, and all the data come from the measurements.

In traditional measurement theory (see, e.g., [8,47,35]), it is usually assumed that we know the probabilities of different values of a measurement error. These values can be obtained if we *calibrate* the measuring instrument, i.e.:

- we use the calibrated instrument in conjunction with a much more accurate one (called a *standard*) in several measurements;
- for each measurement, we compute a difference $e^{(k)} = \tilde{x}^{(k)} - x^{(k)}$ between the results of these two instruments, and use this difference as an estimate of the error of the measurement performed by the calibrated instrument;
- reconstruct the error probability distribution from the recorded sample errors $e^{(1)}$, ..., $e^{(N)}$.

For the situations in which we know the probabilities of different errors, there exist numerous methods that compute statistical characteristics of the resulting error.

In many real-life situations, however, the values of the probabilities are not known:

- in advanced measurements (in radio-astronomy, in elementary particle physics, etc), we are using measuring instruments that have the highest accuracy possible, so, there is simply no "more accurate" measuring instrument that we can for calibrating;
- in manufacturing applications, we can potentially calibrate all the sensors that we use, but this calibration would cost much more than the sensors themselves, so it is usually not done.

In these situations, the manufacturer of the measuring instrument provides us with the guaranteed accuracy $\Delta$, i.e., with a guaranteed upper bound of the error $\Delta x = \tilde{x} - x$ (e.g., "error cannot exceed 0.1"). If our measurement results is $\tilde{x}$, then the possible values of $x = \tilde{x} - \Delta x$ form an *interval* $[\tilde{x} - \Delta, \tilde{x} + \Delta]$. Since we are dealing with intervals, the entire area is called *interval computations* (see, e.g., [29,11,10,18,14]).

The set of possible values of an error is not necessarily an interval. For example, suppose that we are measuring the current inside the computer, and we know that the error cannot exceed a certain value $\Delta$. If we know nothing else about the error, then we may conclude that the error belongs to the interval $[-\Delta, \Delta]$. However, we may know that the error is caused by the influence of a nearby magnetic memory element, which can be in two possible states (corresponding to "0" and "1"). In this case, the error is either positive, or negative (depending on the state), but never 0; actually, the error can never be smaller than some value $\delta$. In this case, the set $X$ of possible values of the error is not an interval, but a union of two intervals: $X = [-\Delta, -\delta] \cup [\delta, \Delta]$. There can be more complicated cases, in which the error can be described by more complicated (crisp) sets $X$ of possible values.

In this case, our problem takes the following form:

*We know:*

- *an algorithm $f$ that transform $n$ real numbers $x_1, ..., x_n$ into a real number $y = f(x_1, ..., x_n)$;*
- *sets $X_1 \subseteq R$, ..., $X_n \subseteq R$ that contain the actual values of $x_i$;*

*We must compute: The set $Y$ of possible values of $y$:*

$$Y = \{f(x_1, ..., x_n) \,|\, x_1 \in X_1, ..., x_n \in X_n\} \tag{1.1}$$

This set $Y$ is usually denoted by $f(X_1, ..., X_n)$.

A measuring instrument can measure several different quantities $x_1, ..., x_n$ at a time. In this case, in addition to the information about the possible errors of each measurement, the manufacturer can guarantee that certain combinations of errors are impossible: e.g., it can happen that $\Delta x_1$ attains its largest possible value $\Delta$, and it can happen that $\Delta x_2$ attains its largest possible value $\Delta$, but they can never attain these extreme values at the same time, because of the restriction that $\Delta x_1^2 + \Delta x_2^2 \leq \Delta^2$ (this situation happens, e.g., if we measure geographical coordinates of a point). Such an information can be described by a set $X \subseteq R^n$ of all the tuples that the manufacturer believes can be possible values of errors $(\Delta x_1, ..., \Delta x_n)$. In this case, the problem takes the following form:

*We know:*

- *an algorithm $f$ that transform $n$ real numbers $x_1, ..., x_n$ into a real number $y = f(x_1, ..., x_n)$;*
- *a set $X \subseteq R^n$ that contains the actual value of $\vec{x} = (x_1, .., x_n)$;*

*We must compute: The set $Y$ of possible values of $y$:*

$$Y = \{f(x_1, ..., x_n) \,|\, (x_1, ..., x_n) \in X\} \tag{1.2}$$

This set $Y$ is usually denoted by $f(X)$.

The previous formulation is a particular case of this one if we take $X = X_1 \times ... \times X_n$.

**General case: processing data that includes expert knowledge as well.** In many cases, in addition to measurements results, we have expert's knowledge about the variables $x_1, ..., x_n$. For example, in order to make a decision on what doze of radiation to assign to a patient, we must know not only the characteristics that are measurable (like blood count, tumor size, etc), but the characteristics that can only be estimated by an expert (e.g., the granularity of a tumor can be "small" or "medium"). We want to process this informal information automatically, therefore, we must be able to represent it in the computer. A word like "medium-size" does not describe one particular value; it can correspond to several different values; some of them are more reasonably described as "medium-size", some values can be in principle described as such, but only occasionally. To describe the meaning of each word, we ascribe to every real number $x$ a value $\mu(x) \in [0, 1]$ that describe to what extent it is reasonable to assume that $x$ is, say, medium-size (1 means that it is absolutely reasonable, 0 that it is not reasonable at all). The resulting function is called a *membership function*, or a *fuzzy set*.

In some cases, the expert's informal statement describes not one variable, but several of them. For example, if we say that a point with coordinates $x_1, x_2$ is close to 0, this means that both $x_1$ and $x_2$ are close to 0. Such knowledge can be represented by a function from $R^2$ to [0,1]. This membership function is called a *fuzzy subset* of $R^2$. If we have such information about $x_i$, and we want to estimate $y$, we get the following problem:

*We know:*

- *a function $f$ of $n$ variables;*
- *a fuzzy set $X \subseteq R^n$ that describes our knowledge about $\vec{x} = (x_1, ..., x_n)$.*

*We want: to describe the resulting knowledge about $y$ in terms of a fuzzy set $Y$.*

This problem was formulated, e.g., in [1,46,39], and it has appeared in many practical cases, including:

- *testing jet engines* [22,20,21];
- *seismic analysis* [3,4];
- *image processing* [19,20,21],

  etc.

The desired description of the set $Y$ is known as *extension principle*. This principle was proposed by Zadeh in his pioneer paper [53] (see also [54] and [5]), and it is based on the following idea:

*A real number $\tilde{y}$ is a reasonable value of $y$ if and only if*
*there exist values $\tilde{x}_1, ...\tilde{x}_n$ for which*
*$\tilde{x}_1$ is a possible values of $x_1$, $\tilde{x}_2$ is a possible value of $x_2$, ..., and $f(\tilde{x}_1, ..., \tilde{x}_n) = \tilde{y}$.*

If we follow the traditional fuzzy set theory and interpret "and" as min, and "there exists" as sup, then we arrive at the following formula:

$$\mu_Y(y) = \sup_{\vec{x} \in R^n} (\min(\mu_X(\vec{x}), \chi_f(\vec{x}, y)),$$

where $\chi_f$ is a characteristic function of the graph of the function $f$ (i.e., $\chi(\vec{x}, y) = 1$ if $f(\vec{x}) = y$, and 0 otherwise). Due to this formula, values $\vec{x}$ for which $f(\vec{x}) \neq y$, do not influence on $\mu_Y(y)$. Therefore, this formula can be rewritten as follows:

$$\mu_Y(y) = \sup_{\vec{x}: f(\vec{x}) = y} \mu_X(\vec{x}). \tag{1.3}$$

This formula is called the *extension principle*, and the resulting fuzzy set $Y$ is denoted by $f(X)$.

If instead of $X$, we have $n$ separate fuzzy sets $X_1, ..., X_n$ that describe our knowledge about $x_1, ..., x_n$, then similar arguments lead to a formula

$$\mu_Y(y) = \sup_{\vec{x}: f(\vec{x}) = y} \min(\mu_{X_1}(x_1), ..., \mu_{X_n}(x_n)). \tag{1.4}$$

The resulting fuzzy set $Y$ is denoted by $f(X_1, ..., X_n)$.

*Comments.*

1. In particular, if we take elementary arithmetic operations ($+, \times, -$, etc) as $f$, we get the definition of arithmetic operations with fuzzy operands $X_1, ..., X_n$.

2. The statement that we have just formalized contains two logical terms: "and" and "there exists". So, to formalize it, we must formalize what these two logical terms mean.

"There exists" can be viewed as an infinite "or": namely, "there exist $\tilde{x}_1, ..., \tilde{x}_n$ with a certain property" means that this property is either true for, say, $(0.0, ..., 0.0)$, or for $(1.1, 0.2, ..., 2.3)$, or for any of infinitely many tuples of $n$ real numbers. Therefore, to get an interpretation of "there exists", we must choose an appropriate fuzzy interpretation $f_\vee(a,b)$ of $\vee$ and apply the resulting $\vee$−operation $f_\vee$ infinitely many times (i.e., apply it to $N$ tuples and then take $N \to \infty$).

In fuzzy logic, many different $\vee$−operations have been proposed (e.g., $f_\vee(a,b) = a+b-a\cdot b$); these operations are also called $t$−conorms. A usual (and natural) restriction on a $t$−conorm $f_\vee$ comes from the fact that for every two statement $A$ and $B$, our degree of belief in $A \vee B$ must be at least as big as the degree of belief in each of these statements $A$ and $B$. If we denote the degree of belief in $A$ by $t(A)$, then this condition turns into $f_\vee(t(A), t(B)) \geq t(A)$ and $f_\vee(t(A), t(B)) \geq t(B)$. In other words, for every $a$ and $b$, $f_\vee(a,b) \geq \max(a,b)$.

If we choose one of the known $t$−conorms for which $f_\vee(a,b) > \max(a,b)$ (e.g., if we choose $f_\vee(a,b) = a+b-ab$), then, the more times we apply $f_\vee$, the larger the resulting degree of belief, and in the limit, we get 1. For example, for $f_\vee(a,b) = a+b-ab = 1-(1-a)\cdot(1-b)$, disjunction of $N$ formulas with the same degree of belief $a \in (0,1)$ leads to $f_\vee(a,\ldots,a) = 1-(1-a)^N$, and this expression $\to 1$ as $N \to \infty$. (Here, $f_\vee(a,b,...,c)$ stands for $f_\vee(...(f_\vee(a,b),...),c)$, i.e., it means that we apply the $\vee$−operation $N$ times.) A similar result can be proven for any (strict or non-strict) Archimedean $t$−conorm (see, e.g., [16,34]). Hence, for such operations, we will have $\mu_Y(y) = 1$ for all $y$, which makes no sense.

Therefore, when we define operations with fuzzy operands, the only meaningful interpretation of "there exists" is through the $\vee$−operation $f_\vee(a,b) = \max(a,b)$, for which for every property $A(x)$, the degree of belief in "there exists $x$ such that $A(x)$" is equal to the "maximum" (or, to use the precise mathematical term, supremum) of all the degrees of belief $t(A(x))$ for all $x$.

As far as an $\&$−operation is concerned, we can use an arbitrary function $f_\& : [0,1] \times [0,1] \to [0,1]$ that extends a usual & operation defined for binary values (with 0 as false as 1 as true), i.e., an arbitrary function $f_\&$ for which $f_\&(0,0) = f_\&(0,1) = f_\&(1,0) = 0$ and $f_\&(1,1) = 1$.

For such interpretation of "and" and "there exists", formula (2) takes the following form:

$$\mu_Y(y) = \sup_{\vec{x}: f(\vec{x}) = y} f_\&(\mu_{X_1}(x_1), ..., \mu_{X_n}(x_n)), \tag{1.4a}$$

where $f_{\&}(a, b, ..., c)$ stands for $f_{\&}(...(f_{\&}(a, b), ...), c)$ (i.e., it means that we apply the $\&-$operation several times).

Our results will be true for an arbitrary choice of an $\&-$operation.

### 1.2. How is the problem of estimating accuracy of the results of data processing solved now?

**In general, the problem is computationally intractable even for crisp sets (even for intervals).** It has been proven that even for the case when the sets $X_i$ are crisp (and are intervals), and the algorithm $f$ is actually a polynomial, the problem of computing the set $Y$ *exactly* is computationally intractable (NP-hard) [9].

This result does not mean, of course, that the problem of computing $Y$ is not practically solvable. The sets $X_i$ describe the inaccuracy of the measuring devices, or the inaccuracy of an expert. These inaccuracies are never known precisely, therefore, it would be quite sufficient to have an *approximate* description of $Y$. Several methods are known for that:

**Case when estimates are pretty accurate: linearization technique.** If the estimates $\tilde{x}_i$ for $x_i$ are pretty accurate, then we can neglect the terms that are quadratic (or of higher order) in $\Delta x_i = \tilde{x}_i - x_i$, and thus, for given estimates $\tilde{x}_i$, describe $y = f(x_1, ..., x_n) = f(\tilde{x}_1 - \Delta x_1, ..., \tilde{x}_n - \Delta x_n)$ by the following approximate formula: $y \approx f_{\rm lin} = \tilde{y} - f_{,1}\Delta x_1 - ... - f_{,n}\Delta x_n$, where by $f_{,i}$, we denoted the partial derivative of $f$ w.r.t. $x_i$:

$$f_{,i} = \frac{\partial f}{\partial x_i}(\tilde{x}_1, ..., \tilde{x}_n).$$

In this case, for interval $X_i = [\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$, we have $Y \approx X_{\rm lin} = f_{\rm lin}(X_1, ..., X_n) = [\tilde{y} - \Delta, \tilde{y} + \Delta]$, where $\Delta = |f_{,1}|\Delta_1 + ... + |f_{,n}|\Delta_n$ (see, e.g., [35]).

Another case when we can estimate $Y$ is when the function $f$ is monotonic in each of the variables; in this case, if, e.g., $f$ is monotonically increasing, and $X_i = [x_i^-, x_i^+]$, we have $Y = [f(x_1^-, ..., x_n^-), f(x_1^+, ..., x_n^+)]$.

For these two cases (small errors or monotonic $f$), there also exist efficient techniques for *fuzzy* data processing [4,51].

**Expert estimates are rarely very accurate, so, other methods are needed.** Measurements typically lead to accurate estimates of $x_i$, so, if all the estimates come from

measurements, we can usually apply linearization techniques. Expert estimates, on the other hand, are rarely very accurate. So, if we have expert estimates, we usually cannot neglect the squares of the errors, and therefore, we need other methods for estimating the error of the result of data processing. For this case, the following idea has been proposed by R. Moore [27,28] (see also [29,11,10,14]). No matter what high-level language we use to describe an algorithm $f$, inside a computer, this algorithm is translated into a sequence of elementary operations (usually, $+$, $-$, $\cdot$, $:$, etc).

For example, a function $f(x_1, x_2, x_3) = (x_1 + x_2)^2 + 2 \cdot x_3$ is computed as follows (we will enumerate all the input and intermediate values by $r_1, r_2$, ...): first, we have $r_1 = x_1$, $r_2 = x_2$, and $r_3 = x_3$; then, we start computing further values:

- we apply $+$ and get $r_4 = r_1 + r_2 = x_1 + x_2$;
- we apply the square operation and get $r_5 = r_4^2$ (so, $r_5 = (x_1 + x_2)^2$).
- we take $r_6 = 2$;
- we compute $r_7 = r_6 \cdot r_3$;
- finally, we compute $r_8 = r_5 + r_7$; this is the desired value $y$.

The idea is to *repeat* the same sequence of operations, but *with intervals* instead of numbers. Elementary operations $g$ are usually monotonic, so, we can explicitly compute $g(X_1, ..., X_n)$ for intervals $X_i$: e.g., for addition $g(x_1, x_2) = x_1 + x_2$, we have $g([x_1^-, x_1^+], [x_2^-, x_2^+]) = [x_1^- + x_2^-, x_1^+ + x_2^+]$.

For example, if we start with the intervals $R_1 = X_1 = [0, 1]$, $R_2 = X_2 = [0, 1]$, and $R_3 = X_3 = [1, 2]$, we get the following sequence of computations:

- we apply $+$ and get $R_4 = R_1 + R_2 = [0, 1] + [0, 1] = [0, 2]$;
- we apply the square operation and get $R_5 = R_4^2 = [0, 4]$;
- take $R_6 = [2, 2]$;
- compute $R_7 = R_6 \cdot R_3 = [2, 2] \cdot [1, 2] = [2, 4]$;
- finally, we compute $\tilde{Y} = R_8 = R_5 + R_7 = [0, 4] + [2, 4] = [2, 8]$; this is the desired estimate for $Y$.

It has been proven that for intervals $X_i$, the resulting estimate $\tilde{Y}$ *contains* the desired interval $Y$.

A similar procedure can be used for fuzzy processing: here, we implement elementary operations $g$ using extension principle.

**These new methods do not always lead to accurate results.** The results of these computations do not always lead to the exact value of $Y$; even for intervals $X_i$, the result

depends on the exact order of the operations performed to compute $f$. For example, we can compute $f(x_1, x_2) = x_1 \cdot x_2$ by simply multiplying the two numbers, or we can compute the same product by using a more complicated formula $x_1 \cdot x_2 = (1/4) \cdot [(x_1 + x_2)^2 - (x_1 - x_2)^2]$. If $X_1 = X_2 = [1, 2]$, then the first algorithm leads to the estimate $\tilde{Y} = [1, 4]$ that coincides with the desired interval $Y = f(X_1, X_2)$. However, the second algorithm leads to a different result: indeed, this algorithm can be represented as a following sequence of computations:

- $r_3 := r_1 + r_2$.
- $r_4 := r_3^2$.
- $r_5 := r_1 - r_2$.
- $r_6 := r_5^2$.
- $r_7 := r_4 - r_6$.
- $r_8 := 4$.
- $r_9 := r_7 / r_8$.

So, for $X_i = [1, 2]$, we get $R_3 = [2, 4]$, $R_4 = [4, 16]$, $R_5 = [-1, 1]$, $R_6 = [0, 1]$, $R_7 = [3, 16]$, $R_7 = [4, 4]$, and $\tilde{Y} = R_9 = [0.75, 4] \neq Y = [1, 4]$. The resulting interval $\tilde{Y}$ contains extra points $[0.75, 1)$.

### 1.3. Going from numbers to intervals and fuzzy sets drastically increases computation time, so hardware support is in order

**Hardware support is necessary.** We have already mentioned that even for real numbers, data processing algorithm that computes $f(x_1, ..., x_n)$ can be quite complicated and time-consuming. When we analyze *accuracy* of data processing, we must go from operations with numbers to operations with intervals, crisp sets, or fuzzy sets. For example, when we go from precise numbers to fuzzy sets, then instead of processing $n$ numbers according to the known algorithm $f$, we have to solve complicated optimization problems to find $\mu_Y(y)$. This increases computation time drastically: for example, in a crisp case, to compute a sum of two numbers $x_1 + x_2$, we must process these two numbers $x_i$ only; all it takes is one arithmetic operation. To compute a sum of two *fuzzy operands*, instead of processing two numbers, we must take as input the values $\mu_{X_1}(x_1)$ and $\mu_{X_2}(x_2)$ that correspond to different $x_i$. Just because of the necessity to process such a long input, these computations are inevitably long. How to speed up fuzzy computations?

One known way to speed up computations in general is to design a *hardware support* for them. This idea worked perfectly well, e.g., for floating point operations that had initially been implemented in software. So, *it is desirable to design hardware support for interval and fuzzy computations as well.*

**A word of warning: Hardware support is not sufficient.** Hardware support does bring a speed-up, and is, therefore, a great idea. However, the very fact that we have a hardware support of several basic operations with intervals, crisp sets, and/or with fuzzy sets, does not necessarily mean that we have improved the quality of the result (we are thankful to the anonymous referee who helped us clarify this point).

As we have shown in Subsection 1.2, if we start with an expression
$(1/4) \cdot [(x_1 + x_2)^2 - (x_1 - x_2)^2]$
and simply substitute interval operations instead of operations with numbers, we will get an overestimation irrespective on whether we implement these operations in software or in hardware. The only way to get the exact estimate is to transform this expression into an equivalent one $x_1 \cdot x_2$ for which interval computations give the exact estimate.

An even more striking example is a function of one variable defined as $f(x_1) = x_1 - x_1$. This function is, of course, identically equal to 0, so, for every interval $X_1$, we should get $f(X_1) = \{0\}$. However, if we take $X_1 = [0, 1]$, and apply interval subtraction, we will get $X_1 - X_1 = [0, 1] - [0, 1] = [-1, 1]$. Whether we are implementing interval subtractions in hardware or in software, we get an overestimation. Again, the only way to get the exact estimate is take into the consideration that the two terms in the original expression cancel each other, and thus, to transform the original expression $x_1 - x_1$ into an equivalent one 0.

So, in addition to hardware, we need some appropriate *symbolic reasoning engine* (of the type implemented in Macsyma or in Mathematica) that would transform the original expression into an equivalent one that will lead to the precise (or at least to a more accurate) estimate.

*At present*, designing such an engine is a more urgent and more potentially rewarding task than working on hardware. Currently, computers only allow unary and binary operations. So, what this engine will do is, given a function, transform it into a sequence of unary and binary operations. The *perfect* engine will output a transformation that leads to the best possible estimate (e.g., to the interval with the smallest possible overestimation). However, this "the best" does not necessarily mean that we will have no overestimation at all: As we will see later, for some functions of three and more variables, no matter how we transform these functions into a sequence of unary and binary operations, interval computations will always lead to an overestimation. This overestimation result is true not only for the existing computers, where only elementary arithmetic operations are hardware supported; this result (as we will show) is true for *any* computer that hardware supports

only unary and binary operations. So, for such functions, the only way to avoid overestimation is to implement operations with three or more interval (corr., fuzzy) operands in hardware.

For the resulting new computer, the computation scheme will include not only standard unary and binary operations, but some new operations (with interval or fuzzy operands) as well. Again, the very fact that we have added these new operations does not automatically mean that we will achieve the exact estimate: before we apply interval computations, we must *transform* the original computation scheme (that may be overestimating) into a new (equivalent) scheme with no overestimation. So, we will again need an appropriate *symbolic reasoning engine* for the new computer.

Summarizing, we can say that achieving *precise* interval and fuzzy computations is a three-step task:

- First, we must design a symbolic reasoning engine based on the *existing* hardware supported operations (namely, elementary arithmetic operations). This engine must transform the original expression (in terms of these elementary operations) for which interval and fuzzy computations overestimate into an equivalent expression that leads to more precise result $Y$.
- Second, we must select operations with three or more interval or fuzzy operands that need to be hardware supported in order to get precise results.
- Third, for these new operations included, we must design a *new* symbolic reasoning engine that will transform every algorithm into a sequence of elementary operations (arithmetic or new ones) for which interval (fuzzy) computations will lead to the precise result.

The first task – the design of the original engine is, in effect, being currently done in interval computations community (see, e.g., [10] and [14]). In the present paper, we consider the *second* task: the choice of the hardware operations to be hardware supported. We give only a partial answer to this task. As soon as this problem will be solved, the need will come for the *third task*: designing a new symbolic reasoning engine for the new computer.

**Hardware support of interval and fuzzy operations: a little bit of history.** The existing hardware support of *interval* operations is described in [23,36,37], and references therein. Usually, the supported operations are arithmetic operations, and the scalar (dot) product $a_1 \cdot b_1 + ... + a_n \cdot b_n$.

The first hardware implementation of operations with *fuzzy sets* has been developed by Yamakawa. For the current state of research, see, e.g., [49,50]; for applications, see,

e.g., [43] and [52]. This first implementation included several chips. The first single-chip hardware implementation of fuzzy operations have been proposed in [45] (for more recent results, see, e.g., [44] and [48]). Parallel hardware implementation has been proposed and described in [2]; see also [38] and [42]. These implementations, however, are mainly oriented towards *fuzzy control* problems (and not fuzzy data processing).

**What interval and fuzzy operations should we support for data processing?** It is impossible to implement in hardware fuzzy operations that correspond to *all* possible functions $f(x_1, ..., x_n)$. So, it is necessary to choose.

The natural idea is to describe all functions $f$ that are hardware supported on the existing computers, and to support the corresponding operations with fuzzy sets. Since usually, only unary and binary operations are hardware supported, we will thus have hardware implementation only of operations of one and two operands.

**What we are planning to do.** In this paper, we show that for intervals and for fuzzy sets, an implementation of only unary and binary operations is not sufficient in the following sense: for some functions $f$ and for some intervals (fuzzy sets) $X_i$, no matter how we represent the function $f$ as a composition of unary and binary operations, if we then apply the above-described methodology, the resulting estimate $\tilde{Y}$ will be different from the desired value $Y = f(X_1, ..., X_n)$. Therefore, if we want fuzzy data processing to be precise, operations with three or more fuzzy operands should also be implemented in hardware.

Some crisp operations with three or more crisp operands are already hardware supported: e.g., many computers contain a math co-processor that, among other things, hardware supports matrix operations, i.e., operations whose operands include an entire matrix (i.e., lots of numerical operands). For example, it is possible, given two arrays $a_1, ..., a_n$ and $b_1, ..., b_n$, to compute their dot (scalar) product $a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n$ by using a single operation of a math co-processor. For crisp numbers, the main purpose of using such operations is to speed up computations: in principle, we can use operations with two operands (in this case, addition and multiplication) and compute the same expression in several steps.

We are planning to show that in interval and fuzzy case, if we restrict ourselves to unary and binary operations only, then not only computations will slow down, but for some functions $f$, and for some intervals (fuzzy sets) $X_i$, we will not get the desired value $f(X_1, ..., X_n)$ at all.

### 1.4. Structure of the paper

In Section 2, we will give some general definitions. In Section 3, we will consider the simplest case when inputs are intervals, and when all operations that are hardware supported are smooth (differentiable) functions. For this case, we will prove that operations with one or two variables are not sufficient. The fact that functions are differentiable makes it possible not only to prove the negative results but also to describe the smallest possible order of the error that can be caused by such computations (linear, quadratic, etc). We also give a list of operations that need to be hardware supported so that we will be able to compute $Y$ with a better accuracy: this list consists of arithmetic operations and a weighted scalar product. In Section 4, these results will be generalized from crisp intervals to fuzzy sets. In Section 5 and 6, we show that even if we allow operations that are not differentiable, still operations with one or two operands will not be sufficient. In Section 7, we discuss the relation between these results and Kolmogorov's theorem (well known in mathematics) that every continuous function of three or more real variables can be represented as a composition of real-valued functions of one and two variables. For reader's convenience, all the proofs are placed in Section 8.

## 2. GENERAL DEFINITIONS

In this Section, we give general definitions that will be used in the following text.

### 2.1. Computation scheme

In this subsection, we will formalize the description of a general computation process (presented in Subsection 1.2) into a formal definition, and show (on an example) how this formalization is related to the original description.

**Definition 2.1.** *By a computation scheme $\mathcal{S}$ with $n$ initial values, and with operations with one or two operands (or, for short, simply a computation scheme), we mean a finite sequence $(S_{n+1}, S_{n+2}, ..., S_N)$ of expressions $S_i$ (called steps). Each step $S_i$ is an expression of one of the following three types:*

- *$r_i := c_i$, where $c_i$ is a real number;*
- *$r_i := f_i(r_j)$, where $f_i$ is a function of one variable (not necessarily everywhere defined), and $j < i$;*
- *$r_i := f_i(r_j, r_k)$, where $f_i$ is a function of two variables (not necessarily everywhere defined), $j < i$, and $k < i$.*

*If $\mathcal{S}$ is a computation scheme with $n$ initial values, and $x_1, ..., x_n$ are $n$ real numbers, we say that the result of applying $\mathcal{S}$ to $x_i$ is $y$, if $r_N = y$, where the sequence $r_i$ is defined as follows:*

- *if $i \le n$, then $r_i = x_i$:*
- *if $i > n$, then depending on the type of the rule $S_i$, we define $r_i$ as follows:*
  - *if the rule if $r_i := c_i$, then $r_i = c_i$;*
  - *if the rule is $r_i := f_i(r_j)$, then $r_i = f_i(r_j)$;*
  - *if the rule is $r_i := f_i(r_j, r_k)$, then $r_i = f_i(r_j, r_k)$.*

**Denotation.** For some $x_1, ..., x_n$, and for each $i \le N$, this Definition provides us with a certain value of $r_i$. This value will be denoted by $r_i(x_1, ..., x_n)$.

*Comment.* We have already mentioned that inside a computer, every algorithm is translated into a sequence of elementary operations. Since in the majority of computers, all elementary operations correspond to functions of one or two variables, an arbitrary algorithm computing $y = f(x_1, ..., x_n)$ can be thus represented as a computation scheme. For example, how in the above-given representation of the function $f(x_1, x_2, x_3) = (x_1 + x_2)^2 + 2 \cdot x_3$, we have:

- $r_4 = r_1 + r_2$ (here, $f_4 = +$, $j = 1$, $k = 2$);
- $r_5 = r_4^2$ (here, $f_5(x) = x^2$, $j = 4$);
- $r_6 = 2$ (here, we have a function of 0 variables, that computes a constant 2);
- $r_7 = r_6 * r_3$;
- $r_8 = r_5 + r_7$.

**Definition 2.2.** *Let $K \subseteq R^n$, and let $f$ be a function from $K$ to $R$. We say that a computation scheme $\mathcal{S}$ computes $f$ if for every $(x_1, ..., x_n) \in K$, the value that $\mathcal{S}$ computes (is defined and) is equal to $f(x_1, ..., x_n)$.*

## 2.2. Applying computation scheme to crisp sets (in particular, to intervals)

In this subsection, we will describe how a computation scheme can be applied to crisp sets; in particular, we will describe how it can be applied to intervals.

**Definition 2.3.** *Let $f(x_1, ..., x_n)$ be a function of $n$ real variables, and let $X_1 \subseteq R$, ..., $X_n \subseteq R$ be (crisp) sets. Then, the result $f(X_1, ..., X_n)$ of applying $f$ to the sets $X_1, ..., X_n$ is defined by the formula (1.1). If instead of the sets $X_i$, we have a (crisp) set $X \subseteq R^n$, then the result $f(X)$ of applying $f$ to $X$ is defined by the formula (1.2).*

**Definition 2.4.** *Assume that $\mathcal{S}$ is a computation scheme with $n$ input values, and that $X_q \subseteq R$, ..., $X_n \subseteq R$ are (crisp) subsets of $R$. We say that the result of applying $\mathcal{S}$ to $X_1, ..., X_n$ is $Y$ if $R_N = Y$, where the sequence of (crisp) sets $R_1, ..., R_N$ is defined as follows:*

- *if $i \le n$, then $R_i = X_i$;*
- *if $i > n$, then, depending of the type of the rule $S_i$, $R_i$ is defined as follows:*
  - *if $S_i$ is $r_i := c_i$, then $R_i = \{c_i\}$;*
  - *if the rule is $r_i := f_i(r_j)$, then $R_i = f_i(R_j)$;*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j > n$ or $k > n$, then $R_i = f_i(R_j, R_k)$.*

**Definition 2.5.** *Assume that $\mathcal{S}$ is a computation scheme with $n$ input values, and that $X$ is a (crisp) subset of $R^n$. We say that the result of applying $\mathcal{S}$ to $X$ is $Y$ if $R_N = Y$, where the sequence of (crisp) sets $R_1, ..., R_N$ is defined as follows:*

- *if $i \leq n$, then $R_i = \pi_i(X)$, where $\pi_i$ is a projection on $i-th$ component (i.e., $\pi_i(\vec{x}) = x_i$);*
- *if $i > n$, then, depending of the type of the rule $S_i$, $R_i$ is defined as follows:*
  - *if $S_i$ is $r_i := c_i$, then $R_i = \{c_i\}$;*
  - *if the rule is $r_i := f_i(r_j)$, then $R_i = f_i(R_j)$;*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j \leq n$ and $k \leq n$, then we take $R_i = f(\pi_{jk}(X))$, where $\pi_{jk}$ is a projection to $j-th$ and $k-th$ components (i.e., $\pi_{jk}(x_1, ..., x_j, ..., x_k, ..., x_n) = (x_j, x_k)$);*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j > n$ or $k > n$, then $R_i = f_i(R_j, R_k)$.*

*Comment.* The main difference between these two definitions is that when we have a set $X \subseteq R^n$, and we want to compute the set $f_i(X)$ of possible values of $f_i(x_1, ..., x_n)$, we must take into consideration the fact that not all values $(x_1, ..., x_n)$ with $x_i \in \pi_i(X)$ are possible.

### 2.3. Applying computation scheme to fuzzy sets

In the previous subsection, we described the result of applying a computation scheme to *crisp* sets. Let us now describe what a computation scheme will do if we apply it to *fuzzy* sets.

**Definition 2.6.** *Let $f(x_1, ..., x_n)$ be a function of $n$ real variables, and let $X_1 \subseteq R$, ..., $X_n \subseteq R$ be fuzzy sets. Then, the result $f(X_1, ..., X_n)$ of applying $f$ to the sets $X_1, ..., X_n$ is defined by the formula (1.4a). If instead of the fuzzy sets $X_i$, we have a fuzzy set $X \subseteq R^n$, then the result $f(X)$ of applying $f$ to $X$ is defined by the formula (1.3).*

**Definition 2.7.** *Assume that $\mathcal{S}$ is a computation scheme with $n$ input values, and that $X_1 \subseteq R$, ..., $X_n \subseteq R$ are fuzzy subsets of $R$. We say that the result of applying $\mathcal{S}$ to $X_1, ..., X_n$ is $Y$ if $R_N = Y$, where the sequence of fuzzy sets $R_1, ..., R_N$ is defined as follows:*

- *if $i \leq n$, then $R_i = X_i$;*
- *if $i > n$, then, depending of the type of the rule $S_i$, $R_i$ is defined as follows:*
  - *if $S_i$ is $r_i := c_i$, then $R_i = \{c_i\}$ (a crisp set);*
  - *if the rule is $r_i := f_i(r_j)$, then $R_i = f_i(R_j)$;*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j > n$ or $k > n$, then $R_i = f_i(R_j, R_k)$.*

**Definition 2.8.** *Assume that $\mathcal{S}$ is a computation scheme with $n$ input values, and that $X$ is a fuzzy subset of $R^n$. We say that the result of applying $\mathcal{S}$ to $X$ is $Y$ if $R_N = Y$, where the sequence of fuzzy sets $R_1, ..., R_N$ is defined as follows:*

- *if $i \le n$, then $R_i = \pi_i(X)$, where $\pi_i$ is a projection on $i-$th component (i.e., $\pi_i(\vec{x}) = x_i$);*
- *if $i > n$, then, depending of the type of the rule $S_i$, $R_i$ is defined as follows:*
  - *if $S_i$ is $r_i := c_i$, then $R_i = \{c_i\}$ (a crisp set);*
  - *if the rule is $r_i := f_i(r_j)$, then $R_i = f_i(R_j)$;*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j \le n$ and $k \le n$, then we take $R_i = f(\pi_{jk}(X))$, where $\pi_{jk}$ is a projection to $j-$th and $k-$th components (i.e., $\pi_{jk}(x_1, ..., x_j, ..., x_k, ..., x_n) = (x_j, x_k)$);*
  - *if the rule is $r_i := f_i(r_j, r_k)$, where $j > n$ or $k > n$, then $R_i = f_i(R_j, R_k)$.*

## 3. SIMPLEST CASE: SMOOTH OPERATIONS, INTERVAL UNCERTAINTY; HARDWARE SUPPORT OF UNARY AND BINARY OPERATIONS IS NOT SUFFICIENT; WEIGHTED SCALAR PRODUCT MUST ALSO BE SUPPORTED

In this section, we will consider the simplest case when inputs are *intervals*, and when all operations that are hardware supported are *smooth* (differentiable) functions. For this case, we will prove that hardware operations with one or two variables are not sufficient.

The fact that functions are differentiable makes it possible not only to prove the negative results but also to describe the smallest possible order of the error that can be caused by such computations (linear, quadratic, etc). We will also give a list of operations that need to be hardware supported so that we will be able to compute $Y$ with a better accuracy: this list consists of arithmetic operations and a weighted scalar product.

**Definition 3.1.**

- *We say that a computation scheme $\mathcal{S}$ is continuous if all the function $f_i$ are continuous.*
- *We say that a function $f$ is smooth if it is defined on an open set, is three times differentiable, and all its third order derivatives are continuous.*
- *We say that a computation scheme $\mathcal{S}$ is smooth if all the function $f_i$ are smooth.*

### 3.1. The main (negative) result: unary and binary operations are not sufficient

In this subsection, we will describe our first negative result: that for smooth operations on intervals, hardware unary and binary operations are not sufficient.

**Definition 3.2.** *Assume that a smooth computation scheme $\mathcal{S}$ computes a smooth function $f$ defined on an open set $K \subseteq R^n$. We say that $\mathcal{S}$ is precise for intervals if for all intervals $X_1, ..., X_n$ for which $X_1 \times ... \times X_n \subseteq K$, the result of applying $\mathcal{S}$ to $X_1, ..., X_n$ coincides with $f(X_1, ..., X_n)$.*

For example, a 1-step computation scheme that computes $f(x_1, x_2) = x_1 \cdot x_2$ by multiplying $x_1$ and $x_2$ is precise for intervals, while the computation scheme based on the expression $(1/4) \cdot [(x_1 + x_2)^2 - (x_1 - x_2)^2]$ is not. It can be proven that the above-given computation scheme for $f(x_1, x_2, x_3) = (x_1 + x_2)^2 + 2 \cdot x_3$ is precise for intervals. As we have already mentioned, for one and the same function, some computation scheme are precise, and some others are not. The question is: for a give function $f$, is there any computation scheme that *is* precise?

**Definition 3.3.** *We say that for a smooth function $f : R^n \to R$, smooth interval computations are precise, if there exists a smooth computation scheme that computes $f$ and that is precise for intervals. If such a smooth computation scheme does not exist, then we say that for this function $f$, smooth interval computations cannot be always precise.*

We will show that for many reasonable smooth functions, smooth interval computations cannot be always precise. To formulate our result, we will need a few denotations and definitions.

**Denotations.**

- By $f_{,i}$, we will denote $i-$th partial derivative of a function $f$.
- By $f_{,ij}$, we will denote the second partial derivative

$$f_{,ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}.$$

**Definition 3.4.**

- *A point $\vec{s}$ is called a stationary point of a function $f$ if $f_{,i}(\vec{s}) = 0$ for all $i$.*
- *A stationary point $\vec{s}$ of a function $f$ is called non-degenerate if the following two conditions are satisfied:*
  - *at this point $\vec{s}$, all components of the Hessian matrix $f_{,ij}(\vec{s})$ are different from 0;*
  - *at this point $\vec{s}$, the determinant of the Hessian matrix is different from 0.*

*Comment.* Almost all matrices satisfy these two properties (in the sense that the set of symmetric matrices for which they are not true forms a subspace of co-dimension 1 in the $n(n+1)/2-$dimensional set of all matrices). So, we can say that *almost all smooth functions with a stationary point have a non-degenerate stationary point.*

**THEOREM 3.1.** *If a smooth function $f(x_1, ..., x_n)$, $n \geq 3$, has a non-degenerate stationary point, then for this function $f$, smooth interval computations cannot be always precise.*

**Reformulation of this result in more informal (and hopefully, more understandable) terms.** In view of the previous comment, this result means that *for almost all smooth functions with a stationary point, smooth interval computations cannot be always precise.*

**Our result and known mathematical results.** The very fact that there exist smooth functions for which smooth interval computations are not always precise is not surprising: indeed, it is known (see, e.g., a survey [26]), that there exists a smooth function of three variables that cannot be represented as a composition of smooth functions of one or two variables. For such functions, a smooth computation scheme cannot be precise even for real numbers, and of course, it is not precise for intervals, because every real number $x_i$ can be viewed as a (degenerate) interval $[x_i, x_i]$. Our negative result is much broader than that: namely:

- Functions that cannot be represented as a composition of smooth unary and binary operations can be viewed rather as an *exception*: e.g., every polynomial can definitely be represented as such a composition.
- For smooth interval computations, we have shown that (in some reasonable sense) *almost all* functions (to be more precise, almost all functions with a stationary point) cannot be represented as compositions of smooth unary and binary interval operations. This result includes functions $f(x_1, ..., x_n)$ like quadratic polynomials, that can be represented as a composition for numerical $x_i$.

**What does this theorem tell us about the choice of operations for hardware implementation.** With respect to hardware support, Theorem 3.1 says the following: suppose that we have chosen a list of operations that we intend to implement in hardware (both as operations with numbers and as operations with intervals). On the resulting computer, every algorithm $f(x_1, ..., x_n)$ will thus be represented as a composition of the hardware supported operations $f_i$. If we want to compute the *interval* $f(X_1, ..., X_n)$, we will apply the same sequence of operations as in computing $f(x_1, ..., x_n)$, but with intervals instead of numbers (i.e., we will apply the computation scheme $\mathcal{S}$ that computes $f$ to intervals $X_1, ..., X_n$). For some computation schemes, e.g., for

$$x_1 \cdot x_2 = (1/4) \cdot [(x_1 + x_2)^2 - (x_1 - x_2)^2],$$

we may get an overestimate of $f(X_1, ..., X_n)$; for some others, hopefully, we will get a precise estimate. In view of the previously mentioned result, for some smooth functions, no smooth computation scheme will return the precise interval $f(X_1, ..., X_n)$.

We would like to choose a set of hardware supported operations in such a way that for each smooth function that has smooth computation schemes, at least one of these schemes will be precise for intervals (i.e., the above-described method will give exactly $f(X_1, ..., X_n)$). Theorem 3.1 tell us that we cannot achieve this goal if we only support unary and binary operations; so, *we must also implement some operations with three or more operands in hardware.*

### 3.2. Second negative result: if we only use unary and binary operations, we cannot even compute the main term correctly

According to Theorem 3.1, if a smooth function $f(x_1, ..., x_n)$ has a non-degenerate stationary point, and if we are only using unary and binary operations, then it is impossible to always compute $f(X_1, ..., X_n)$ precisely for intervals $X_i$. So, if a computation scheme $\mathcal{S}$ computes $f$, then the result $\tilde{Y}$ of applying $\mathcal{S}$ to some intervals $X_1, ..., X_n$ will be different from the desired interval $Y = f(X_1, ..., X_n)$. How different can it be?

Since we consider smooth functions, we can try to describe this difference in terms of the order: is it linear (first order), quadratic (second order), cubic (third order), etc, in $\Delta_i$? It could be that the difference is, say, of fifth order w.r.t. errors $\Delta_i$, and therefore, for practical purposes (this difference being much smaller than the interval itself) we could safely neglect it, and treat $\tilde{Y}$ as a good approximation for $Y$. Alas, the reality is not so good: it turns out that smooth interval computations (with unary and binary operations) do not even give a correct *main term* for $Y$.

To describe this result in mathematical terms, let us first describe the asymptotic of $Y$:

**PROPOSITION 3.1.** *Let $f(x_1, ..., x_n)$ be a smooth function. Then, the lower $f^-$ and upper $f^+$ bounds of the interval $f([x_1 - \Delta_1, x_1 + \Delta_1], ..., [x_n - \Delta_n, x_n + \Delta_n])$ satisfy the property $f^{\pm} = f(x_1, ..., x_n) \pm (|f_{,1}(\vec{x})|\Delta_1 + ... + |f_{,n}(\vec{x})|\Delta_n) + O(\Delta_i^2)$.*

**Definition 3.5.** *Assume that $\mathcal{S}$ is a smooth computation scheme for a smooth function $f(x_1, ..., x_n)$. We say that $\mathcal{S}$ always computes the main error term correctly if for every $\vec{x}$, the difference between the actual endpoints of the interval*

$$Y = f([x_1 - \Delta_1, x_1 + \Delta_1], ..., [x_n - \Delta_n, x_n + \Delta_n])$$

*and the values computed by applying $\mathcal{S}$ to intervals $X_i = [x_i - \Delta_i, x_i + \Delta_i]$ is $O(\Delta_i^2)$.*

*Comment.* In other words, we allow smooth interval computations not to be precise in the sense that their result can differ in terms that are *quadratic* (or of higher order) in $\Delta_i$, but we require that the *main term* in $\Delta_i$ be computed precisely.

**THEOREM 3.2.** *If a smooth function $f(x_1, ..., x_n)$, $n \geq 3$, has a non-degenerate stationary point, then for this function $f$, smooth interval computations cannot always compute the main error term correctly.*

### 3.3. Third negative result: if we only use unary and binary operations, we cannot even be locally asymptotically correct

An even stronger negative result can be proved. Namely, in our definition of what it means to compute the main term correctly, we required that the main term should be computed exactly. In general, the fact that a function is several times differentiable, means that we can approximate it by its Taylor polynomial. For example, if $f$ is twice differentiable, then in the neighborhood of a point $(s_1, ..., s_n)$, we can approximate the function $f(x_1, ..., x_n)$ by a linear function: $f(x_1, ..., x_n) = f(s_1, ..., s_n) + \sum f_{,i}(s_1, ..., s_n)(x_i - s_i) + O((x_i - s_i)^2)$. If the function is analytical, these Taylor polynomials actually converge to the function $f$. So, if by using smooth interval computations, we cannot compute the main term precisely, then maybe, we can at least compute correctly the first few terms in the expansion of the main term? I.e., e.g., we may be able to compute the main term with the accuracy of $O((x_i - s_i)^2)$ terms? Alas, even this, we cannot compute, as the following result shows.

**Definition 3.6.** *Let $f(\vec{x})$ be a smooth function, $\vec{s} = (s_1, ..., s_n)$ is a point in $R^n$, and $\mathcal{S}$ is a smooth computation scheme for $f$. We say that $\mathcal{S}$ is locally asymptotically correct (in computing the main error term) in the neighborhood of $\vec{s}$, if the difference between the actual endpoints of the interval $f([x_1 - \Delta_1, x_1 + \Delta_1], ..., [x_n - \Delta_n, x_n + \Delta_n])$ and the values computed by applying $\mathcal{S}$ to the intervals $X_i = [x_i - \Delta_i, x_i + \Delta_i]$ is $O(\Delta_i^2) + O(\Delta_i \cdot (x_j - s_j)^2)$.*

**THEOREM 3.3.** *If a smooth function $f(x_1, ..., x_n)$, $n \geq 3$, has a non-degenerate stationary point, then for this function $f$, there exists a point $\vec{s}$ such that no matter what smooth computation scheme $\mathcal{S}$ we choose to compute $f$, the resulting smooth interval computations are not locally asymptotically correct in the neighborhood of $\vec{s}$.*

### 3.4. Positive result: if we add weighted scalar product, smooth interval computations become locally asymptotically correct

Indeed, assume that in our definition of a computation scheme, we allow, in addition to unary and binary operations, weighted scalar product, i.e., an operation

$$a_1, ..., a_n, b_1, ..., b_n \to w_1 \cdot a_1 \cdot b_1 + ... + w_n \cdot a_n \cdot b_n, \tag{3.1}$$

where $n$ is an arbitrary positive integer, and $w_i$ are arbitrary real numbers (called *weights*). Before we give a formal definition, we must make one comment.

In operations with real numbers, if we can implement a binary operation $f(x_1, x_2)$, then we can automatically implement the function $g(x) = f(x, x)$ that is obtained by applying $f$ to two equal values: namely, to implement $g$, we can simply apply $f$ to two equal values. For interval operations, this idea does not always lead to an implementation of $g$. As an example, we can take subtraction $f(x_1, x_2) = x_1 - x_2$. For subtraction, $g(x) = f(x, x) = x - x = 0$; the corresponding interval operation is $f([x_1^-, x_1^+], [x_2^-, x_2^+]) = [x_1^- - x_2^+, x_1^+ - x_2^-]$. If we apply this operation to $[x_1^-, x_1^+] = [x_2^-, x_2^+] = [0, 1]$, we get $f([0, 1], [0, 1]) = [-1, 1]$. This result is different from the desired $g([0, 1]) = \{0\}$. Because of this comment, when we describe a computation scheme that involves a certain interval operation $f$, we must specifically include interval analogues of all functions that can be obtained from $f$ by applying it to equal values of the variables. As a result, for weighted scalar product, we arrive at the following definition:

**Definition 3.7.**

- *Let a function $f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{j-1}, x_j, x_{j+1}, ..., x_n)$ be given. We say that a function $g(x_1, ..., x_{j-1}, x_{j+1}, ..., x_n) = f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{j-1}, x_i, x_{j+1}, ..., x_n)$ with $n - 1$ variables is a simplified version of a function $f$; transition from $f$ to $g$ will be called a simplification.*
- *We say that a function $g$ is a version of a function $f$ if $g$ can be obtained from $f$ by a sequence of simplifications.*
- *By a weighted scalar product, we mean an operation (3.1).*
- *By a computation scheme with weighted scalar products, we mean a sequence $\mathcal{S}$ of expressions each of which is either a function of zero, one, or two variables (like in Definition 2.1), or an application of a weighted scalar product or of one of its versions.*

*Example.* If we use weighted scalar product and its versions, we can simplify the computation of the above-given expression $(x_1 + x_2)^2 + 2 \cdot x_3$ as follows: we still have $r_i = x_i$ for $i = 1, 2, 3$, and then:

- $r_4 = r_2 + r_3$ (this is a binary operation);
- $r_5 = 2$;
- $r_6 = r_4^2 + r_5 \cdot r_3$; here, to the values $r_4$, $r_5$, and $r_6$, we apply a function $f_6(y_1, y_2, y_3) = y_1^2 + y_2 \cdot y_3$ obtained by simplifying a weighted scalar product: $f_6(y_1, y_2, y_3) = f^*(y_1, y_1, y_2, y_3)$, where $f^*(y_1, y_4, y_2, y_3) = y_1 \cdot y_4 + y_2 \cdot y_3$ is a weighted scalar product with both weights equal to 1.

For such computation schemes, we can repeat definitions 2.2 (what it means that a scheme computes a function $f$), 2.3 (how to apply a scheme to intervals), and 3.6 (what it means to be locally asymptotically correct). Now, we can formulate our positive result:

**THEOREM 3.4.** *For every smooth function $f(x_1, ..., x_n)$, $n \geq 3$, and for every point $\vec{s}$ in its domain, there exists a computation scheme $\mathcal{S}$ with weighted scalar products for which the resulting smooth interval computations are locally asymptotically correct in the neighborhood of $\vec{s}$.*

*Comments.*

- *Good news: the result is applicable not only to intervals, but to arbitrary crisp sets as well.* From the proof of this theorem, one can easily see that the designed computation scheme is locally asymptotically correct not only for intervals $X_i = [x_i - \Delta_i, x_i + \Delta_i]$, but also for arbitrary crisp sets $X_i \subseteq [x_i - \Delta_i, x_i + \Delta_i]$.

- *Not so good news: We know what operations we need to implement in hardware, but we do not yet know how to implement them all.* Theorem 3.4 says that if we hardware support all weighted scalar products, then smooth interval computation becomes locally asymptotically correct. A word of warning: this result does not mean that we can immediately implement all these operations and make computations (asymptotically) precise, because we do not yet know how to implement all the necessary operations. Indeed, according to Definition 3.7, we need separate versions of a function to deal with each simplification obtained when two or more arguments to the function represent the same variable. For a function with $n$ arguments, this will require $n!$ hardware implementations of the function. When $n$ is large, $n!$ is so unrealistically large that it is practically impossible to have $n!$ different hardware devices that compute $n!$ different simplifications. So, to implement all these simplifications, we need a *flexible* implementation that will change when some of the arguments represent the same variable. At present, we do not know how to design such a flexible implementation. This implementation issue is an interesting open problem.

## 4. SMOOTH OPERATIONS, INDEPENDENT FUZZY SETS $X_1, ..., X_n$: HARDWARE SUPPORT OF UNARY AND BINARY OPERATIONS IS NOT SUFFICIENT; WEIGHTED SCALAR PRODUCT MUST ALSO BE SUPPORTED

In the previous section, we proved that unary and binary operations are not sufficient to support arbitrary smooth operations on *intervals*. In this section, we will show that,

similarly, unary and binary operations are not sufficient to describe smooth operations on (independent) *fuzzy sets.*

**Definition 4.1.** *Assume that a smooth computation scheme $\mathcal{S}$ computes a function $f$ defined on an open set $K \subseteq R^n$. We say that $\mathcal{S}$ is precise for independent fuzzy sets if for all fuzzy sets $X_1, ..., X_n$ for which $X_1 \times ... \times X_n \subseteq K$, the result of applying $\mathcal{S}$ to $X_1, ..., X_n$ coincides with $f(X_1, ..., X_n)$.*

**Definition 4.2.** *We say that for a smooth function $f : R^n \to R$, smooth computations are precise for independent fuzzy sets, if there exists a smooth computation scheme that computes $f$ and that is precise for fuzzy sets. If such a smooth computation scheme does not exist, then we say that for this function $f$, smooth computations cannot be always precise for independent fuzzy sets.*

**THEOREM 4.1.** *If a smooth function $f(x_1, ..., x_n)$, $n \geq 3$, has a non-degenerate stationary point, then for this function $f$, smooth computations cannot be always precise for independent fuzzy sets.*

*Comments.*

- Theorem 4.1 follows directly from Theorem 3.1: indeed, if a smooth computation scheme is precise for arbitrary independent fuzzy sets, then, in particular, it must be precise for crisp intervals, and this (according to Theorem 3.1) is impossible.
- Similarly, Theorems 3.2 and 3.3 show that smooth fuzzy computations with only unary and binary operations cannot even describe the main term in $f(X_1, ..., X_n)$ correctly.
- In many reasonable cases, extension principle that defines $Y = f(X_1, ..., X_n)$ for fuzzy sets $X_1, ..., X_n$ can be reformulated in terms of their (crisp) $\alpha-$*cuts* $X_i(\alpha) = \{x | \mu_{X_i}(x) \geq \alpha\}$: namely, $Y(\alpha) = f(X_1(\alpha), .., X_n(\alpha))$ (see [33]; for counterexamples, see [33] and [7]). So, if we add weighted scalar product to the list of hardware supported operations, then, due to Theorem 3.4, we will be able to get all $\alpha-$cuts $Y(\alpha)$ locally asymptotically correctly, and in this sense, we will be able to compute the fuzzy set $Y$ itself with the same accuracy.

## 5. GENERIC (NOT NECESSARY SMOOTH) OPERATIONS, CRISP SETS: HARDWARE SUPPORT OF UNARY AND BINARY OPERATIONS IS NOT SUFFICIENT

Let us now consider the general case of operations that are not necessarily smooth, and of the information that is not necessarily representable by independent sets $X_1, ..., X_n$.

**Definition 5.1.** *Assume that a computation scheme $\mathcal{S}$ computes a function $f$ defined on a set $K \subseteq R^n$. We say that $\mathcal{S}$ is precise for all (crisp) sets if for every (crisp) set $X \subseteq K$, the result of applying $\mathcal{S}$ to $X$ coincides with $f(X)$.*

*Comment.* We are going to prove that if our list of elementary operations includes only operations with one and two operands, and a function $f$ of three and more variables is (in some reasonable sense) non-degenerate, then no computation scheme is applicable to fuzzy processing (i.e., none of them will provide the exact fuzzy result). Let us describe what we mean by non-degenerate.

**Definition 5.2.** *Let $K \subseteq R^n$, $n \geq 3$. We say that a function $f : K \to R$ is degenerate if $K$ can be subdivided into finitely many subsets $K_i$ so that on each subset, $f$ coincides with some function of one or two variables (i.e., on which $f(x_1, ..., x_n) = g(x_j, x_k)$ for some $j$ and $k$ and for some function $g$). A function that is not degenerate will be called non-degenerate.*

*Comment.* As an example of a degenerate function, we can take $f(x_1, ..., x_n) = \max(x_1, ..., x_n)$. For this function, $K = R^n$ can be subdivided into the subsets $K_i$ in which $x_i$ is greater than or equal to all other values, and on each $K_i$, $f(x_1, ..., x_n) = x_i$ (i.e., is equal to a function of one variable).

The following two results show that many functions of three or more variables are non-degenerate:

**PROPOSITION 5.1.** *If a function $f(x_1, ..., x_n)$ of three or more variables is defined on a domain $K$, and on some subset $M \subseteq K$ with a non-empty interior, $f$ is strictly monotonic in each variable, then $f$ is non-degenerate.*

*Comment.* To describe non-degenerate functions, we need to recall a notion of a *real analytic* function: this means a function that can be represented as a sum of its Taylor series. All known elementary functions (arithmetic operations, sin, cos, exp, ln, etc) and their compositions are real analytic functions. It turns out that a real analytical function is non-degenerate if and only if it actually depends on all of its variables:

**PROPOSITION 5.2.** *If $f(x_1, \ldots, x_n)$ is a real analytic function of $n \geq 3$ variables, and $f$ is not equal to a function of $< n$ variables, then the function $f$ is non-degenerate (in the sense of Definition 5.2).*

*Examples.*

- A function $f(x_1, x_2, x_3) = \sin(x_1 + \exp(x_2 \cdot x_3))$ is non-degenerate, because it actually depends on each of its variables.
- In contrast, a function $f(x_1, x_2, x_3) = x_1 \cdot x_2$ is degenerate, because it does not depend on the variable $x_3$ at all and is thus equal to the function of two variables.

**THEOREM 5.1.** *If a function $f$ of three and more variables is non-degenerate, then no computation scheme that computes $f$ is precise for all crisp sets.*

*Comment.* This result is based on our Definition 2.1, in which we assumed that all elementary operations are operations with one or two (set-valued) operands. Thus, in case of incomplete knowledge, when we have *sets* of possible values of the variables, operations with one and two operands are not sufficient. This suggests that for this case, we need to implement hardware operations with 3 or more operands.

## 6. GENERIC (NOT NECESSARY SMOOTH) OPERATIONS, FUZZY SETS: HARDWARE SUPPORT OF UNARY AND BINARY OPERATIONS IS NOT SUFFICIENT

In the previous section, we proved that unary and binary operations are not sufficient to support generic (not necessarily smooth) operations on *crisp* sets. In this section, we will show that, similarly, unary and binary operations are not sufficient to describe generic operations on *fuzzy* sets.

**Definition 6.1.** *Assume that a computation scheme $\mathcal{S}$ computes a function $f$ defined on a set $K \subseteq R^n$. We say that $\mathcal{S}$ is precise for all fuzzy sets if for every fuzzy subset $X \subseteq K$, the result of applying $\mathcal{S}$ to $X$ coincides with $f(X)$.*

**THEOREM 6.1.** *If a function $f$ of three and more variables is non-degenerate, then no computation scheme that computes $f$ is precise for all fuzzy sets.*

*Comments.*

- Since crisp sets are a particular case of fuzzy sets, this theorem is a corollary of Theorem 5.1 (just like Theorem 4.1. is a corollary of Theorem 3.1).
- This result is based on our Definition 2.1, in which we assumed that all elementary operations are operations with one or two (fuzzy) operands. Thus, for soft computing, operations with one and two operands are not sufficient. This suggests that *for soft computing, we need to implement hardware operations with 3 or more operands.*

## 7. OUR RESULTS AND KOLMOGOROV'S THEOREM

### 7.1 What is Kolmogorov's theorem: in brief

The fact that every continuous function of three and more variables can be represented as a composition of functions of one or two variables (and can be thus computed by an appropriate computation scheme) has been first proved by Kolmogorov [15] as a solution to the famous Hilbert's problem: one of 22 problems that Hilbert has proposed in 1900 as a challenge to the XX century mathematics [13]. Kolmogorov's result was later improved in [40,41], and turned out to be applicable to theoretical and practical aspects of computation (see, e.g., [6,12,24,25,31,30]).

### 7.2. In some reasonable sense, Kolmogorov's theorem may not be extended to interval and fuzzy computations

Our negative theorems (3.1–3.3, 4.1, 5.1, and 6.1) show that *Kolmogorov's theorem might not be possible to extend* to *interval* or *fuzzy* cases: there are functions that cannot be represented as a composition of operations with one or two interval (resp. fuzzy) operands.

### 7.3. In some other sense (less computationally straightforward) we *can* extend Kolmogorov's theorem to intervals

The above results are about the following: we have a function; we describe its computations step-by-step, and substitute operations with intervals instead of operations with numbers.

In [32], the following result have been proven: if we *do not* follow the algorithm $f$ step-by-step, i.e., if we *do not* require that interval operations follow the flow of numerical computations, then Kolmogorov's theorem is true: namely, an arbitrary interval function *can* be represented as a composition of functions of one and two variables. The proof is rather simple: suppose that we have an interval-valued function $\mathbf{f}(\mathbf{x_1},...,\mathbf{x_n}) = [\mathbf{f}^-(\mathbf{x_1},...),\mathbf{f}^+(\mathbf{x_1},...)]$ of $n$ interval variables $\mathbf{x}_1 = [x_1^-, x_1^+], ..., \mathbf{x}_n = [x_n^-, x_n^+]$ . This means that we have *two* functions $f^-$ and $f^+$ of $2n$ *real* variables $x_1^-, ..., x_n^-, x_1^+, ..., x_m^+$. Each of these functions can be (due to Kolmogorov's theorem) represented as a composition of functions of one and two variables. So, we can do the following:

- first, apply the interval functions $-$ and $\top$ that transform an interval $[x^-, x^+]$ into $[x^-, x^-]$ and, correspondingly, $[x^+, x^+]$.
- follow the operations from Kolmogorov's theorem with these degenerate intervals, and get the numerical-valued functions $[f^-, f^-]$ and $[f^+, f^+]$ as the desired composition;
- apply a *combination operation* $\mathrm{comb}([a,a],[b,b]) = [a,b]$ to $[f^-, f^-]$ and $[f^+, f^+]$ and get the desired interval $f = [f^-, f^+]$.

With respect to hardware operations it means that in principle, we can restrict ourselves to unary and binary operations only, but in this case, to compute the interval $f(X_1, ..., X_n)$, we will *not* be able to *simply* follow the algorithm $f$ step-by-step: for each $f$, we will have to design a new method of interval (and therefore, for fuzzy) data processing.

*Comment.* An important open question:

- the result from [32] (described in this section) is proven for *intervals*;
- what happens in the fuzzy case?

## 8. PROOFS

### Proof of Proposition 3.1

Since the function $f$ is smooth, if $\Delta x_i$ are such that $|\Delta x_i| \le \Delta_i$, then

$$f(x_1 + \Delta_1, ..., x_n + \Delta x_n) = f_{\rm lin} + O(\Delta_i^2),$$

where we denoted $f_{\rm lin} = f(x_1, ..., x_n) + \Delta x_1 \cdot f_{,1} + ... + \Delta x_n \cdot f_{,n}$. The maximum of $f_{\rm lin}$ is attained when each of the component terms $\Delta x_i \cdot f_{,i}$ attains the largest value for $\Delta x_i \in [-\Delta_i, \Delta_i]$: for $f_{,i} > 0$, the maximum is attained when $\Delta x_i = \Delta_i$, and for $f_{,i} < 0$, the maximum is attained when $\Delta x_i = -\Delta_i$. In both cases, the maximum of $i-$th term is equal to $|f_{,i}|\Delta_i$. Therefore, the maximum of $f_{\rm lin}$ is equal to $f(x_1, ..., x_n) + |f_{,1}|\Delta_1 + ... + |f_{,n}|\Delta_n$. Hence, the maximum $f^+$ of $f$ is equal to this expression plus $O(\Delta_i^2)$. The result for $f^-$ is proved similarly. Q.E.D.

### Proof of Theorems 3.1–3.3

Theorems 3.1 and 3.2 follow from Theorem 3.3, so it is sufficient to prove Theorem 3.3. We will prove that the statement of this theorem is true for the non-degenerate stationary point $\vec{s}$. This will be proven by reduction to a contradiction. Namely, let us assume that there exist a smooth function with a non-degenerate stationary point $\vec{s}$ and a computation scheme $\mathcal{S}$ for which smooth interval computations are locally asymptotically correct in the neighborhood of $\vec{s}$. Each such scheme $\mathcal{S}$ can be characterized by an integer: namely, by the total number of computation steps. Among such schemes, there exists a scheme with the smallest possible value of this integer. Let us denote this scheme by $\mathcal{S}$, the function that is computed by this scheme by $f(x_1, ..., x_n)$, and the stationary point for this function by $\vec{s} = (s_1, .., s_n)$.

First, let us describe what type of expression we can get after each step of the smooth computation scheme.

**Definition 8.1.** *By a generalized linear function of $n$ variables $x_1, ..., x_n$, we mean a linear combination of functions 1, $x_1$, ..., $x_n$, and absolute values $|a_1x_1 + ... + a_nx_n|$ of homogeneous linear functions $a_1x_1 + ... + a_nx_n$.*

*Example.* A function $1 + x_1 + x_2 + |2x_1 - x_2| + |x + 3|$ is a generalized linear function.

**LEMMA 1.** *For every smooth computation scheme $\mathcal{S}$ that computes a function $f(x_1, ..., x_n)$, and for every point $\vec{s} \in R^n$, the endpoints $\tilde{f}^{\pm}$ of the result of smooth interval computations can be represented as*

$$f(x_1, ..., x_n) \pm \sum \tilde{f}_i(x_1 - s_1, ..., x_n - s_n)\Delta_i + O(\Delta_i^2) + O((x_i - s_i)^2\Delta_j)$$

*for some generalized linear functions $\tilde{f}_i$.*

**Proof of Lemma 1.** We will prove this lemma using induction over the number of steps in a smooth computation scheme.

*Induction base.* If a smooth computation scheme has not steps at all, this means that the function that we are computing simply coincides with one of the input variables. For an input variable $x_i$, each endpoint $x_i^{\pm}$ of the interval $[x_i - \Delta_i, x_i + \Delta_i]$ is already represented in the desired form with $\tilde{f}_i = \pm 1$, and $\tilde{f}_j = 0$ for $j \neq i$.

*Induction step.* Assume now that we have proved this result for all smooth computation schemes of length $\leq k$, and we want to prove it for smooth computation schemes of length $k+1$. Let $\mathcal{S}$ be any smooth computation scheme of length $k+1$. By definition of a smooth computation scheme, the final result of $\mathcal{S}$ is obtained in the last $((k+1)-$st) step by applying a smooth function of one or two variables to the results of previous computations. These results of previous computations are thus computable by computation schemes of length $\leq k$. Therefore, for these results, due to the induction assumption, the result can be represented in the desired form. Let us use these forms to describe the result of applying $(k+1)-$st step.

To prove it, we will consider two possible cases:

- The first case if when the last step of $\mathcal{S}$ consists of applying a smooth function of one variable.
- The second case if when the last step of $\mathcal{S}$ consists of applying a smooth function of two variables.

In the first case, $f(x_1, ..., x_n) = F(r(x_1, ..., x_n))$, and the result of applying smooth interval computations to $r$ is already known to be expressible in the form

$$[r(x_1, ..., x_n) - l + O(\Delta_i^2) + O((x_i - s_i)^2 \cdot \Delta_j), r(x_1, ..., x_n) + l + O(\Delta_i^2) + O((x_i - s_i)^2 \cdot \Delta_j)],$$

where $l = \tilde{r}_1\Delta_1 + ... + \tilde{r}_n\Delta_n$ is a linear expression in $\Delta_i$, with coefficients $\tilde{r}_i$ that are generalized linear function in $x_i - s_i$. Similarly to the proof of Proposition 3.1, we can prove that applying $F$ results in an interval $[y^-, y^+]$, where

$$y^{\pm} = F(r(x_1, ..., x_n)) \pm |F'(r(x_1, ..., x_n))| \cdot l + O((x_i - s_i)^2 \cdot \Delta_j).$$

To prove the desired result, let us first approximate $|F'(r(x_1, ..., x_n))|$ by a generalized linear function. To do that, we can first approximate the composition $F'(r(x_1, ..., x_n))$ by a generalized linear function. This part is easy: Since both $F'$ and $r$ are smooth functions, we conclude that $F'(r(x_1, ..., x_n))$ is also a smooth function, and therefore, it can be represented as $a_0 + a_1(x_1 - s_1) + ... + a_n(x_n - s_n) + O((x_i - s_i)^2)$.

- If $a_0 = 0$, then the absolute value of this function can be represented as

  $$|a_1(x_1 - s_1) + ... + a_n(x_n - s_n)| + O((x_i - s_i)^2),$$

  i.e., as a generalized linear function plus $O(...)$.
- If $a_0 \neq 0$, then $a_0 + \sum a_i(x_i - s_i) = a_0(1 + \sum(a_i/a_0)(x_i - s_i))$. The absolute value of the product can be represented as the product of the absolute values. When $x_i \to s_i$, then $\sum(a_i/a_0)(x_i - s_i) \to 0$, hence $\sum(a_i/a_0)(x_i - s_i) > -1$, and so, $|1 + \sum(a_i/a_0)(x_i - s_i)| = 1 + \sum(a_i/a_0)(x_i - s_i) + O((x_i - s_i)^2)$. Therefore, $|a_0 + a_1(x_1 - s_1) + ... + a_n(x_n - s_n)| = |a_0|(1 + \sum(a_i/a_0)(x_i - s_i)) + O((x_i - s_i)^2)$ is a linear (and thus, generalized linear) function of $x_i - s_i$.

In both cases, we represent $|F'(r(x_1, ..., x_n))|$ as a sum of the generalized linear function $\tilde{g}$ and an $O((x_i - s_i)^2)$ term. Therefore, the product $|F'(r(x_1, ..., x_n)))| \cdot l = |F'(r(x_1, ..., x_n)))| \cdot \tilde{r}_1\Delta_1 + ... + |F'(r(x_1, ..., x_n)))| \cdot \tilde{r}_n\Delta_n$ can be represented as

$$\tilde{g} \cdot \tilde{r}_1 \cdot \Delta_1 + ... + \tilde{g} \cdot \tilde{r}_n \cdot \Delta_n + O((x_i - s_i)^2\Delta_j).$$

For every $i$, the coefficient $\tilde{g} \cdot \tilde{r}_i$ at $\Delta_i$ is a product of two generalized linear functions. By definition, each generalized linear function is a sum of a constant (maybe 0), and a homogeneous first order part (i.e., terms $a_ix_i$ and $|a_1x_1 + ... + a_nx_n|$). The product $\tilde{g} \cdot \tilde{r}_i$ of two generalized linear functions $\tilde{g}$ and $\tilde{r}_i$ is thus a product of two sums and can, therefore, be represented as a sum of four terms:

- a product of two constants, which is a constant;
- a product of a constant term of $\tilde{g}$ and a homogeneous first order part of $\tilde{r}_i$, which is a generalized linear function;
- a product of a constant term of $\tilde{r}_i$ and a homogeneous first order part of $\tilde{g}$, which is also a generalized linear function;

- a product of a homogeneous first order parts of $\tilde{g}$ and $\tilde{r}_i$, which is $O((x_i - s_i)^2)$.

The sum $\tilde{f}_i$ of the first three products is generalized linear function; the fourth term lead to the term $O((x_i - s_i)^2 \Delta_i)$ in $\tilde{g} \cdot \tilde{r}_i \cdot \Delta_i$. Therefore, $\tilde{g} \cdot \tilde{r}_i \Delta_i = \tilde{f}_i \Delta_i + O((x_i - s_i)^2 \cdot \Delta_j)$ for a generalized linear function $\tilde{f}_i$. So, $|F'(r(x_1, ..., x_n))| \cdot l = L + O((x_i - s_i)^2 \cdot \Delta_j)$, where $L = \tilde{f}_1 \Delta_1 + ... + \tilde{f}_n \Delta_n$, $\tilde{f}_i$ are generalized linear, and therefore, the resulting interval is indeed equal to the desired expression $[f(x_1, ..., x_n) - L + O(...), f(x_1, ..., x_n) + L + O(...)]$.

The second case can be described in a similar manner. In this case, instead of $f = F(r)$, we have $f(x_1, ..., x_n) = F(r(x_1, ..., x_n), t(x_1, ..., x_n))$ for some functions $r$ and $t$ that have been computed on the previous steps. The resulting proof is similar. The lemma is proved. Q.E.D.

**LEMMA 2.** *Let $\mathcal{S}$ be a smooth computation scheme that computes a function $f(x_1, ..., x_n)$, and let $\vec{s} \in R^n$. Then, $|f_{,i}| \leq \tilde{f}_i + O((x_i - s_i)^2)$, where $\tilde{f}_i$ are generalized linear functions that appear (as coefficients at $\Delta_i$) in the description of the result of applying smooth interval computations.*

**Proof of Lemma 2.** For interval computations, the resulting interval always contains the actual interval of values of $f$. Due to Lemma 1, the endpoints of the resulting interval are $f \pm (\tilde{f}_1 \Delta_1 + ... + \tilde{f}_n \Delta_n) + O(...)$, and due to the Proposition, the endpoints of the interval of values of $f$ are $f \pm (|f_{,1}| \Delta_1 + ... + |f_{,n}| \Delta_n) + O(...)$. The fact that the first interval contains the second one means, in particular, that the width of the first interval is $\geq$ than the width of the second interval, i.e., that $\tilde{f}_1 \Delta_1 + ... + \tilde{f}_n \Delta_n \geq |f_{,1}| \Delta_1 + ... + |f_{,n}| \Delta_n + O(...)$. If we fix $i$, choose $\Delta_i = 1$ for this $i$, and $\Delta_j = 0$ for all $j \neq i$, we get the desired inequality. Q.E.D.

**LEMMA 3.** *Let $\mathcal{S}$ be a smooth computation scheme that computes a function $f(x_1, ..., x_n)$, and let $\vec{s} \in R^n$. Then, the following statements are equivalent to each other:*

- *for $\mathcal{S}$, smooth interval computations are locally asymptotically correct in the neighborhood of $\vec{s}$;*
- *$|f_{,i}| = \tilde{f}_i + O((x_i - s_i)^2)$, where $\tilde{f}_i$ are generalized linear functions that appear (as coefficients at $\Delta_i$) in the description of the result of applying smooth interval computations.*

**Proof of Lemma 3.** This result immediately follows from Lemma 1 and Proposition 3.1. Q.E.D.

**Proof of Theorem 3.3 itself.** Let us now prove Theorem 3.3 itself. By definition, each step of a smooth computation scheme $\mathcal{S}$ that does not assign a constant value to a variable

(i.e., that is not of the type $R_i = \{c_i\}$), consists of applying a function of one or two variables either to initial data $x_i$, or to the values computed on one of the previous steps.

The last step of the scheme $\mathcal{S}$ cannot be of the form $R_i = \{c_i\}$ since otherwise the function would have no non-degenerate stationary point. Therefore, the last step of the scheme $\mathcal{S}$ can consists of applying either:

- a function of one variable, or
- a function of two variables.

Let us prove the theorem for these two cases.

**First case: the last step consists of applying a function of one variable.** Let us first show that this last step cannot consist of applying a smooth function of one variable. We will prove this statement by reduction to a contradiction. Assume that the last step of $\mathcal{S}$ does consist in applying a smooth function of one variable $F$ to the result $r(x_1, ..., x_n)$ of some previous step. In this case, $f(x_1, ..., x_n) = F(r(x_1, ..., x_n))$. Let us prove that $r(\vec{x})$ has a non-degenerate stationary point (at the same $\vec{s}$), and that for this function $r$, smooth interval computations are locally asymptotically correct in the neighborhood of $\vec{s}$.

Indeed, we know that $\vec{s}$ is a stationary point for $f$, i.e., that $f_{,i}(s_1, ..., s_n) = 0$. Since $f(x_1, ..., x_n) = F(r(x_1, ..., x_n))$, we have $f_{,i}(s_1, ..., s_n) = F'(r(s_1, ..., s_n)) \cdot r_{,i}(s_1, ..., s_n)$. Similarly, $f_{,ij}(\vec{s}) = F''(r(\vec{s}))r_{,i}(\vec{s})r_{,j}(\vec{s}) + F'(r(\vec{s})) \cdot r_{,ij}(\vec{s})$. Since $f_{,i}(\vec{s}) = 0$, we can conclude that either $F'(r(\vec{s})) = 0$, or $r_{,i}(\vec{s}) = 0$. In the first case, $f_{,ij} = F''(\vec{s})r_{,i}r_{,j}$, and the determinant of the Hessian at $\vec{s}$ is 0, which contradicts to our assumption that $\vec{s}$ is a non-degenerate point. Therefore, $F'(\vec{s}) \neq 0$, and $r_{,i}(\vec{s}) = 0$, and $\vec{s}$ is a stationary point of the function $r(x_1, ..., x_n)$.

Since $F'(r(\vec{s})) \neq 0$ and $r_{,i}(\vec{s}) = 0$, from $f_{,ij}(\vec{s}) = F''(r(\vec{s}))r_{,i}(\vec{s})r_{,j}(\vec{s}) + F'(r(\vec{s})) \cdot r_{,ij}(\vec{s})$, we can conclude that $f_{,ij}(\vec{s}) = F'(r(\vec{s})) \cdot r_{,ij}(\vec{s})$, and $r_{,ij}(\vec{s}) = C \cdot f_{,ij}(\vec{s})$, where $C = 1/F'(r(\vec{s}))$. Since $\vec{s}$ is a non-degenerate stationary point of $f$, we have $f_{,ij} \neq 0$, hence $r_{,ij}(\vec{s}) = C \cdot f_{,ij}(\vec{s}) \neq 0$. Similarly, from $\det |f_{,ij}| \neq 0$, we conclude that $\det |r_{,ij}| \neq 0$. So, $\vec{s}$ is a non-degenerate stationary point for the function $r$.

Let us now show that for the smooth computation scheme that leads to $r$, smooth interval computations are locally asymptotically correct in the neighborhood of $\vec{s}$. Indeed, let $\tilde{r}_i$ be the generalized linear functions that appear (as coefficients at $\Delta_i$) in the description of the result of applying smooth interval computations to $r$. This means that after we reach $r$, the resulting interval is equal to $r \pm (\tilde{r}_1\Delta_1 + ... + \tilde{r}_n\Delta_n) + O(...)$. Due to Proposition 3.1, after applying the function $F$ to this interval, we get

$$F(r) \pm |F'(r(x_1, ..., x_n))|(\tilde{r}_1\Delta_1 + ... + \tilde{r}_n\Delta_n) + O(...).$$

Now, since $F'(r)$ is a smooth function, we get

$$F'(r(x_1,...,x_n)) = F'(r(\vec{s})) + \sum F''(r(\vec{s}))r_{,i}(x_i - s_i) + O((x_i - s_i)^2).$$

Since $\vec{s}$ is a stationary point for $r$, we have $r_{,i}(\vec{s}) = 0$, and therefore, $F'(r(x_1,...,x_n)) = F'(r(\vec{s})) + O((x_i - s_i)^2)$ and $|F'(r(x_1,...,x_n))| = |F'(r(\vec{s}))| + O((x_i - s_i)^2)$. So, the result of applying smooth interval computations to $\mathcal{S}$ is equal to $F(r) \pm |F'(r(\vec{s}))|(\tilde{r}_1\Delta_1 + ... + \tilde{r}_n\Delta_n) + O(...)$. Since for $f$ and $\mathcal{S}$, smooth interval computations are locally asymptotically correct, we can (due to Lemma 3) conclude that for all $i$, $|F'(r(\vec{s}))| \cdot \tilde{r}_i(\vec{x}) = |f_{,i}(\vec{x})| + O(...)$. But, since $f(\vec{x}) = F(r(\vec{x}))$, we have $f_{,i}(\vec{x}) = F'(r(\vec{x}))r_{,i}(\vec{x})$ and $|f_{,i}(\vec{x})| = |F'(r(\vec{x}))| \cdot |r_{,i}(\vec{x})|$. We already know that $|F'(r(x_1,...,x_n))| = |F'(r(\vec{s}))| + O((x_i - s_i)^2)$. Therefore, $|f_{,i}(\vec{x})| = |F'(r(\vec{s}))| \cdot |r_{,i}(\vec{x})| + O(...)$. So, from

$$|F'(r(\vec{s}))| \cdot \tilde{r}_i(\vec{x}) = |f_{,i}(\vec{x})| + O(...) = |F'(r(\vec{s}))| \cdot |r_{,i}(\vec{x})| + O(...),$$

we can conclude that $\tilde{r}_i(\vec{x}) = |r_{,i}(\vec{x})| + O(...)$, and hence, due to Lemma 3, that for $r$, smooth interval computations are locally asymptotically correct.

Since $r$ is computed on a *previous* step of the smooth computation scheme $\mathcal{S}$, the number of steps that lead to $r$ is smaller than the number of computation steps in a scheme $\mathcal{S}$, and this contradicts to our choice of $\mathcal{S}$ as the shortest smooth computation scheme that leads to a smooth non-degenerate function for which smooth interval computations are locally asymptotically correct. So, the last step of our smooth computation scheme $\mathcal{S}$ cannot consist of applying a function of one variable.

**Second case: the last step consists of applying a function of two variables.** To complete our proof, let us show that the last step cannot consist of applying a function of two variables. Indeed, suppose that the last step consists of applying a smooth function $F$ of two variables to two functions $r(\vec{x})$ and $t(\vec{x})$ that have been computed on a previous computation step. In this case, $f_{,i} = F_{,r} \cdot r_{,i} + F_{,t} \cdot t_{,i}$, where by $F_{,r}$ and $F_{,t}$, we denoted partial derivatives of $F$ w.r.t. $r$ and $t$.

Let us first show that for $\vec{s}$, both partial derivatives of $F$ cannot be 0. Indeed, suppose that they are. For $f = F(r,t)$, the Hessian matrix is equal to $f_{,ij}(\vec{x}) = F_{,rr}(\vec{x})r_{,i}r_{,j} + 2F_{,rt}(\vec{x})r_{,i}t_{,j} + F_{,tt}(\vec{x})t_{,i}t_{,j} + F_{,r}(\vec{x})r_{,ij} + F_{,t}(\vec{x})t_{,ij}$. In particular, for $\vec{x} = \vec{s}$, taking into consideration that $F_{,r} = F_{,t} = 0$, we conclude that $f_{,ij}(\vec{s}) = F_{,rr}(\vec{s})r_{,i}r_{,j} + 2F_{,rt}(\vec{s})r_{,i}t_{,j} + F_{,tt}(\vec{s})t_{,i}t_{,j}$. If we choose vectors $r_{,i}(\vec{s})$ and $t_{,i}(\vec{s})$ as the first two elements of the base, then the Hessian matrix will only have 11, 12, and 22 components. Therefore, the determinant

of the Hessian matrix $f_{,ij}(\vec{s})$ will be 0, which contradicts to our assumption that the stationary point $\vec{s}$ is non-degenerate. This contradiction shows that the derivatives $F_{,r}(\vec{s})$ and $F_{,t}(\vec{s})$ cannot be both equal to 0. Hence, we only need to consider the following three subcases:

- $F_{,r}(r(\vec{s}),t(\vec{s})) \neq 0$ and $F_{,t}(r(\vec{s}),t(\vec{s})) \neq 0$.
- $F_{,r}(r(\vec{s}),t(\vec{s})) \neq 0$ and $F_{,t}(r(\vec{s}),t(\vec{s})) = 0$.
- $F_{,r}(r(\vec{s}),t(\vec{s})) = 0$ and $F_{,t}(r(\vec{s}),t(\vec{s})) \neq 0$.

We will analyze these three subcases separately.

**First subcase of the second case: both partial derivatives of $F$ are different from 0 at $\vec{s}$.** Let us first consider the case when both partial derivatives of $F$ are different from 0 at $\vec{s}$. Let us show that in this case, $r_{,i}(\vec{s}) = t_{,i}(\vec{s}) = 0$. Indeed, the interval that corresponds to $r$ is equal to $r \pm \sum \tilde{r}_i \Delta_i + O(...)$, and the interval that corresponds to $t$ is equal to $t \pm \sum \tilde{t}_i \Delta_i + O(...)$. Due to Proposition 3.1, the interval that corresponds to $f$ is thus equal to $f \pm \sum \tilde{f}_i \Delta_i + O(...)$, where $\tilde{f}_i(\vec{x}) = |F_{,r}(\vec{x})| \cdot \tilde{r}_i(\vec{x}) + |F_{,t}(\vec{x})| \cdot \tilde{t}_i(\vec{x}) + O((x_i - s_i)^2)$. Let us analyze this equality for $\vec{x} = \vec{s}$. For this $\vec{x}$, the $O$ term is equal to 0, so we have an exact equality $\tilde{f}_i(\vec{x}) = |F_{,r}(\vec{x})| \cdot \tilde{r}_i(\vec{x}) + |F_{,t}(\vec{x})| \cdot \tilde{t}_i(\vec{x})$. Since smooth interval computations are locally asymptotically correct for $f$, we have (due to Lemma 2) $\tilde{f}_i = |f_{,i}| + O((x_i - s_i)^2)$. In particular, for $\vec{x} = \vec{s}$, we have $\tilde{f}_{,i}(\vec{s}) = |f_{,i}(\vec{x})| = 0$. So, the left-hand side of the equality is 0: $0 = |F_{,r}(\vec{s})| \cdot \tilde{r}_i(\vec{s}) + |F_{,t}(\vec{s})| \cdot \tilde{t}_i(\vec{s})$. The coefficients $\tilde{r}_i$ and $\tilde{t}_i$ are non-negative for all $\vec{x}$. So, 0 is equal to the sum of two non-negative products ($|F_{,r}(\vec{s})| \cdot \tilde{r}_i(\vec{s})$ and $|F_{,t}(\vec{s})| \cdot \tilde{t}_i(\vec{s})$). The only way for this to happen is when both products are equal to 0. Since $F_{,r} \neq 0$ for $\vec{x} = \vec{s}$, from $|F_{,r}(\vec{s})| \cdot \tilde{r}_i(\vec{s}) = 0$, it follows that $r_{,i}(\vec{s}) = 0$. Similarly, we can conclude that $t_{,i}(\vec{s}) = 0$.

Due to $r_{,i}(\vec{s}) = t_{,i}(\vec{s}) = 0$, we can (similarly to the case of $f = F(r)$) conclude that $F(\vec{x}) = F(\vec{s}) + O((x_i - s_i)^2)$. Therefore, $\tilde{f}_i(\vec{x}) = |F_{,r}(\vec{s})| \tilde{r}_i(\vec{x}) + |F_{,t}(\vec{s})| \tilde{t}_i(\vec{x}) + O(...)$. On the other hand, $\tilde{f}_i = |f_{,i}| + O(...)$, where $f_{,i}(\vec{x}) = F_{,r}(\vec{x}) r_{,i}(\vec{x}) + F_{,t}(\vec{x}) t_{,i}(\vec{x})$, and due to $F(\vec{x}) = F(\vec{s}) + O((x_i - s_i)^2)$, we can rewrite this equality as $f_{,i}(\vec{x}) = F_{,r}(\vec{s}) r_{,i}(\vec{x}) + F_{,t}(\vec{s}) t_{,i}(\vec{x})$. So,

$$\tilde{f}_i = |f_{,i}| + O(...) = |F_{,r}(\vec{s}) r_{,i}(\vec{x}) + F_{,t}(\vec{s}) t_{,i}(\vec{x})| + O(...).$$

Since $|p + q| \leq |p| + |q|$, we conclude that

$$\tilde{f}_i \leq |F_{,r} \cdot r_{,i} + F_{,t} \cdot t_{,i}| + O(...) \leq |F_{,r}| \cdot |r_{,i}| + |F_{,t}| \cdot |t_{,i}| + O(...).$$

Due to Lemma 2, $|r_{,i}| \leq \tilde{r}_i + O(...)$ and $|t_{,i}| \leq \tilde{t}_i + O(...)$. Therefore,

$$\tilde{f}_i \leq |F_{,r} \cdot r_{,i} + F_{,t} \cdot t_{,i}| + O(...) \leq |F_{,r}| \cdot |r_{,i}| + |F_{,t}| \cdot |t_{,i}| + O(...) \leq |F_{,r}| \cdot \tilde{r}_i + |F_{,t}| \cdot \tilde{t}_i + O(...).$$

But the right-hand side of this inequality is already known to be equal to $\tilde{f_i}(\vec{x})$ (modulo quadratic terms). Therefore,

$$\tilde{f_i} \le |F_{,r} \cdot r_{,i} + F_{,t} \cdot t_{,i}| + O(...) \le |F_{,r}| \cdot |r_{,i}| + |F_{,t}| \cdot |t_{,i}| + O(...) \le \tilde{f_i} + O(..).$$

In this chain of inequalities the starting and the ending expressions coincide, and hence, they are all equalities. So,

$$|F_{,r} \cdot r_{,i} + F_{,t} \cdot t_{,i}| = |F_{,r}| \cdot |r_{,i}| + |F_{,t}| \cdot |t_{,i}| + O(...).$$

The only case when $|p+q| = |p| + |q|$ is when $p$ and $q$ are of the same sign. So, we can conclude that the linear parts of the expressions $F_{,r}(\vec{s}) \cdot r_{,i}(\vec{x})$ and $F_{,t}(\vec{s}) \cdot t_{,i}(\vec{x})$ are of the same sign. Clearly, the sign of the linear part of

$$f_{,i} = F_{,r}(\vec{s}) \cdot r_{,i}(\vec{x}) + F_{,t}(\vec{s}) \cdot t_{,i}(\vec{x}) + O(...)$$

will be of the same sign. This linear part is $\sum f_{,ij}(\vec{s})(x_j - s_j)$. Similarly, we can express the linear parts of $r_{,i}$ and $t_{,i}$. We know that the numbers $F_{,r}(\vec{s})$ and $F_{,t}(\vec{s})$ are different from 0. Let us assume that both numbers are positive (the cases when one of them is negative can be treated in a similar manner). In this case, if a linear term in $f_{,i}$ is positive, then the linear terms in $r_{,i}$ and $t_{,i}$ are also non-negative. In other words, if $\sum f_{,ij}(\vec{s})(x_j - s_j) > 0$, then $\sum f_{,ij}(\vec{s})(x_j - s_j) \ge 0$. Turning to a limit, we can conclude that if $\sum f_{,ij}(\vec{s})(x_j - s_j) \ge 0$, then $\sum f_{,ij}(\vec{s})(x_j - s_j) \ge 0$. Similarly, if $\sum f_{,ij}(\vec{s})(x_j - s_j) \le 0$, then $\sum f_{,ij}(\vec{s})(x_j - s_j) \le 0$. Therefore, if $\sum f_{,ij}(\vec{s})(x_j - s_j) = 0$, then we have both $\sum f_{,ij}(\vec{s})(x_j - s_j) \ge 0$ and $\sum f_{,ij}(\vec{s})(x_j - s_j) \le 0$, hence, we have both $\sum r_{,ij}(\vec{s})(x_j - s_j) \ge 0$ and $\sum r_{,ij}(\vec{s})(x_j - s_j) \le 0$, and thence, $\sum r_{,ij}(\vec{s})(x_j - s_j) = 0$.

So, if $\sum f_{,ij}(\vec{s})(x_j - s_j) = 0$, then $\sum r_{,ij}(\vec{s})(x_j - s_j) = 0$. In geometric terms, the condition means that a vector $x_j - s_j$ is orthogonal to the vector $\vec{f_i}$ with coordinates $f_{,i1}(\vec{s}), ..., f_{,in}(\vec{s})$. Similarly, the conclusion means that a vector $x_j - s_j$ is orthogonal to the vector $\vec{r_i}$ with coordinates $r_{,ij}(\vec{s})$. Since $\vec{s}$ is a non-degenerate stationary point, the vector $\vec{f_i}$ is not zero, and therefore, vectors that are orthogonal to $\vec{f_i}$ form a (hyper)plane. A vector $\vec{r_i}$ is orthogonal to every vector from this (hyper)plane, and is, therefore, collinear with $\vec{f_i}$. This means that for every $i$, there exists a constant $c_i$ such that $r_{,ij}(\vec{s}) = c_i f_{,ij}(\vec{s})$ for all $i$ and $j$.

The matrix of second derivatives is symmetric: $r_{,ij} = r_{,ji}$. Therefore, for every $i$ and $j$, $c_i f_{,ij} = c_j f_{,ji}$. Since second derivatives of $f_{,ij}$ also form a symmetric matrix, we have

$c_i f_{,ij} = c_j f_{,ij}$. Due to the fact that $\vec{s}$ is a non-degenerate stationary point for $f$, we conclude that $f_{,ij} \neq 0$ and therefore, $c_i = c_j$ for all $i$ and $j$. Let us denote the common value of all $c_i$ by $c$. Then, $r_{,ij}(\vec{s}) = c f_{,ij}(\vec{s})$. Similarly, $t_{,ij}(\vec{s}) = c' f_{,ij}(\vec{s})$. The coefficients $c$ and $c'$ cannot be both equal to 0, because then, we would have $f_{,ij}(\vec{s}) = F_{,r}(\vec{s}) r_{ij}(\vec{s}) + F_{,t}(\vec{s}) t_{,ij}(\vec{s}) = 0$. So, either $c \neq 0$, or $c' \neq 0$. If $c \neq 0$, then $r_{,ij}(\vec{s}) = c f_{,ij}(\vec{s})$ is a non-degenerate matrix with non-zero determinant and all elements non-zero. Therefore, $r$ is a function that has a non-degenerate stationary point at $\vec{s}$, and that can be computed in fewer steps than $f$, which contradicts to our choice of $f$. Similarly, if $c' \neq 0$, then $t$ is a function that contradicts to our choice of $f$. In both cases, we get a contradiction.

**Second and third subcases of the second case: only one partial derivative of $F$ is different from 0 at $\vec{s}$.** We have derived a contradiction for the case when both partial derivatives $F_{,r}(\vec{s}) \neq 0$ and $F_{,t}(\vec{s}) \neq 0$. To complete the proof of the theorem, we must deduce a contradiction for the case when one of these partial derivatives is equal to 0. Without losing generality, we can assume that $F_{,r}(\vec{s}) \neq 0$ and $F_{,t}(\vec{s}) = 0$.

In this subcase, from the equality $\tilde{f_i}(\vec{x}) = |F_{,r}(\vec{x})| \cdot \tilde{r_i}(\vec{x}) + |F_{,t}(\vec{x})| \cdot \tilde{t_i}(\vec{x})$ (that can be proven exactly like in the first subcase), we conclude that $r_{,i}(\vec{s}) = 0$.

Since $F_{,t}(\vec{s}) = 0$, we can conclude that $F_{,t}(\vec{s}) = \sum a_j (x_j - s_j) + O((x_i - s_i)^2)$. Therefore, only zeroth-order terms in $t_{,i}(\vec{x})$ need to be taken into consideration, and we have

$$f_{,i}(\vec{x}) = \sum f_{,ij}(x_j - s_j) + O(...) = F_{,r}(\vec{s}) \sum r_{,ij}(\vec{s})(x_j - s_j) + (\sum a_j (x_j - s_j)) t_{,i}(\vec{s}) + O(...).$$

Similarly to the first subcase, we can prove that the two linear components of this sum must be of the same sign as $f_i$, and therefore, that $r_{,ij} = c f_{,ij}$ and $a_j t_{,i} = c' f_{,ij}$. The coefficient $c'$ cannot be different from 0, because then, we would have $f_{,ij} = (1/c') a_j t_{,i}$, and $\det |f_{,ij}| = 0$,which contradicts to our assumption that the Hessian matrix $f_{,ij}$ is non-degenerate. Therefore, $c' = 0$, and hence, $c = 0$ is impossible. Therefore, $r_{,ij} = (1/c) f_{,ij}$, and $r$ is a function that has a non-degenerate stationary point at $\vec{s}$, and that can be computed in fewer steps than $f$, which contradicts to our choice of $f$.

We have proven the result for the first case, and for all subcases of the second case; therefore, the theorem is proven. Q.E.D.

### Proof of Theorem 3.4

Since the function $f$ is smooth, for every point $\vec{s}$, we have $f(\vec{x}) = f_{\text{quadr}}(\vec{x}) + O((x_i - s_i)^3)$, where

$$f_{\text{quadr}} = f(\vec{s}) + \sum_i f_{,i}(\vec{s})(x_i - s_i) + \frac{1}{2} \sum_{i,j} f_{,ij}(x_i - s_i)(x_j - s_j).$$

Therefore (similarly to the proof of Proposition 3.1), we can conclude that the difference between the intervals $f(X_1, ..., X_n)$ and $f_{\rm quadr}(X_1, ..., X_n)$ is also $O((x_i - s_i)^3)$. The function $f_{\rm quadr}$ is quadratic in $x_i$, so, we can represent it as

$$f_{\rm quadr}(x_1, ..., x_n) = a_0 + \sum_i a_i x_i + \sum_{i,j} a_{ij} x_i x_j$$

for some real numbers $a_i$ and $a_{ij}$. If we define $a_{0i} = a_i$ for $i = 0, 1, ..., n$, then we can represent this expression as $f_{\rm quadr}(x_1, ..., x_n) = g(x_0, x_1, ..., x_n)$ for $x_0 = 1$ and

$$g(x_0, ..., x_n) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{ij} x_i x_j.$$

The function $g$, in its turn, can be represented as a result of applying $n$ simplifications $x_i = y_i$, $1 \le i \le n$, to a weighted scalar product

$$f^*(x_0, x_1, ..., x_n, y_0, y_1, ..., y_n) = \sum_{i=0}^{n} \sum_{j=0}^{n} a_{ij} x_i y_j$$

with weights $a_{ij}$.

So, $f_{\rm quadr}$ can be computed using a computation scheme $\mathcal{S}$ with weighted scalar products. Since the difference between the intervals $f(X_1, ..., X_n)$ and $f_{\rm quadr}(X_1, ..., X_n)$ is $O((x_i - s_i)^3)$, we can conclude that smooth interval computations described by the scheme $\mathcal{S}$ are locally asymptotically correct for the original function $f$. Q.E.D.

**Proof of Proposition 5.1**

1. Let us prove this Proposition by reduction to a contradiction. Namely, assume that $f : K \to R$ is strictly monotonic in each variable, and that $f$ is degenerate. By definition, this means that $K$ can be represented as a union of finitely many sets $K_1, ..., K_p$ on each of which $f$ is equal to a function of two of fewer variables. Let's deduce a contradiction from here.

2. Since the interior of $M \subseteq K$ is non-empty, the set $M$ contains a ball.

Inside the ball, we can place a cube $[a_1, b_1] \times [a_2, b_2] \times ... \times [a_n, b_n]$. Let's divide each side $[a_i, b_i]$ of the cube into $p + 1$ equally distanced values of $x_i$: $a_i$, $a_i + (b_i - a_i)/p$, $a_i + 2 \cdot (b_i - a_i)/p$, ..., $a_i + p \cdot (b_i - a_i)/p = b_i$. By combining these values for different $i$, we get $(p + 1)^n$ different points that all belong to the cube and therefore, to $K$. Let us denote the set of all these points by $P$.

3. Let us take one of the sets $K_i$, and let us estimate how many of the points from $P$ belong to $K_i$. Since $f$ is degenerate of $K_i$, it means that on $K_i$ $f$ is equal to a function of two or fewer variables. But $f$ is a function of $n \geq 3$ variables. So, this means, that for $\vec{x} \in K_i$, $f$ cannot depend on all the variables. Hence, it does not depend on one of the variables. Let us denote one of such variables by $x_j$. The fact that for $\vec{x} \in K_i$, $f$ does not depend on $x_j$, means that if we have two different points with different values of $x_j$ and equal values of all other coordinates, then the value of $f$ for both point will be the same.

4. Formally, if $\vec{x} = (x_1, ..., x_{j-1}, x_j, x_{j+1}, ..., x_n) \in P \subseteq M$ and

$$\vec{y} = (x_1, ..., x_{j-1}, x_j + \Delta x_j, x_{j+1}, ..., x_n) \in M,$$

where $\Delta x_j \neq 0$, then $f(\vec{x}) = f(\vec{y})$.

But on $M$, $f$ is strictly monotonic in each variable. This means, in particular, that $f$ is either strictly increasing in $x_j$, or strictly decreasing in $x_j$. In both cases, $f(\vec{x}) \neq f(\vec{y})$. This contradiction with $f(\vec{x}) = f(\vec{y})$ shows that no such pairs $(\vec{x}, \vec{y})$ are possible. In other words, if we fix the values of $n-1$ coordinates (all of them except for $x_j$), i.e., if we fix the values $x_1, ..., x_{j-1}, x_{j+1}, ..., x_n$, then at most one point with these values of $n-1$ coordinates belongs to $K_i$.

5. For $P$, if we fix the values of all the coordinates but one, then, we can choose $p+1$ different values of $x_j$ and hence, have $p+1$ different points from $P \subseteq K$. At most one of them belongs to $K_i$. Therefore, $K_i$ contains at most $1/(p+1)$ part of all the points from $P$.

6. The same inequality is true for each of the sets $K_i$. So, totally, $p$ sets $K_1$, ..., $K_p$ contain $\leq p/(p+1)$ of all the points from $P$. Since $p/(p+1) < 1$, it means that there are points from $P$ (and hence from $K$) that are not covered by any of the sets $K_i$. This contradicts to our assumption that $f$ is degenerate and $K = \cup K_i$.

This contradiction shows that our assumption was false, and so $f$ is not degenerate. Q.E.D.

**Proof of Proposition 5.2**

Let us show that if a real analytic function $f(x_1, ..., x_n)$ does not coincide with a function of less than $n$ variables, then it is non-degenerate in the sense of Definition 5.2. Indeed, an arbitrary real analytic function $g(x_1, ..., x_n)$ has the following property (similar to complex analytical functions): it is either identically equal to 0, or it is equal to 0 on a set of (Lebesgue) measure 0 (i.e., it is different from 0 on a set of full measure) (see, e.g., [17]).

Also, for an arbitrary real analytic function, each of its derivatives is also real analytic. In particular, the partial derivative $\partial f/\partial x_1$ of the given function is real analytic. If it was identically equal to 0, then $f$ would not depend on $x_1$ at all. Therefore, according to the above-cited result, the set of points $(x_1, ..., x_n)$ on which this derivative is equal to 0 is of Lebesgue measure 0. Similarly, for each $i = 2, ..., n$, the set of all points $(x_1, ..., x_n)$, at which $i-$th partial derivative $\partial f/\partial x_i$ is equal to zero, is also of measure 0.

Therefore, the union of these $n$ sets, i.e., the set of all points on which at least one of the partial derivatives is different from 0, is also of measure 0 (as a union of measure-zero sets). Therefore, there exists a point $(x_1^{(0)}, ..., x_n^{(0)})$ that does not belong to this union. By definition of the union it means that in this point, all $n$ partial derivatives are different from 0.

These derivatives are real analytic and therefore, continuous. Therefore, there exists a spherical neighborhood $M$ of this point $(x_1^{(0)}, ..., x_n^{(0)})$ in which sign of each of the partial derivatives coincides with its sign at the point $(x_1^{(0)}, ..., x_n^{(0)})$. For those $i$ for which this sign is positive ($\partial f/\partial x_i > 0$), $f$ is strictly increasing in $x_i$. For those $i$ for which this sign is negative ($\partial f/\partial x_i < 0$), $f$ is strictly decreasing in $x_i$. So, for all points from a set $M$ with a non-empty interior, the function $f$ is strictly monotonic in each variable. Therefore, according to Proposition 5.1, this function $f$ is non-degenerate. Q.E.D.

**Proof of Theorem 5.1**

1. Let us prove this theorem by reduction to a contradiction. Namely, we will assume that there exists a non-degenerate function $f$ and a computation scheme $\mathcal{S}$ that computes $f$ and that is precise for all (crisp) sets, and we will deduce a contradiction from this assumption.

To deduce this contradiction, we will use the following observation: According to our definitions, the phrase "$\mathcal{S}$ is precise for all (crisp) sets" means that for every (crisp) set $X \subseteq R^n$, the result $R_N$ of applying $\mathcal{S}$ to $X$ coincides with $f(X)$.

In this proof, we will need the following two Lemmas:

**LEMMA 4.** *Assume that a function $f : K \to R$ ($K \subseteq R^n$) is computed by a composition scheme $\mathcal{S}$, and $X \subseteq K$. Let's denote by $R_N$ the result of applying $\mathcal{S}$ to $X$. Then, $f(X) \subseteq R_N$.*

**Proof of Lemma 4.** This Lemma is similar to the Main Theorem of Interval Computations (see, e.g., [29]), and is proved similarly: namely, using induction, one can then prove that for every $i$, $r_i(X) \subseteq R_i$. For $i = N$, we get the desired result. Q.E.D.

**LEMMA 5.** *Assume that $k$ is an integer, $f : R \to R$, $g : R^k \to R$, and $h : R^k \to R$ is a composition of $f$ and $g$ (i.e., $h(\vec{x}) = f(g(\vec{x}))$). Then, for every (crisp) set $X \subseteq R^k$, $h(X) = f(g(X))$.*

**Proof of Lemma 5** follows directly from the definitions. Q.E.D.

2. We will need the following auxiliary notion: by a *complexity* of a computation scheme $\mathcal{S} = (S_{n+1}, ..., S_N)$, we will understand the number $N$.

We assumed that there exists a computation scheme that computes a non-degenerate function of $n$ variables and that is precise for all (crisp) sets. Out of all computation schemes with this property, there exists a one with the smallest possible complexity $N$. Let's choose one of these "simplest" schemes. In the following text, this chosen scheme will be denoted by $\mathcal{S}_s$ ($s$ stands for *simplest*). The corresponding function will be denoted by $f_s$.

*Comment.* A computation scheme $\mathcal{S}_s$ consists of the rules of the type $r_i := c_i$, $r_i := f_i(r_j)$, and $r_i := f_i(r_j, r_k)$. In principle, the corresponding functions $f_i$ can be defined everywhere. However, when we apply this computation scheme to compute the value of the function $f_s$ that is defined on some set $K$, we will use only the values of $f_i$ for $r_j \in r_i(K)$; or, for the function of two variables, only the values for $(r_j, r_k) \in D_{jk}$, where

$$D_{jk} = \{(a, b) \,|\, \exists \vec{x}\, (\vec{x} \in K \,\&\, r_j(\vec{x}) = a \,\&\, r_k(\vec{x}) = b\}.$$

Therefore, to simplify our proofs, we will assume in the following text that $f_i$ is defined only for these values.

It is easy to check that if initially $\mathcal{S}_s$ computed $f_s$ and was precise for all (crisp) sets, then after such a restriction on $f_i$ it still has the same properties.

3. Depending on what the final step $S_N$ of $\mathcal{S}_s$ is, we have the following five possibilities:

- $N \le n$;
- $N > n$ and $r_N := c_N$;
- $N > n$ and $r_N := f_N(r_j)$, where $j < N$;
- $N > n$ and $r_N := f_N(r_j, r_k)$, where $j \le n$ and $k \le n$;
- $N > n$ and $r_N := f_N(r_j, r_k)$, where $j > n$ or $k > n$.

In the following five subsections, we will prove that in all these five cases, we have a contradiction with our initial assumption.

4. $N \le n$.

In this case, $f_s(x_1, ..., x_n) = r_N(x_1, ..., x_n) = x_N$. So, $f_s$ depends only on one of its variables. Hence, $f_s$ is degenerate, which contradicts to the assumption that it is non-degenerate.

5. $N > n$ and $r_N := c_N$.

In this case, $f_s(x_1, ..., x_n) = r_N(x_1, ..., x_n) = c_N$ does not depend on any variables at all. Therefore, it is degenerate.

6. $N > n$ and $r_N := f_N(r_j)$, where $j < N$.

Let us prove (by considering all possible cases) that this case is impossible. Depending on the case, we will prove it either directly, or by "merging" the last step with one of the previous ones, and thus coming up with a new computation scheme that is simpler than the scheme $\mathcal{S}_s$ that is by definition the simplest possible (so, we have a contradiction).

6.1. If $j \le n$, then $r_j = x_j$, and $f_s = f_N(r_j) = f_N(x_j)$ is a function of one variable (namely, $x_j$) and is thus degenerate, which contradicts to our choice of $f_s$.

6.2. If $j > n$, and $j-$th step is $r_j := c_j$, then $f_s = r_N = f_N(c_j) =$const, i.e., $f_s$ is also degenerate.

6.3. If $j > n$, and $j-$th step is $r_j := f_j(r_k)$ for some $k < j$, then $r_N = f_N(f_j(r_k))$. In other words, $r_N = \tilde{f}_N(r_k)$, where by $\tilde{f}_N$, we denoted the composition of $f_N$ and $f_j$.

6.3.1. If $k \le n$, then $f_s$ is again a function of one variable $x_k$ (i.e., degenerate).

6.3.2. If $k > n$, then, we can replace the original computation scheme $\mathcal{S}$ with the following simplified one: $\tilde{\mathcal{S}} = (S_{n+1}, ..., S_k, \tilde{S}_N)$, where by $\tilde{S}_N$, we denoted the following step: $r_N := \tilde{f}_N(r_k)$. Because of our formulas, this scheme computes exactly the same function $f_s$. The fact that this scheme computes the same function and is precise for all (crisp) sets, follows from Lemma 5.

Since we deleted at least one step ($S_j$), this new scheme has a smaller complexity that $\mathcal{S}_s$. This contradicts to our choice of $\mathcal{S}_s$ as the computation scheme with the smallest possible complexity that computes a non-degenerate function of 3 or more variables and that is precise for all (crisp) sets.

6.4 If $j > n$, and $j-$th step is $r_j := f_j(r_k, r_l)$ for some $k, l < j$, then $r_N = f_N(f_j(r_k, r_l)) = \tilde{f}_N(r_k, r_l)$, where by $\tilde{f}_N$, we denoted a composition $\tilde{f}_N(a, b) = f_N(f_j(a, b))$. In this case, we can also delete one step from the computation scheme and thus arrive at the contradiction.

7. $N > n$ and $r_N := f_N(r_j, r_k)$, where $j \le n$ and $k \le n$.

In this case, $f_s(x_1, ..., x_n) = r_N(x_1, ..., x_n) = f_N(x_j, x_k)$. Hence, $f_s$ depends on only two of its variables, and is therefore degenerate.

8. $N > n$ and $r_N := f_N(r_j, r_k)$, where $j > n$ or $k > n$.

8.1. We assumed that $\mathcal{S}_s$ computes $f_s$, and that $\mathcal{S}_s$ is precise for all (crisp) sets. This means, in particular, that for each input set $X \subseteq K$, $R_N = f_s(X)$. In particular, if we take arbitrary two elements $\vec{x} \in K$ and $\vec{y} \in K$, then this equality must be true for a 2-point set $X = \{\vec{x}, \vec{y}\}$.

8.2. For this choice of $X$, the *right-hand side* of the equality $R_N = f_s(X)$ (i.e., the set $f_s(X)$) is easy to describe: it consists of two values $f_s(\vec{x}) = f_N(r_j(\vec{x}), r_k(\vec{x}))$ and $f_s(\vec{y}) = f_N(r_j(\vec{y}), r_k(\vec{y}))$.

8.3. Now, let's find an element of the *left-hand side*. By definition of $R_N$, this set is equal to $R_N = f_N(R_j, R_k)$. Due to Lemma 4, $R_j \supseteq r_j(X)$, and $R_k \supseteq r_k(X)$. Since $r_j(\vec{x}) \in r_j(X)$, we can thus conclude that $r_j(\vec{x}) \in R_j$. Similarly, we can conclude that $r_k(\vec{y}) \in R_k$. Therefore, $f_N(r_j(\vec{x}), r_k(\vec{y})) \in f_N(R_j, R_k) = R_N$.

Since $R_N = f_s(X)$, every element of $R_N$ must coincide with one of the two elements of $f_s(X)$. In particular, for the above-discovered element, it means the following:

*For every $\vec{x} \in K$ and $\vec{y} \in K$, $f_N(r_j(\vec{x}), r_k(\vec{y}))$ is either equal to $f_N(r_j(\vec{x}), r_k(\vec{x}))$, or it is equal to $f_N(r_j(\vec{y}), r_k(\vec{y}))$.*

8.4. This statement enables us to make the following conclusion about the function $f_N(a, b)$:

*If we can find $\vec{x}$ and $\vec{y}$ such that $a = r_j(\vec{x})$ and $b = r_k(\vec{y})$, then either $f_N(a, b) = f_N(a, r_k(\vec{x}))$, or $f_N(a, b) = f_N(r_j(\vec{y}), b)$.*

8.5. For each $a \in r_j(K)$, there exists a vector $\vec{x}$ for which $r_j(\vec{x}) = a$ (it is possible that several such vectors exist). For different vectors $\vec{x}$, the value $r_k(\vec{x})$ may also be different. For each $a$, let us pick one of these values $r_k(\vec{x})$ and denote it by $g_{kj}(a)$.

Similarly, for each $b \in r_k(K)$, we will pick a vector $\vec{x}$ with the property that $r_k(\vec{x}) = b$, and denote the value $r_j(\vec{x})$ for thus picked $\vec{x}$ by $g_{jk}(b)$.

8.6. Using these denotations, we can reformulate the statement from 8.4 as follows:

*For every* $a \in r_j(K)$ *and* $b \in r_k(K)$, *either* $f_N(a,b) = f_N(a, g_{kj}(a))$, *or* $f_N(a,b) = f_N(g_{jk}(b), b)$.

If we denote $h_1(a) = f_N(a, g_{kj}(a))$ and $h_2(b) = f_N(g_{jk}(b), b)$, then we can further simplify this conclusion:

*For every* $(a,b) \in D_{jk}$, *either* $f_N(a,b) = h_1(a)$, *or* $f_N(a,b) = h_2(b)$.

8.7. In particular, this property is true if we choose an arbitrary $\vec{x} \in K$ and take $a = r_j(\vec{x})$ and $b = r_k(\vec{x})$.

Let us denote by $K_1$ the set of all $\vec{x} \in K$ for which $f_N(r_j(\vec{x}), r_k(\vec{x})) = h_1(r_j(\vec{x}))$, and by $K_2$, the set of all values $\vec{x} \in K$ for which $f_N(r_j(\vec{x}), r_k(\vec{x})) = h_2(r_k(\vec{x}))$. Then, this property means that $K = K_1 \cup K_2$.

8.8. Since the function $f$ defined on $K$ is non-degenerate, its restriction to either $K_1$ or $K_2$ is also non-degenerate.

Indeed, if it were not true, then we would be able to describe both $K_1$ and $K_2$ (and hence, their union) as the union of finitely many subsets on which $f_s$ is degenerate. On the other hand, we assumed that $f_s : K \to R$ is non-degenerate, which means that for $K$, such a representation is impossible.

8.9. Let us choose a set $K_i$ (i.e., $K_1$ or $K_2$) for which the restriction of $f_s$ is non-degenerate. For this set, we can form the restriction of $f_s$ and $\mathcal{S}_s$. For this restriction, we can take $r_N := h_1(r_j)$ (or $r_N := h_2(r_k)$) as a last step of the computation scheme (instead of the step $r_N := f_N(r_j, r_k)$), and the resulting computation scheme will still compute the restriction of $f_s$, and it will still precise for all (crisp) sets.

If we were able to compute a non-degenerate function $f_s$ by another computation scheme $\mathcal{S}_1$ whose complexity is $< N$ computation steps, then we would get a contradiction with our choice of $N$ as the smallest possible complexity of a computation scheme that computes a non-degerenate function. Therefore, the resulting computation scheme is still the "simplest" in the sense that its complexity $N$ is the smallest possible among all computation schemes that compute non-degenerate functions.

So, we arrive at the situation where we have the "simplest" computation scheme, and the function at the last step is a function of one variable; we have already proved (in part 6 of this proof) that such situation leads to a contradiction.

9. So, in all five cases, we arrive at a contradiction. Hence, our initial assumption is false, namely, the assumption that a non-degenerate function of 3 or more variables can

be computed by a computation scheme (with unary and binary operations only) that is precise for all (crisp) sets. Q.E.D.

## CONCLUSIONS

**Theoretical.** The main *theoretical* result of this paper is that functions of several interval or fuzzy variables cannot always be computed precisely if we use only operations with one or two interval (resp. fuzzy) variables. Moreover, for smooth functions $f$, we show that even the main term in the result of fuzzy (interval) computations cannot always be computed correctly. The accuracy of interval and fuzzy data processing drastically improves if we add weighted scalar product to the list of elementary (hardware supported) operations.

For numerical operands, scalar product is already hardware supported in modern computers: by math co-processors. There have been successful attempts to hardware support scalar product of interval operands.

**Practical.** Therefore, our main *practical* recommendation is that for fuzzy data processing, it is desirable to hardware support (weighted) scalar product of fuzzy operands as well (at least, some operations with three or more fuzzy operands).

**Acknowledgments.** This work was partially supported by NSF grants No. CDA-9015006 and No. EEC-9322370, by a Grant No. PF90–018 from the General Services Administration (GSA), administered by the Materials Research Institute and the Institute for Manufacturing and Materials Management, and by a NASA grant No. NAG 9-757. The authors are greatly thankful to Paul Kainen, Baker Kearfott, Vera Kurkova, and Reza Langari for valuable discussions, and to the anonymous referees for their extremely valuable suggestions and help.

## REFERENCES


[1] O. Artbauer, "Application of interval, statistical, and fuzzy methods to the evaluation of measurements", *Metrologia*, 1988, Vol. 25, pp. 81–86.

[2] H. Dhirf and D. Sarkar, "Fuzzy arithmetic on systolic arrays", *Parallel Computing*, 1993, Vol. 19, pp. 1283–1301.

[3] W. M. Dong, W. L. Chiang, H. C. Shah, "Fuzzy information processing in seismic hazard analysis and decision making", *International Journal of Soil Dynamics and Earthquake Engineering*, 1987, Vol. 6, No. 4., pp. 220–226.

[4] W. Dong and F. Wong, "Fuzzy weighted averages and implementation of the extension principle", *Fuzzy Sets and Systems*, 1987, Vol. 21, pp. 183–199.

[5] D. Dubois and H. Prade. *Fuzzy sets and systems: theory and applications,* Academic Press, N.Y., London, 1980.

[6] H. L. Frisch, C. Borzi, G. Ord, J. K. Percus, and G. O. Williams, "Approximate Representation of Functions of Several Variables in Terms of Functions of One Variable", *Physical Review Letters*, 1989, Vol. 63, No. 9, pp. 927–929.

[7] R. Fuller and T. Keresztfalvi, "On generalization of Nguyen's theorem", *Fuzzy Sets and Systems*, 1990, Vol. 4, pp. 371–374.

[8] W. A. Fuller, *Measurement error models*, J. Wiley & Sons, New York, 1987.

[9] A. A. Gaganov, "Computational complexity of the range of the polynomial in several variables", *Cybernetics*, 1985, pp. 418–421.

[10] R. Hammer, M. Hocks, U. Kulisch, D. Ratz, *Numerical toolbox for verified computing. I. Basic numerical problems*, Springer Verlag, Heidelberg, N.Y., 1993.

[11] E. R. Hansen, *Global optimization using interval analysis*, Marcel Dekker, N.Y., 1992.

[12] R. Hecht-Nielsen, "Kolmogorov's Mapping Neural Network Existence Theorem", *IEEE International Conference on Neural Networks*, San Diego, SOS Printing, 1987, Vol. 2, pp. 11–14.

[13] D. Hilbert, "Mathematical Problems, lecture delivered before the International Congress of Mathematics in Paris in 1900", translated in *Bull. Amer. Math, Soc.*, 1902, Vol. 8, pp. 437–479.

[14] R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[15] A. N. Kolmogorov, "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition", *Dokl. Akad. Nauk SSSR*, 1957, Vol. 114, pp. 369–373.

[16] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications*, Prentice

Hall, Upper Saddle River, NJ, 1995.

[17] V. Kreinovich, "On the problem of recovering the $\psi-$function in non-relativistic quantum mechanics", *Teoreticheskaya i Mathematicheskaya Fizika*, 1976, Vol. 28, No. 1, pp. 56–64 (in Russian); English translation: *Theoretical and Mathematical Physics*, 1976, Vol. 8, No. 7, pp. 56–64.

[18] V. Kreinovich (ed.), *Reliable Computing*, 1995, Supplement (Extended Abstracts of APIC'95: International Workshop on Applications of Interval Computations, El Paso, TX, Febr. 23–25, 1995).

[19] V. Kreinovich, Ching-Chuang Chang, L. Reznik, G. N. Solopchenko, "Inverse problems: fuzzy representation of uncertainty generates a regularization", *Proceedings of NAFIPS'92: North American Fuzzy Information Processing Society Conference, Puerto Vallarta, Mexico, December 15–17, 1992*, NASA Johnson Space Center, Houston, TX, 1992, Vol. II, pp. 418–426.

[20] V. Kreinovich, C. Quintana, L. Reznik, *Gaussian membership functions are most adequate in representing uncertainty in measurements*, In: *Proceedings of NAFIPS'92: North American Fuzzy Information Processing Society Conference, Puerto Vallarta, Mexico, December 15–17, 1992*, NASA Johnson Space Center, Houston, TX, 1992, Vol. II, pp. 618–624.

[21] V. Kreinovich et al, *What non-linearity to choose? Mathematical foundations of fuzzy control*, Proceedings of the 1992 International Conference on Fuzzy Systems and Intelligent Control, Louisville, KY, 1992, pp. 349–412.

[22] V. Kreinovich and L. K. Reznik, "Methods and models of formalizing a priori information (on the example of processing measurements results)", In: *Analysis and Formalization of Computer Experiments, Proceedings of the Mendeleev Metrology Institute*, 1986, pp.37–41 (in Russian).

[23] U. Kulisch, G. Bohlender, "Features of a hardware implementation of an optimal arithmetic", In: U. Kulisch, W. L. Miranker (eds), *A new approach to scientific computation*, Academic Press, Orlando, FL, 1983, pp. 269–290.

[24] V. Kurkova, "Kolmogorov's Theorem Is Relevant", *Neural Computation*, 1991, Vol. 3, pp. 617–622.

[25] V. Kurkova, "Kolmogorov's Theorem and Multilayer Neural Networks", *Neural Networks*, 1992, Vol. 5, pp. 501–506.

[26] G. G. Lorentz, "The 13-th problem of Hilbert", in: F. E. Browder (ed.), *Mathematical Developments Arizing from Hilbert's Problems*, American Math. Society, Providence, RI, 1976, Part 2, pp. 419–430.

[27] R. E. Moore, *Automatic error analysis in digital computation*, Lockheed Missiles and

Space Co. Technical Report LMSD-48421, Palo Alto, CA, 1959.

[28] R. E. Moore, C. T. Yang, *Interval analysis*, Lockheed Missiles and Space Co. Technical Report LMSD-285875, Palo Alto, CA, 1959.

[29] R. E. Moore, *Methods and applications of interval analysis*, SIAM, Philadelphia, 1979.

[30] M. Nakamura, R. Mines, V. Kreinovich, "Guaranteed Intervals for Kolmogorov's Theorem (and Their Possible Relation to Neural Networks)", Interval Computations, 1993, No. 3, pp. 183–199.

[31] M. Ness, "Approximative versions of Kolmogorov's superposition theorem, proved constructively", *J. Comput. Appl. Math.*, 1993.

[32] V. M. Nesterov, "Interval analogues of Hilbert's 13th problem", In: *Abstracts of the Int'l Conference Interval'94, St. Petersburg, Russia, March 7–10, 1994*, 185–186.

[33] H. T. Nguyen, "A note on the extension principle for fuzzy sets", *J. Math. Anal. and Appl.*, 1978, Vol. 64, pp. 359–380.

[34] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, CRC Press, Boca Raton, Florida, 1996 (to appear).

[35] S. Rabinovich, *Measurement errors: theory and practice*, American Institute of Physics, N.Y., 1993.

[36] M. J. Schulte and E. E. Swartzlander, Jr., "Parallel Hardware Designs for Correctly Rounded Elementary Functions", *Interval Computations*, 1993, No. 4, pp. 65–88.

[37] M. J. Schulte and E. E. Swartzlander, Jr., "Design and applications for variable-precision, interval arithmetic coprocessors", In: V. Kreinovich (ed.), *Reliable Computing*, 1995, Supplement (Extended Abstracts of APIC'95: International Workshop on Applications of Interval Computations, El Paso, TX, Febr. 23–25, 1995), pp. 166–172.

[38] S. M. Shah and R. Horvath, *A hardware digital fuzzy inference engine using standard integrated circuits*, Information Sciences, 1994, Vol. 1, pp. 1–7.

[39] G. N. Solopchenko, "Formal metrological components of measuring systems", *Measurement*, 1994, Vol. 13, pp. 1–12.

[40] D. A. Sprecher, "On the Structure of Continuous Functions of Several Variables", *Transactions Amer. Math. Soc.*, 1965, Vol. 115, No. 3, pp. 340–355.

[41] D. A. Sprecher, "An Improvement in the Superposition Theorem of Kolmogorov", *Journal of Mathematical Analysis and Applications*, 1972, Vol. 38, pp. 208–213.

[42] H. Surmann *et al.*, "What kind of hardware is necessary for a fuzzy rule based system?", In: *Proceedings of the FUZZ-IEEE'94 International Conference*, Orlando, FL, July 1994, Vol. 1, pp. 274–278.

[43] M. Takahashi, E. Sanchez, R. Bartolin, J. P. Aurrand-Lions, E. Akaiwa, T. Yamakawa,

and J. R. Monties, "Biomedical applications of fuzzy logic controllers", In: *Intl. Conference on Fuzzy Logic and Neural Networks*, Iizuka, Fukuoka, Japan, 1990, pp. 553–556.

[44] M. Togai and S. Chiu, "A fuzzy accelerator and a programming environment for real-time fuzzy control", In: *Second IFSA Congress*, Tokyo, Japan, 1987, pp. 147–151.

[45] M. Togai and H. Watanabe, "Expert systems on a chip: an engine for real-time approximate reasoning", *IEEE Experts Systems Magazine*, 1986, No. 1, pp. 55–62.

[46] M. J. Tretter, "Interval analysis isn't fuzzy is it?", *Abstracts for an International Conference on Numerical Analysis with Automatic Result Verification: Mathematics, Application and Software, February 25 – March 1, 1993,* Lafayette, LA, 1993, p. 104.

[47] H. M. Wadsworth, Jr. (editor), *Handbook of statistical methods for engineers and scientists*, McGraw-Hill Publishing Co., N.Y., 1990.

[48] H. Watanabe and W. Detloff, "Reconfigurable fuzzy logic processor: a full custom digital VLSI", In: *Intl. Workshop on Fuzzy Systems Applications*, Iizuka, Japan, 1988, pp. 49–50.

[49] T. Yamakawa, "Fuzzy microprocessors – rule chip and defuzzifier chip", In: *Intl. Workshop on Fuzzy Systems Applications*, Iizuka, Japan, 1988, pp. 51–52.

[50] T. Yamakawa, "Intrinsic fuzzy electronic circuits for sixth generation computer", In: M. M. Gupta and T. Yamakawa (eds.), *Fuzzy Computing*, Elsevier, 1988, pp. 157–181.

[51] H. Q. Yang, H. Yao, and J. D. Jones, "Calculating functions of fuzzy numbers", *Fuzzy Sets and Systems*, 1993, Vol. 55, pp. 273–283.

[52] Y. Yoshikawa, T. Deguchi, and T. Yamakawa, "Exclusive fuzzy hardware systems for the appraisal of orthodentic results", In: *Intl. Conference on Fuzzy Logic and Neural Networks*, Iizuka, Fukuoka, Japan, 1990, pp. 939–942.

[53] L. A. Zadeh, "Fuzzy sets", *Information and control*, 1965, Vol. 8, pp. 338–353.

[54] L. A. Zadeh, *Outline of a new approach to the analysis of complex systems and decision processes*, IEEE Transactions on Systems, Man and Cybernetics, 1973, Vol. 3, pp. 28–44.



The author has requested enhancement of the downloaded file. All in-text references underlined in blue are linked to publications on ResearchGate.