A New Cauchy-Based Black-Box Technique for Uncertainty in Risk Analysis

Vladik Kreinovich[1] and Scott A. Ferson[2]

[1]Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, TX 79968, USA, vladik@cs.utep.edu
[2]Applied Biomathematics, 100 North Country Road
Setauket, NY 11733, USA, scott@ramas.com

***Abstract*— Uncertainty is very important in risk analysis. A natural way to describe this uncertainty is to describe a set of possible values of each unknown quantity (this set is usually an interval), plus any additional information that we may have about the probability of different values within this set. Traditional statistical techniques deal with the situations in which we have a complete information about the probabilities; in real life, however, we often have only partial information about them. We therefore need to describe methods of handling such partial information in risk analysis. Several such techniques have been presented, often on a heuristic basis. The main goal of this paper is to provide a justification for a general formalism for handling different types of uncertainty, and to describe a new black-box technique for processing this type of uncertainty.**

1. Introduction: uncertainty in risk analysis

Uncertainty in risk analysis: why. By definition, risk analysis deals with situations with uncertainty, i.e., with situations in which we do not have a complete and accurate knowledge about the state of the system. It is therefore very important that we be able to represent uncertainty in risk analysis as adequately as possible.

First component of uncertainty description: interval (set) uncertainty. In order to fully describe a system, we must know the exact values of all the physical quantities characterizing this system. For example, in environmental problems related to chemical pollution, a polluted system (e.g., a lake) can be fully described if we know the exact concentration of different pollutants in different parts of the lake.

Thus, to describe the uncertainty of our knowledge about a system, we must describe the uncertainty with which we know the values of each of the quantities (parameters) describing the system. Uncertainty means that we do not know the exact value of the quantity, several different values may be possible. For example, we may not know the exact value of the concentration but we may know that this concentration is between, say, 10^{-5} and 10^{-3}. In this case, any value from the interval $[10^{-5}, 10^{-3}]$ is possible; see, e.g., [12, 13].

An important risk-related situation that leads to intervals is when a measurement does not detect any presence of a certain substance because its concentration x is below the detection limit D. In this case, the only information we have about x is that x belongs to the interval $[0, D]$.

In general, we usually known an interval $\mathbf{x}$ of possible values of the unknown quantity x – or, sometimes, a more general set X of possible values of x (different from an interval, e.g., the union of two intervals).

Second component of uncertainty description: probabilistic uncertainty. The set X of possible values describes which values of the analyzed quantity are possible and which values are not. In addition to this information, we often know which values are more probable and which are less probable. In other words, we often have some information about the probability of different values x from the interval (set) $\mathbf{x}$ of possible values.

Probabilistic uncertainty: traditional techniques. In some cases, we know the exact expression for this distribution. In these cases, we can use standard statistical techniques to represent, elicit, and aggregate uncertainty. A survey of the corresponding techniques as applied to risk analysis is given, e.g., in [2].

The need for techniques corresponding to partial information about probabilities. In many other real-life situations, however, we have only *partial* information about the probabilities. To handle such situations, it is necessary to expand known statistical techniques of representing, eliciting, and aggregating uncertainty to problems in which we only have partial information about the probabilities.

What we are planning to do. The main objective of this paper is to provide a justification for a general formalism for handling different types of uncertainty, and to describe a new black-box technique for processing this type of uncertainty.

For a survey with a detailed description of our approach see [8]; see also [4, 6, 7, 21, 18, 25].

2. What is a natural way of representing partial information about probabilities?

Which representation of probability distribution should we choose? In probability theory, there are many different ways of representing a probability distribution. For example, one can use a probability density function (pdf), or a cumulative distribution function (CDF), or a probability measure, i.e., a function which maps different sets into a probability that the corresponding random variable belongs to this set. The reason why there are many different representations is that in different problems, different representations turned out to be the most useful.

We would like to select a representation which is the most useful for problems related to risk analysis. To make this selection, we must recall where the information about probabilities provided by risk analysis is normally used.

How is the partial information about probabilities used in risk analysis? The main objective of risk analysis is to make decisions. A standard way of making a decision is to select the action a for which the expected utility (gain) is the largest possible. This is where probabilities are used: in computing, for every possible action a, the corresponding expected utility. To be more precise, we usually know, for each action a and for each actual value of the (unknown) quantity x, the corresponding value of the utility $u_a(x)$. We must use the probability distribution for x to compute the expected value $E[u_a(x)]$ of this utility.

In view of this application, the most useful characteristics of a probability distribution would be the ones which would enable us to compute the expected value $E[u_a(x)]$ of different functions $u_a(x)$.

Which representations are the most useful for this intended usage? General idea. Which characteristics of a probability distribution are the most useful for computing mathematical expectations of different functions $u_a(x)$? The answer to this question depends on the type of the function, i.e., on how the utility value u depends on the value x of the analyzed parameter.

Smooth utility functions naturally lead to moments. One natural case is when the utility function $u_a(x)$ is smooth. We have already mentioned, in Section I, that we usually know a (reasonably narrow) interval of possible values of x. So, to compute the expected value of $u_a(x)$, all we need to know is how the function $u_a(x)$ behaves on this narrow interval. Because the function is smooth, we can expand it into Taylor series. Because the interval is narrow, we can safely consider only linear and quadratic terms in this expansion and ignore higher-order terms:

$$u_a(x) \approx c_0 + c_1 \cdot (x - x_0) + c_2 \cdot (x - x_0)^2,$$

where x_0 is a point inside the interval. Thus, we can approximate the expectation of this function by the expectation of the corresponding quadratic expression:

$$E[u_a(x)] \approx E[c_0 + c_1 \cdot (x - x_0) + c_2 \cdot (x - x_0)^2],$$

i.e., by the following expression:

$$E[u_a(x)] \approx c_0 + c_1 \cdot E[x - x_0] + c_2 \cdot E[(x - x_0)^2].$$

So, to compute the expectations of such utility functions, it is sufficient to know the first and second moments of the probability distribution.

In particular, if we use, as the point x_0, the average $E[x]$, the second moment turns into the variance of the original probability distribution. So, instead of the first and the second moments, we can use the mean E and the variance V.

From numerical moments to interval-valued moments. When we know the exact probability distribution, we must use the *exact* values of the first and the second moment (or mean and variance).

If we only have a *partial* information about the probability distribution, then we cannot compute the exact value of these moments; instead, we have *intervals* of possible values of these moments. So, from this viewpoint, a natural representation of the partial information about the probability distribution is given by intervals $\mathbf{E}$ and $\mathbf{V}$ of possible values of mean E and variance V.

In risk analysis, non-smooth utility functions are common. In engineering applications, most functions are smooth, so usually the Taylor expansion works pretty well. In risk analysis, however, not all dependencies are smooth. There is often a threshold x_0 after which, say, a concentration of a certain chemical becomes dangerous.

This threshold sometimes comes from the detailed chemical and/or physical analysis. In this case, when we increase the value of this parameter, we see the drastic increase in effect and hence, the drastic change in utility value. Sometimes, this threshold simply comes from regulations. In this case, when we increase the value of this parameter past the threshold, there is no drastic increase in effects, but there is a drastic decrease of utility due to the necessity to pay fines, change technology, etc. In both cases, we have a utility function which experiences an abrupt decrease at a certain threshold value x_0.

Non-smooth utility functions naturally lead to CDFs. We want to be able to compute the expected value $E[u_a(x)]$ of a function $u_a(x)$ which changes smoothly until a certain value x_0, then drops it value and continues smoothly for $x > x_0$. We usually know the (reasonably narrow) interval which contains all possible values of x. Because the interval is narrow and the dependence before and after the threshold is smooth, the resulting change in $u_a(x)$ before x_0 and after x_0 is much smaller than the change at x_0. Thus, with a reasonable accuracy, we can ignore the small changes before and after x_0, and assume that the function $u_a(x)$ is equal to a constant u^+ for $x < x_0$, and to some other constant $u^- < u^+$ for $x > x_0$.

The simplest case is when $u^+ = 1$ and $u^- = 0$. In this case, the desired expected value $E[u_a^{(0)}(x)]$ coincides with the probability that $x < x_0$, i.e., with the corresponding value $F(x_0)$ of the cumulative distribution function (CDF). A generic function $u_a(x)$ of this type, with arbitrary values u^- and u^+, can be easily reduced to this simplest case, because, as one can easily check, $u_a(x) = u^- + (u^+ - u^-) \cdot u^{(0)}(x)$ and hence, $E[u_a(x)] = u^- + (u^+ - u^-) \cdot F(x_0)$.

Thus, to be able to easily compute the expected values of all possible non-smooth utility functions, it is sufficient to know the values of the CDF $F(x_0)$ for all possible x_0.

From CDF to interval-valued CDF: the notion of a p-bound. When we know the exact probability distribution, we must use the exact values $F(x)$ of the CDF. If we only have a *partial* information about the probability distribution, then we cannot compute the exact values $F(x)$ of the CDF. Instead, for every x, we have an *interval* $[F^-(x), F^+(x)]$ of possible values of the probability $F(x)$. Such a pair of two CDFs $F^-(x)$ and $F^+(x)$ which bounds the (unknown) actual CDF is called a *probability bound*, or a *p-bound*, for short.

So, in risk analysis, a natural representation of the partial information about the probability distribution is given by a p-bound.

p-bounds or moments? We have shown that for decision problems with smooth utility functions, the best representation is by interval mean and interval variance, and for decision problems with discontinuous utility functions, the best representation of partial information is a p-bound.

Of the two corresponding representations of a probability distribution, CDF is much more informative: if we know CDF, we can compute the moments, but if we only know the moments, we can have many different CDFs. Thus, because we want to make our representation as informative as possible, it makes sense to use CDFs and their interval analogues – p-bounds.

Real numbers, intervals, and probability distributions are particular cases of p-bounds It is worth mentioning that several other types of uncertainty can be viewed as particular cases of p-bounds.

For example, the case of complete certainty, when we know the exact value x_0 of the desired quantity, can be represented as a p-bound in which

$$F^-(x) = F^+(x) = \begin{cases} 0 & \text{if } x \le x_0, \\ 1 & \text{otherwise} \end{cases}$$

The case when our only information about x is that x belongs to the interval $[x^-, x^+]$ can be represented by the following p-bound:

$$F^-(x) = \begin{cases} 0 & \text{if } x \le x^+, \\ 1 & \text{otherwise} \end{cases}$$

$$F^+(x) = \begin{cases} 0 & \text{if } x \le x^-, \\ 1 & \text{otherwise} \end{cases}$$

Finally, a probability distribution with a CDF $F(x)$ can be represented as a p-bound with $F^-(x) = F^+(x) = F(x)$.

Information about moments can also be represented in terms of p-bounds; see, e.g., [6, 25]. For example, if we know the interval $[x^-, x^+]$ on which the distribution is located, and if we know its mean E, then we can conclude that $F(x) \in [F^-(x), F^+(x)]$, where, e.g.,

$$F^+(x) = \min\left(1, \frac{x^+ - E}{x^+ - x}\right).$$

p-bounds have been successfully used in practice. We have shown that in risk analysis, a natural way to represent risk-related partial information about probabilities is by using a second-order probability distribution, namely, a *p-bound* – a pair of CDFs $F^-(x)$ and $F^+(x)$ for which $F^-(x) \le F^+(x)$. In particular, a real number, an interval, and a probability distribution are all particular cases of p-bounds.

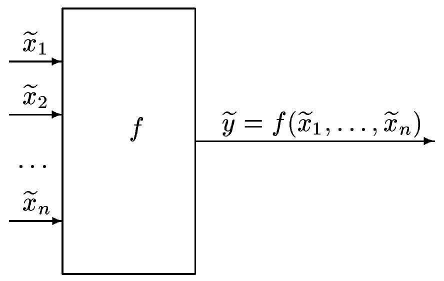
p-bounds have been successfully used in different risk analysis problems ranging from problems related to pollution and environment to risk analysis for nuclear engineering; see, e.g., [6, 25].

3. Error estimation for indirect measurements – formulation of the problem

What are indirect measurements. In many real-life situations, it is difficult or even impossible to directly measure the quantity y in which we are interested. For example, it is, at present, practically impossible to directly measure a distance to a distant quasar, or the amount of oil in a given area. Since we cannot measure such quantities *directly*, we have to measure them *indirectly*: Namely, we measure some other quantities $x_1, \ldots, x_n$ which are related to y by a known dependence $y = f(x_1, \ldots, x_n)$, and then apply the known function f to the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring x_i.

For example, to determine the amount of oil y in a given area, we measure the results x_i of sending ultrasound signals between the two parallel wells, and then estimate y by solving the appropriate system of partial differential equations (in this example, $f(x_1, \ldots, x_n)$ is a program for solving this system).

In general, such a two-stage procedure (measurement followed by computations) is called an *indirect measurement*, and the value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ resulting from this two-stage procedure is called the *result* of indirect measurement.



Toy example. To make the exposition clearer, we will illustrate these notions on the following toy example: Suppose that we are interested in the voltage V, but we have no voltmeter at hand. One possibility of measuring V indirectly follows from Ohm's law: we can measure the current I and the resistance R, and compute V as $I \cdot R$. In this case, x_1 is the current, $x_2 = R$, and $f(x_1, x_2) = x_1 \cdot x_2$.

If the measured value $\widetilde{x}_1$ of the current is 1.0, and the measured value of the resistance is $\widetilde{x}_2 = 2.0$, then the result of the corresponding indirect measurement is $\widetilde{y} = 1.0 \cdot 2.0 = 2.0$.

Error estimation for indirect measurements: a real-life problem. Measurements are never 100% accurate; hence, the result $\widetilde{x}_i$ of each direct measurement is, in general, somewhat different from the actual value of the measured quantity. As a result of these measurement errors $\Delta x_i = \widetilde{x}_i - x_i$, the result $\widetilde{y} = f(\widetilde{x}_1, \dots, \widetilde{x}_n)$ of applying f to the measurement result will be, in general, different from the actual value $y = f(x_1, \dots, x_n)$ of the desired quantity.

For example, in our toy problem, the actual value of the current may be $x_1 = 0.9 \neq \widetilde{x}_1 = 1.0$, and the actual value of the resistance is $x_2 = 2.05 \neq \widetilde{x}_2 = 2.0$. In this case, the actual value of the voltage is $y = x_1 \cdot x_2 = 0.9 \cdot 2.05 = 1.845 \neq 2.0$.

Since the result $\widetilde{y}$ of indirect measurement is, in general, different from the actual value y, it is desirable to know the characteristics of the error $\Delta y = \widetilde{y} - y$ of indirect measurement. How can we estimate these characteristics?

Possible information available for estimating the error of indirect measurements. First, we know the function $f(x_1, \dots, x_n)$. This function may be given as an analytical expression, or, more frequently, as an algorithm. It may be a program written in a high-level programming language (i.e., a *source code*), which can be translated into an executable file ready for computations, or it may be only an executable file, with no source code provided.

Second, we know the results $\widetilde{x}_1, \dots, \widetilde{x}_n$ of direct measurements.

Finally, we need some information about the errors of the direct measurements. The errors Δx_i come from the imperfection of the corresponding measuring instruments. For an instrument to be called *measuring*, its manufacturer must supply some (well-defined) information about the measurement errors. Ideally, this information must include the probability distribution of different measurement errors.

The knowledge of these probabilities is desirable but not always required and not always possible. In many practical cases, we only know the upper bounds Δ_i for the possible measurement errors, i.e., we only know that $|\Delta x_i| \leq \Delta_i$. In such cases, after each direct measurement, the only information that have about the actual value x_i of the measured quantity is that this value belongs to the *interval* $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

For example, in our toy case, the manufacturer of the measuring instruments may guarantee that the measurement error Δx_1 of measuring current cannot exceed $\Delta_1 = 0.1$, and the measurement error Δx_2 of measuring resistance cannot exceed $\Delta_2 = 0.05$. If no other information about the measurement accuracy is given, then, after we got the measurement results $\widetilde{x}_1 = 1.0$ and $\widetilde{x}_2 = 2.0$, the only information we have about the actual value of the current x_1 is that $x_1 \in [1.0 - 0.1, 1.0 + 0.1] = [0.9, 1.1]$. Similarly, the only information we have about the actual value of the resistance x_2 is that $x_2 \in [2.0 - 0.05, 2.0 + 0.05] = [1.95, 2.05]$. (The actual values $x_1 = 0.9$ and $x_2 = 1.05$, of course, belong to these intervals; if they did not, this would mean that the manufacturer's bounds are incorrect.)

In the situations when we only know the upper bounds on the measurement errors, the problem of estimating the error of indirect measurement is called the problem of *interval computations*; for details and examples of practical applications, see, e.g., [12, 13]. The setting when we only know intervals will be one of the settings considered in this paper.

Another setting which we will consider is a setting described in standard engineering textbooks on measurement (see, e.g., [9, 24]; see also [3, 11]). In this setting, the measurement error Δx_i of each direct measurement is normally distributed with 0 average and known standard deviation σ_i, and measurement errors of different direct measurements are independent random variables.

4. How engineers solve these problems?

In this paper, we will only consider situations when the measurements are reasonably accurate. In this paper, we will only consider situations in which the direct measurements are accurate enough, so that the resulting measurement errors Δx_i are small, and terms which are quadratic (or of higher order) in Δx_i can be safely neglected, and so, the dependence of the desired value $y = f(x_1, \dots, x_n) = f(\widetilde{x}_1 - \Delta x_1, \dots, \widetilde{x}_n - \Delta x_n)$ on Δx_i can be safely assumed to be linear.

In our toy example, $f(x_1, x_2) = x_1 \cdot x_2$, so

$$y = f(\widetilde{x}_1 - \Delta x_1, \widetilde{x}_2 - \Delta x_2) =$$

$$\widetilde{x}_1 \cdot \widetilde{x}_2 - \widetilde{x}_2 \cdot \Delta x_1 - \widetilde{x}_1 \cdot \Delta x_2 + \Delta x_1 \cdot \Delta x_2.$$

In our case, $\widetilde{x}_1 = 1.0$, $\widetilde{x}_2 = 2.0$, so $y = 2.0 - 2\Delta x_1 - \Delta x_2 + \Delta x_1 \cdot \Delta x_2$. The only non-linear term in this expansion is the quadratic term $\Delta x_1 \cdot \Delta x_2$.
Here, $\Delta x_1 = \widetilde{x}_1 - x_1 = 1.0 - 0.9 = 0.1$, $\Delta x_2 = 2.0 - 2.05 = -0.05$, and $\Delta y = 2.0 - 1.845 = 0.155$. If we ignore the quadratic term, we get approximate values $y_{\text{approx}} = 2.0 - 2\Delta x_1 - \Delta x_2 = 1.85$ and $\Delta y_{\text{approx}} = \widetilde{y} - y_{\text{approx}} = 0.15$. The error of this linear approximation is $0.005 \ll 0.15$.

Comments.

- To avoid possible confusion, we must emphasize the following. We are *not* talking about functions f which are linear for *all* possible values of input data. In this paper, we are considering data processing functions f which can be approximated by linear ones in the close vicinity of every measurement result $\vec{\widetilde{x}} = (\widetilde{x}_1, \ldots, \widetilde{x}_n)$. These linear approximations, however, are *different* for different measurement results.
- There are practical situations when the accuracy of the direct measurements is not high enough, and hence, quadratic terms cannot be safely neglected (see, e.g., [13] and references therein). In this case, the problem of error estimation for indirect measurements becomes computationally difficult (NP-hard) even when the function $f(x_1, \ldots, x_n)$ is quadratic [17, 27]. However, in most real-life situations, the possibility to ignore quadratic terms is a reasonable assumption, because, e.g., for an error of 1% its square is a negligible 0.01%.

With the above restriction in place, we can easily deduce the explicit expression for the error Δy of indirect measurement.

Indirect measurement error: derivation and the resulting formula. Due to the accuracy requirement, we can simplify the expression for $\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n)$ if we expand the function f in Taylor series around the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and restrict ourselves only to linear terms in this expansion. As a result, we get the expression

$$\Delta y = \vec{c} \cdot \vec{\widetilde{x}} \stackrel{\text{def}}{=} c_1 \cdot \Delta x_1 + \ldots + c_n \cdot \Delta x_n, \quad (1)$$

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

$$c_i = \frac{\partial f}{\partial x_i}_{|(\widetilde{x}_1, \ldots, \widetilde{x}_n)}. \quad (2)$$

Probability distribution of the indirect measurement error: derivation and the resulting formula. In the statistical setting, the desired measurement error Δy is a linear combination of independent Gaussian variables Δx_i, and hence, Δy is also normally distributed, with 0 average and the standard deviation

$$\sigma = \sqrt{c_1^2 \cdot \sigma_1^2 + \ldots + c_n^2 \cdot \sigma_n^2}. \quad (3)$$

Comment. A similar formula holds if we *do not* assume that Δx_i are normally distributed: it is sufficient to assume that they are independent variables with 0 average and known standard deviations σ_i.

Interval of possible values of the indirect measurement error: derivation and the resulting formula. In the interval setting, we do not know the probability of different errors Δx_i; instead, we only know that $|\Delta x_i| \le \Delta_i$. In this case, the sum (1) attains its largest possible value if each term $c_i \cdot \Delta x_i$ in this sum attains the largest possible value:

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

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

$$\Delta = |c_1| \cdot \Delta_1 + \ldots + |c_n| \cdot \Delta_n. \quad (4)$$

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

Hence, the interval of possible values of Δy is $[-\Delta, \Delta]$, with Δ defined by the formula (4).

Comment. In our toy problem, it is easy to compute the actual interval of possible values of $y = x_1 \cdot x_2$ when $x_1 \in [0.9, 1.1]$ and $x_2 \in [1.95, 2.05]$: Indeed, the function $f(x_1, x_2) = x_1 \cdot x_2$ is monotonically increasing as a function of each of its variables (for $x_1 > 0$ and $x_2 > 0$). Thus, the largest possible value of $y = f(x_1, x_2)$ is attained when both input variables take their largest possible values, i.e., when $x_1 = 1.1$ and $x_2 = 2.05$, and is equal to $1.1 \cdot 2.05 = 2.255$. Similarly, the smallest possible value of $y = f(x_1, x_2)$ is attained when both input variables take their smallest possible values, i.e., when $x_1 = 0.9$ and $x_2 = 1.95$; this smallest value of y is equal to $0.9 \cdot 1.95 = 1.755$. So, the interval of possible values of y is equal to $[1.755, 2.255]$. Hence, the interval of possible values for $\Delta y = \widetilde{y} - y = 2 - y$ is $[-0.255, 0.245]$.

On the other hand, applying formula (4), we get $\Delta = 2.0 \cdot 0.1 + 1.0 \cdot 0.05 = 0.25$ and the interval $[-0.25, 0.25]$. (We can see that this is indeed a good approximation to the actual interval.)

Error estimation for indirect measurement: a precise computational formulation of the problem. As a result of the above analysis, we get the following explicit formulation of the problem: given a function $f(x_1, \ldots, x_n)$, n numbers $\widetilde{x}_1, \ldots, \widetilde{x}_n$, and n positive numbers $\sigma_1, \ldots, \sigma_n$ (or $\Delta_1, \ldots, \Delta_n$), compute the corresponding expression (3) or (4).

Let us describe how this problem is solved now.

Textbook case: the function f is given by its analytical expression. If the function f is given by its analytical expression, then we can simply explicitly differentiate it, and get an explicit expression for (3) and (4). This is the case which is typically analyzed in textbooks on measurement theory (see, e.g., [9, 24]).

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

When this algorithm is presented in one of the standard programming languages such as Fortran or C, we can apply one of the existing analytical differentiation tools (see, e.g., [1, 10]), and automatically produce a program which computes the partial derivatives c_i. These tools analyze the code and produce the differentiation code as they go.

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

In this paper, we will analyze such black-box situations, and describe the optimal algorithm for computing σ, and a new algorithm for computing Δ. Before we describe these algorithms, we must describe known black-box-oriented algorithms.

A straightforward method of solving this problem: numerical differentiation. The most straightforward algorithm for solving this problem is to compute the derivatives c_i one-by-one, and then use the corresponding formula (3) or (4) to compute the desired σ. To compute the i-th partial derivative, we change the i-th input x_i to $\widetilde{x}_i + h_i$ for some h_i, and leave other inputs unchanged, i.e., we take $\delta_i = h_i$ for this i and $\delta_j = 0$ for all $j \neq i$. Then, we estimate c_i as

$$c_i = \frac{1}{h_i} \cdot \left(f\left(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n\right) - \widetilde{y}\right).$$

This algorithm is called *numerical differentiation.*

We want the change h_i to be small (so that quadratic terms can be neglected); we already know that changes of the order σ_i are small. So, it is natural to take $h_i = \sigma_i$ (or, correspondingly, $h_i = \Delta_i$). In other words, to compute c_i, we use the following values: $\delta_1 = \ldots = \delta_{i-1} = 0$, $\delta_i = \sigma_i$ (or $\delta_i = \Delta_i$), $\delta_{i+1} = \ldots = \delta_n = 0$.

Problem: sometimes, numerical differentiation takes too long. If a function $f(x_1, \ldots, x_n)$ is simple and fast-to-compute (e.g., if it is given by an explicit analytical expression), then we do not need the black-box-oriented algorithms at all. We only need these algorithms when the program f is itself time-consuming (e.g., computing f may involve solving an inverse problem). In this case, applying the function f is the most time-consuming part of this algorithm. So, the total time T that it takes us to compute σ is (approximately) equal to the running time T_f for the program f multiplied by the number of times N_f that we call the program f.

For numerical differentiation, $N_f = n$ (we call f n times to compute n partial derivatives). Hence, if the program f takes a long time to compute, and n is huge, then the resulting time T may be too long. For example, if we are determining some parameters of an oil well from the geophysical measurements, we may get n in the thousands, and T_f in minutes. In this case, $T = T_f \cdot n$ may take several weeks. This may be OK for a single measurement, but too long if we want more on-line results.

Known solution for statistical setting: Monte-Carlo simulation. In statistical setting, it is known that a straightforward simulation (Monte-Carlo type) saves time drastically. In this algorithm, we use a computer-based random number generator to simulate the normally distributed error. A standard normal random number generator usually produces a normal distribution with 0 average and standard deviation 1. So, to simulate a distribution with a standard deviation σ_i, we multiply the result α_i of the standard random number generator by σ_i. In other words, we take $\delta_i = \sigma_i \cdot \alpha_i$.

As a result of N Monte-Carlo simulations, we get N values $c^{(1)} = \vec{c} \cdot \vec{\delta}^{(1)}, \ldots, c^{(N)} = \vec{c} \cdot \vec{\delta}^{(N)}$ which are normally distributed with the desired standard deviation σ. So, we can determine σ by using the standard statistical estimate

$$\sigma = \sqrt{\frac{1}{N-1} \cdot \sum_{k=1}^{N} \left(c^{(k)}\right)^2}.$$

The relative error of this estimate depends only on N (as $\approx 1/\sqrt{N}$), and not on the number of variables n. Therefore, the number of steps N_f needed to achieve a given accuracy does not depend on the number of variables at all.

The error of the above algorithm is asymptotically normally distributed, with a standard deviation $\sigma_e \sim \sigma/\sqrt{2N}$. Thus, if we use a "two sigma" bound, we conclude that with probability 95%, this algorithm leads to an estimate for σ which differs from the actual value of σ by $\leq 2\sigma_e = 2\sigma/\sqrt{2N}$.

This is an error with which we estimate the error of indirect measurement; we do not need too much accuracy in this estimation, because, e.g., in real life, we say that an error is $\pm 10\%$ or $\pm 20\%$, but *not* that the error is, say, $\pm 11.8\%$. Therefore, in estimating the error of indirect measurements, it is sufficient to estimate the characteristics of this error with a relative accuracy of, say, 20%.

For the above "two sigma" estimate, this means that we need to select the smallest N for which $2\sigma_e = 2\sigma/\sqrt{2N} \leq 0.2 \cdot \sigma$, i.e., to select $N_f = N = 50$.

In many practical situations, it is sufficient to have a standard deviation of 20% (i.e., to have a "two sigma" guarantee of 40%). In this case, we need only $N = 13$ calls to f.

On the other hand, if we want to guarantee 20% accuracy in 99.9% cases, which correspond to "three sigma", we must use N for which $3\sigma_e = 3 \cdot \sigma/\sqrt{2N} \leq 0.2 \cdot \sigma$, i.e., we must select $N_f = N = 113$, etc.

For $n \approx 10^3$, all these values of N_f are much smaller than $N_f = n$ required for numerical differentiation.

So, if we have to choose between the (deterministic) numerical differentiation and the randomized Monte-Carlo algorithm, we must select:

- a deterministic algorithm when the number of variables n satisfies the inequality $n \leq N_0$ (where $N_0 \approx 50$), and
- a randomized method if $n \geq N_0$.

Additional advantage: parallelization. In Monte-Carlo algorithm, we need 50 calls to f. If each call requires a minute, the resulting time takes about an hour, which may be too long for on-line results. Fortunately, different calls to the function f are independent on each other, so we can run all the simulations in parallel.

The more processors we have, the less time the resulting computation will take. If we have as many processors as the required number of calls, then the time needed to estimate the error of indirect measurement becomes equal to the time of a single call, i.e., to the time necessary to compute the result $\widetilde{y}$ of this indirect measurement. Thus, if we have enough processors working in parallel, we can compute the result of the indirect measurement *and* estimate its error during the same time that it normally takes just to compute the result.

In particular, if the result $\widetilde{y}$ of indirect measurement can be computed in real time, we can estimate the error of this result in real time as well.

5. New method based on Cauchy distribution

Can we use a similar idea in the interval setting? Since Monte-Carlo simulation speeds up computations, it is desirable to use a similar technique in interval setting as well.

There is a problem here. In the interval setting, we do not know the exact distribution, we may have different probability distributions – as long as they are located within the corresponding intervals. If we only use one of these distributions for simulations, there is no guarantee that the results will be valid for other distributions as well.

In principle, we could repeat simulations for several different distributions, but this repetition would drastically increase the simulation time and thus, eliminate the advantages of simulation as opposed to numerical differentiation.

Yes, we can. Luckily, there is a mathematical trick that enables us to use Monte-Carlo simulation in interval setting as well. This trick is based on using *Cauchy distribution* – i.e., probability distributions with the probability density

$$\rho(z) = \frac{\Delta}{\pi \cdot (z^2 + \Delta^2)};$$

the value Δ is called the *scale parameter* of this distribution, or simply a *parameter*, for short.

Cauchy distribution has the following property that we will use: if $z_1, \ldots, z_n$ are independent random variables, and each of z_i is distributed according to the Cauchy law with parameter Δ_i, then their linear combination $z = c_1 \cdot z_1 + \ldots + c_n \cdot z_n$ is also distributed according to a Cauchy law, with a scale parameter $\Delta = |c_1| \cdot \Delta_1 + \ldots + |c_n| \cdot \Delta_n$.

Therefore, if we take random variables δ_i which are Cauchy distributed with parameters Δ_i, then the value

$$c = f(\widetilde{x}_1 + \delta_1, \ldots, \widetilde{x}_n + \delta_n) - f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$$
$$= c_1 \cdot \delta_1 + \ldots + c_n \cdot \delta_n$$

is Cauchy distributed with the desired parameter (4). So, repeating this experiment N times, we get N values $c^{(1)}, \ldots, c^{(N)}$ which are Cauchy distributed with the unknown parameter, and from them we can estimate Δ.

The bigger N, the better estimates we get.

There are two questions to be solved:

- how to simulate the Cauchy distribution;
- how to estimate the parameter Δ of this distribution from a finite sample.

Simulation can be based on the functional transformation of uniformly distributed sample values:

$$\delta_i = \Delta_i \cdot \tan(\pi \cdot (r_i - 0.5)),$$

where r_i is uniformly distributed on the interval $[0, 1]$.

In order to estimate σ, we can apply the Maximum Likelihood Method $\rho(d^1) \cdot \rho(d^2) \cdot \ldots \cdot \rho(d^n) \to \max$, where $\rho(z)$ is a Cauchy distribution density with the unknown Δ. When we substitute the above-given formula for $\rho(z)$ and equate the derivative of the product with respect to Δ to 0 (since it is a maximum), we get an equation

$$\frac{1}{1 + \left(\frac{c^{(1)}}{\Delta}\right)^2} + \ldots + \frac{1}{1 + \left(\frac{c^{(N)}}{\Delta}\right)^2} = \frac{N}{2}. \qquad (5)$$

The left-hand side of (5) is an increasing function that is equal to $0(< N/2)$ for $\Delta = 0$ and $> N/2$ for $\Delta = \max|c^{(k)}|$; therefore the solution to the equation (5) can be found by applying a bisection method to the interval $[0, \max|c^{(k)}|]$.

It is important to mention that we assumed that the function f is reasonably linear within the box

$$[\widetilde{x}_1 - \Delta_1, \widetilde{x}_1 + \Delta_1] \times \ldots \times [\widetilde{x}_n - \Delta_n, \widetilde{x}_n + \Delta_n].$$

However, the simulated values δ_i may be outside the box. When we get such values, we do not use the function f for them, we use a normalized function that is equal to f within the box, and that is extended linearly for all other values (we will see, in the description of an algorithm, how this is done).

As a result, we arrive at the following algorithm (first described in [15, 16, 19, 26]):

Algorithm.

```
• Apply $f$ to the results of direct measurements:
  $\widetilde{y} := f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$;
• For $k = 1, 2, \ldots, N$, repeat the following:
    • use the standard random number generator to compute $n$ numbers $r_i^{(k)}$, $i = 1, 2, \ldots, n$, that are uniformly distributed on the interval $[0, 1]$;
    • compute Cauchy distributed values
      $c_i^{(k)} := \tan(\pi \cdot (r_i^{(k)} - 0.5))$;
    • compute the largest value of $|c_i^{(k)}|$ so that we will be able to normalize the simulated measurement errors and apply $f$ to the values that are within the box of possible values: $K := \max_i |c_i^{(k)}|$;
    • compute the simulated measurement errors
      $\delta_i^{(k)} := \Delta_i \cdot c_i^{(k)}/K$;
    • compute the simulated measurement results
      $x_i^{(k)} := \widetilde{x}_i + \delta_i^{(k)}$;
    • apply the program $f$ to the simulated measurement results and compute the simulated error of the indirect measurement:
      $c^{(k)} := K \cdot \left(f\left(x_1^{(k)}, \ldots, x_n^{(k)}\right) - \widetilde{y}\right)$;
• Compute $\Delta$ by applying the bisection method to solve the equation (5).
```

Philosophical comment: sometimes, distortion of simulated phenomenon makes simulation more efficient. The use of Cauchy distribution in the above algorithm may seem somewhat counter-intuitive (see, e.g., [14, 22]). Indeed, in the interval setting, we do not know the exact probability distribution of each error Δ_i, but we do know that each error Δ_i belongs to the corresponding interval $[-\Delta_i, \Delta_i]$, so the actual (unknown) probability distribution for Δ_i must be located on this interval with probability 1. So, at first glance, if we want to design a simulation-type technique for computing Δ, we should use one of such possible distributions in our simulations. Instead, we use a Cauchy distribution for which the probability to be outside the interval $[-\Delta_i, \Delta_i]$ is non-zero. In other words, in order to make the simulations work, in these simulations, we *distort* the actual distributions.

At first glance, it may therefore seem natural to use, in our simulations, instead of n independent variables distributed according to Cauchy distribution, n independent variables δ_i distributed according to some distributions which are actually located on the interval $[-\Delta_i, \Delta_i]$. It is sufficient to select a distribution corresponding to $\Delta_i = 1$; let a and σ denote the average and standard deviation of this variable. Then, by scaling (namely, by multiplying by Δ_i), we can get a distribution corresponding to an arbitrary Δ_i. In this case, for each variable δ_i, its average is equal to $\Delta_i \cdot a$, and its standard deviation is equal to $\Delta_i \cdot \sigma$.

As a result of each simulation, we get the value $c_1 \cdot \delta_1 + \ldots + c_n \cdot \delta_n$. For large n, we can apply the limit theorem to this sum and conclude that this value is approximately normally distributed, with an average $\sum c_i \cdot \Delta_i$ and the standard deviation $\sqrt{\sum c_i^2 \cdot \Delta_i^2}$. The larger n, the closer this resulting distribution to normal. It is known that a normal distribution is uniquely determined by its first two moments; hence, for large n, the only information that we will be able to extract from the simulation results are the average and the standard deviation of the resulting distribution. From these two sums $\sum c_i \cdot \Delta_i$ and $\sum c_i^2 \cdot \Delta_i^2$, we cannot uniquely determine the desired value $\sum |c_i| \cdot \Delta_i$. Thus, we cannot use un-distorted simulation, and *distortion is inevitable.*

A general conclusion is: *In simulation, sometimes distorting the simulated process leads to a faster simulation-based algorithm.*

At a more general level, the advantages of simulations with distortions over accurate simulations may explain:

- why an artistic (somewhat geometrically distorted) portrait often captures our impression of a person much better than a (geometrically correct) photo, and
- why, in spite of humans' high optical abilities, we sometimes (as in optical illusions) distort the image that we are trying to reproduce.

When is this randomized algorithm better than deterministic numerical differentiation? To determine the parameter Δ, we use the maximum likelihood method. It is known that the error of this method is asymptotically normally distributed, with 0 average and standard deviation $1/\sqrt{N \cdot I}$, where I is Fisher's information:

$$I = \int_{-\infty}^{\infty} \frac{1}{\rho} \cdot \left(\frac{\partial \rho}{\partial \Delta}\right)^2 \, \mathrm{d}z.$$

For Cauchy probability density $\rho(z)$, we have $I =$

$1/(2\Delta^2)$, so the error of the above randomized algorithm is asymptotically normally distributed, with a standard deviation $\sigma_e \sim \Delta \cdot \sqrt{2/N}$. Thus, if we use a "two sigma" bound, we conclude that with probability 95%, this algorithm leads to an estimate for Δ which differs from the actual value of Δ by $\leq 2\sigma_e = 2\Delta \cdot \sqrt{2/N}$. So, if we want to achieve a 20% accuracy in the error estimation, we must use the smallest N for which $2\sigma_e = 2\Delta \cdot \sqrt{2/N} \leq 0.2 \cdot \Delta$, i.e., to select $N_f = N = 200$.

When it is sufficient to have a standard deviation of 20% (i.e., to have a "two sigma" guarantee of 40%), we need only $N = 50$ calls to f. For $n \approx 10^3$, both values N_f are much smaller than $N_f = n$ required for numerical differentiation.

So, if we have to choose between the (deterministic) numerical differentiation and the randomized Monte-Carlo algorithm, we must select:

- a deterministic algorithm when the number of variables n satisfies the inequality $n \leq N_0$ (where $N_0 \approx 200$), and
- a randomized algorithm if $n \geq N_0$.

Comment. If we use fewer than N_0 simulations, then we still get an approximate value of the range, but with worse accuracy – and the accuracy can be easily computed by using the above formulas.

This algorithm is naturally parallelizable. Similarly to the Monte-Carlo algorithm for statistical setting, we can run all N simulations in parallel and thus, speed up the computations.

6. Remark: the problem of non-linearity

Problem: In the above text, we assumed that the intervals $\mathbf{x}_i$ are narrow. In this case, terms quadratic in Δx_i are negligible, and so, we can safely assume that the desired function $f(x_1, \ldots, x_n)$ is linear on the box

$$\mathbf{x}_1 \times \ldots \times \mathbf{x}_n.$$

In practice, some intervals $\mathbf{x}_i$ may be wide, so even when restricted to the box, the function $f(x_1, \ldots, x_n)$ is nonlinear.

Solution. Usually, experts (e.g., designers of the corresponding technical system) know for which variables x_i, the dependence is non-linear. For each of these variables, we can *bisect* the corresponding interval $[\underline{x}_i, \overline{x}_i]$ into two smaller subintervals – for which the dependence is approximately linear. Then, we estimate the range of the function f separately on each of the resulting subboxes, and take the union of these two ranges as the range over the entire box.

If one bisection is not enough and the dependence of f on x_i is non-linear over one or several subboxes, we can bisect these boxes again, etc.

This bisection idea has been successfully used in interval computations; see, e.g., [13].

7. Testing our method on a variant of a challenge problem

The original challenge problem. The original challenge [23] included two problems: the simpler one, with only two variables, and a more sophisticated oscillator problem.

In the oscillator problem, we are interested in the parameter y that is connected with the mass m, spring constant k, damping coefficient c, and the frequency ω by a formula

$$y = \frac{k}{\sqrt{(k - m \cdot \omega^2)^2 + c^2 \cdot \omega^2}}.$$

Here, we have four input variable: $x_1 = m$, $x_2 = k$, $x_3 = c$, and $x_4 = \omega$.

Since we are interested in interval estimates, for each of these four variables x_i, we consider the interval of possible values. These intervals are: $\mathbf{x}_1 = [10, 12]$, $\mathbf{x}_2 = [60, 230]$, $\mathbf{x}_3 = [5, 25]$, and $\mathbf{x}_4 = [2.0, 3.5]$.

According to expert estimates, the most "non-linear" variable is the frequency $\omega = x_4$.

Why cannot we use the original challenge problem? The authors of [23] presented simplified problems with few variables, so that it would be easy to test relative advantages of different techniques before applying them to more realistic and more sophisticated problems.

For many uncertainty processing techniques, starting with such a simplified problem makes perfect sense. However, as we have mentioned earlier, the main advantage of Cauchy method is that it works faster when we have a large number of inputs – at least 50 or 200. For that many variables, Cauchy method has an advantage because in numerical differentiation, we need as many calls for the function f as there are variables (n), while in the Cauchy method, the number of calls N does not depend on the number of variables at all. So, when $n \gg N$, the use of Cauchy methods drastically decreases the running time – but when $n \ll N$, Cauchy method actually require longer time than numerical differentiation and therefore, it does not make sense to use it. Thus, it does not make sense to use Cauchy method in the original challenge problem. We do get the correct result – but after a lot of computations. This does not mean that Cauchy method is useless, because the challenge problem is an oversimplification of a real problem where the number of inputs is large.

So, to test the efficiency of Cauchy approach, we decided, instead of the original challenge problem, to use a more complex variant of this problem.

Multiple oscillator problem. Specifically, instead of a single oscillator, we decided to consider a multiple oscillator problem, in which, instead of a coefficient y of a single oscillator, we are interested in the sum of the values of this parameter corresponding to different oscillators. In precise terms, we have N_{osc} oscillators. Each oscillator i ($1 \leq i \leq N_{\text{osc}}$) is characterized by three parameters:

its mass m_i, its spring constant k_i, and its damping coefficient c_i. The same frequency ω is applied to all the oscillators. The resulting value y is equal to:

$$y = \sum_{i=1}^{N_{\rm osc}} \frac{k_i}{\sqrt{(k_i - m_i \cdot \omega^2)^2 + c_i^2 \cdot \omega^2}}.$$

So, we have a problem with $3N_{\rm osc} + 1$ inputs. In our simulation, we used $N_{\rm osc} = 400$ oscillators, with $3 \cdot 400 + 1 = 1,201$ inputs.

For each of the parameters k_i, m_i, and c_i, we selected the corresponding range by dividing the original range into $N_{\rm osc}$ equal subintervals. For example, we divide the original interval $[\underline{m}, \overline{m}] = [10, 12]$ for m into $N_{\rm osc}$ equal subintervals:

- the interval $\mathbf{m}_1$ of possible values of m_1 is
$$\left[\underline{m}, \underline{m} + \frac{1}{N_{\rm osc}} \cdot (\overline{m} - \underline{m})\right];$$
- the interval $\mathbf{m}_2$ of possible values of m_2 is
$$\left[\underline{m} + \frac{1}{N_{\rm osc}} \cdot (\overline{m} - \underline{m}), \underline{m} + \frac{2}{N_{\rm osc}} \cdot (\overline{m} - \underline{m})\right];$$
- ...
- the interval $\mathbf{m}_i$ of possible values of m_i is
$$\left[\underline{m} + \frac{i-1}{N_{\rm osc}} \cdot (\overline{m} - \underline{m}), \underline{m} + \frac{i}{N_{\rm osc}} \cdot (\overline{m} - \underline{m})\right];$$
- ...
- the interval $\mathbf{m}_{N_{\rm osc}}$ of possible values of $m_{N_{\rm osc}}$ is
$$\left[\overline{m} - \frac{1}{N_{\rm osc}} \cdot (\overline{m} - \underline{m}), \overline{m}\right];$$

For the frequency ω, we used the same interval $[2.0, 3.5]$ as in the original oscillator problem.

With that many variables, how can we check that our results are correct? To check that our results are correct, we must be able to compute the correct values. It turns out that it is possible to compute the smallest possible value $\underline{y}$ of y; it is much more difficult to compute $\overline{y}$. We therefore compared the actual value of $\underline{y}$ with the estimate $\widetilde{y} - \Delta$ generated by the Cauchy method.

For a fixed ω, the sum y attains the smallest possible value if and only if each of the terms in this sum is the smallest possible. It is easy to see that this expression decrease when c increases, so the smallest possible value of y is attained when each c_i attains its largest possible value $\overline{c}_i$.

With respect to m_i, each term is the smallest if and only if the expression $(k - m\omega^2)^2 + c^2 \cdot \omega^2$ is the largest. This expression is quadratic in terms of m and is increasing when $m \to -\infty$ and $m \to \infty$. Well known properties of a quadratic function enable us to conclude that that this maximum can be attained only at the endpoints of the corresponding interval. Thus, with respect to m_i, the minimum o i-th term is attained when either $m_i = \underline{m}_i$ or $m_i = \overline{m}_i$.

With respect to k_i, the i-th term is the smallest if and only if its square P/Q, where $P \overset{\rm def}{=} k^2$ and $Q \overset{\rm def}{=} (k - m \cdot \omega^2)^2 + c^2 \cdot \omega^2$ attains the smallest possible value. The local maxima and minima of P/Q can be determined if we equate the derivative of this expression to 0, i.e., if we solve the equation $P' \cdot Q = P \cdot Q'$. If we perform the differentiation, open the parentheses, and delete equal terms on both sides, we end up with a single value $k_{\rm extr} = m \cdot \omega^2 + c^2/m$. Is it local minimum or local maximum? For $k = 0$, we have $y = 0$; when $k \to \infty$, we have $y \to 1$. Since $y \ge 0$, we cannot have local minimum, so it is a local maximum. Thus, the minimum at each interval is attained at one of its endpoints.

So, to find the minimum of i-th terms, it is sufficient to consider four different combinations of m_i, k_i, and c_i: in all four combinations, $c_i = \overline{c}_i$, k_i is equal to either $\underline{k}_i$ to $\overline{k}_i$, and m_i is equal to either $\underline{m}_i$ to $\overline{m}_i$. The smallest of these values of the desired minimum of i-th term. The minimum of the entire sum is the sum of these $N_{\rm osc}$ minima.

Thus, we compute the minimum for a given ω. To compute the minimum over all possible ω, we repeat these computations for the frequencies $\underline{\omega}$, $\underline{\omega} + h$, $\underline{\omega} + 2h$, ..., $\overline{\omega}$ for some small step h.

We also compared the results of Cauchy method with the results of numerical differentiation.

Results of testing. Since the function is non-linear relative to ω, we divided ("bisected") the interval $[2.0, 3.5]$ of possible values of ω into two equal subintervals $[2.0, 2.75]$ and $[2.75, 3.5]$.

For the first subinterval $[2.0, 2.75]$, the actual value of $\widetilde{y} - \underline{y}$ is 161, the value obtained by Cauchy method is 184 – pretty close. Numerical differentiation leads to 151.

For the second subinterval $[2.75, 3.5]$, the actual value of $\widetilde{y} - \underline{y}$ is 54, the value obtained by Cauchy method is 36. This is too far away from the actual value, which means that the function is still non-linear over this subbox. Therefore, we bisected the interval $[2.75, 3.5]$ once again: into the third quarter $[2.75, 3.125]$ and the fourth quarter $[3.125, 3.5]$.

As we can see from the table, for both quarters, Cauchy method leads to reasonable results (not very good results are italicized)

Interval	actual value	num. diff.	Cauchy method
$[2.0, 2.75]$	161	151	184
$[2.75, 3.5]$	54	59	*36*
3rd quarter	23	*5*	16
4th quarter	37	39	42
# calls to f	$\gg 1200$	1200	200

Comments and conclusions.

- If we use $N < 200$ iterations, we still get an estimate – but more overestimating. In general, once get an estimate Δ from Cauchy method, we can then say that with probability 95%, the actual difference $\widetilde{y} - \underline{y}$ is bounded by the value

$$\Delta \cdot \left(1 + 2 \cdot \sqrt{\frac{2}{N}}\right).$$

  In particular:
  - for $N = 200$, we get 20% overestimation;
  - for $N = 50$, we get 40% overestimation.
- Cauchy method works well on this simulated example:
  - for $n \approx 1,200$ variables, we cut the number of calls to f ("gold-plated" calls) 6 (or 24) times (depending on whether we use $N = 200$ or $N = 50$);
  - for $n \approx 1,200,000$ variables, we cut the number of calls to f 6,000 (or 24,000) times.
  - in general, the number of calls to f is always 200 (or 50), no matter how many variables we have.

8. From intervals to more general case: preliminary results and future work

Combination of probabilistic and interval uncertainty. So far, we have considered two cases:

- *probabilistic* uncertainty, when the errors Δx_i of direct measurements are Gaussian distributed with 0 average and known standard deviation σ_i; in this case, we can use Monte-Carlo technique with Gaussian distribution;
- *interval* uncertainty, when the only information about Δx_i is that $|\Delta x_i| \le \Delta_i$; in this case, we can use Monte-Carlo techniques with Cauchy distribution.

What if we have *both* uncertainties?

Example: for a certain parameter x, we have a uniform distribution that is located on an interval $[a, b]$. However, we do not know the exact values of a and b; instead, we only know *intervals* $[\underline{a}, \overline{a}]$ and $[\underline{b}, \overline{b}]$ of possible values of a and b. In this case, x can be represented as

$$x = a + (b - a) \cdot \eta,$$

where η is uniformly distributed on the interval $[0, 1]$ (i.e., given with probabilistic uncertainty), while a and b are give with interval uncertainty.

If we have such an uncertainty for each x_i, then, to find the corresponding uncertainty in Δy, we can simulate the random and interval error components separately, and then combine the results; for details, see [26].

Cauchy methods for independent case. In the interval setting, we assumed that all possible combinations of values $x_i \in [\underline{x}_i, \overline{x}_i]$ are possible – in particular, that all possible combinations of extreme values are possible. In reality, often, extreme cases are not very probable.

For example, let us consider the case when the intervals for x_i are confidence intervals $[a_i - k \cdot \sigma_i, a_i + k \cdot \sigma_i]$. In this case, in addition to the variables x_i, we can apply a similar idea to their linear combinations and conclude that, say, the sum $x_1 + x_2$ can only lie within the corresponding interval $[a - k \cdot \sigma, a + k \cdot \sigma]$. One can show that as a result, instead of the original rectangular box, we have an ellipsoid – for which the combination of extreme values are indeed not possible. There exists a version of our Cauchy algorithm for the case when the input vector $(x_1, \ldots, x_n)$ can take any value within a given ellipsoid; this version is described in [26].

It is worth mentioning that there is an additional advantage of considering ellipsoids. Indeed, so far, we consider linear approximations for the function f. A natural idea is: why not get the next – quadratic – approximation? In other words, why not consider quadratic functions f? Alas, the problem of finding the range of a function f over the box is NP-hard, it needs (unless P=NP) about 2^n computations – which, for large n, is not practically possible.

Good news is that we can feasibly optimize a quadratic function $f(x_1, \ldots, x_n) = a_0 + \sum a_i \cdot x_i + \sum_{ij} a_{ij} \cdot x_i \cdot x_j$ over an ellipsoid $b_0 + \sum b_i \cdot x_i + \sum b_{ij} \cdot x_i \cdot x_j \le 1$. Indeed, e.g., the minimum of f is attained either inside the ellipsoid – where equating all partial derivatives to 0 leads to a easy-to-solve system of n linear equations with n unknown, or at the border, in which case the Lagrange multiplier method also leads to a simple system of linear equations; see [17] and references therein for details.

The problem is even simpler: we do not need to consider all possible values of the coefficients a_{ij} describing the "dependence" between x_i and x_j, it is sufficient to ask experts which pairs variables are more probable to be dependent on each other.

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCC5-209 and grant NCC2-1232, by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants numbers F49620-95-1-0518 and F49620-00-1-0365, by NSF grants CDA-9522207, EAR-0112968, EAR-0225670, and 9710940 Mexico/Conacyt, and by a research grant from Sandia National Laboratories as part of the Department of Energy Accelerated Strategic Computing Initiative (ASCI).

We want to thank Lev Ginzburg, Jan Beck, and all the participants of the Epistemic Uncertainty Workshop organized and sponsored by Sandia National Laboratories (August 6–7, 2002), especially William Oberkampf, Jon Helton, Steve Wojtkiewicz, and Cliff Joslyn, for valuable discussions.

References

[1] M. Berz, C. Bischof, G. Corliss, and A. Griewank, *Computational differentiation: techniques, applications, and tools*, SIAM, Philadelphia, 1996.

[2] R. T. Clemen and R. L. Winkler, "Combining probability distributions from experts in risk analysis", *Risk Analysis*, 1999, Vol. 19, No. 2, pp. 187–203.

[3] A. A. Clifford, *Multivariate error analysis*, J. Wiley & Sons, N.Y., 1973.

[4] J. A. Cooper, S. Ferson, and L. R. Ginzburg, "Hybrid processing of stochastic and subjective uncertainty data". Risk Analysis, 1996, Vol. 16, pp. 785–791.

[5] G. de Cooman, T. L. Fine, et al. *ISIPTA'01; 2nd Int'l Symposium on Imprecise Probabilities and Their Applications*, Cornell University, Ithaca, NY, 2001.

[6] S. Ferson, L. Ginzburg, and R. Akçakaya, "Whereof one cannot speak: when input distributions are unknown", *Risk analysis* (to appear).

[7] S. Ferson, L. Ginzburg, V. Kreinovich, and H. Schulte, "Interval Computations as a Particular Case of a General Scheme Involving Classes of Probability Distributions", In: J. Wolff von Gudenberg and W. Kraemer (eds.), *Scientific Computing, Validated Numerics, Interval Methods*, Kluwer, Dordrecht, 2001, pp. 355–366.

[8] S. Ferson and V. Kreinovich, *Representation, Elicitation, and Aggregation of Uncertainty in Risk Analysis – From Traditional Probabilistic Techniques to More General, More Realistic Approaches: A Survey*, Technical Report, Applied Biomathematics, 2001, 135 pp.

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

[10] A. Griewank, *Evaluating derivatives: Principles and techniques of algorithmic differentiation*, SIAM, Philadelphia, 2000.

[11] H. G. Hecht, *Mathematics in chemistry. An introduction to modern methods*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[12] Interval computations website http://www.cs.utep.edu/interval-comp

[13] R. B. Kearfott and V. Kreinovich, *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[14] V. Kreinovich, *In simulation modeling, sometimes simulation with distortions is useful*, Center for New Information Technology "Informatika", Leningrad, 1989 (in Russian).

[15] V. Kreinovich, A. Bernat, E. Villa, and Y. Mariscal, "Parallel computers estimate errors caused by imprecise data", *Interval Computations*, 1991, Vol. 2, pp. 21–46.

[16] V. Kreinovich, A. P. Bernat, E. Villa, and Y. Mariscal, "Parallel computers estimate errors caused by imprecise data", *Technical Papers of the the Society of Mexican American Engineers and Scientists 1992 National Symposium*, San Antonio, Texas, April 1992, pp. 192–199.

[17] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1998.

[18] V. Kreinovich, C. Langrand, and H. T. Nguyen, "Combining Fuzzy and Probabilistic Knowledge Using Belief Functions", *Proc. 2nd Vietnam-Japan Bilateral Symposium on Fuzzy Systems and Applications VJFUZZY'2001*, Hanoi, Vietnam, December 7–8, 2001, pp. 191–198.

[19] V. Kreinovich, M. I. Pavlovich, "Error estimate of the result of indirect measurements by using a calculational experiment", *Measurement Techniques*, 1985, Vol. 28, No. 3, pp. 201–205.

[20] V. Kreinovich, S. A. Starks, and R. Trejo, "Automatic Differentiation or Monte-Carlo Methods: Which is Better for Error Estimation?", *Abstracts of the SIAM Annual Meeting*, Toronto, July 13–17, 1998, p. 51.

[21] V. Kreinovich et al., "From Interval Methods of Representing Uncertainty To A General Description of Uncertainty", In: H. Mohanty and C. Baral (eds.), *Trends in Information Technology*, Tata McGraw-Hill, New Delhi, 2000, pp. 161–166.

[22] S. Nesterov and V. Kreinovich, "The worse, the better: a survey of paradoxical computational complexity of interval computations", In: M. A. Campos (ed.), *Abstracts of the II Workshop on Computer Arithmetic, Interval and Symbolic Computation (WAI'96)*, Recife, Pernambuco, Brazil, August 7-8, 1996, pp. 61A–63A.

[23] W. L. Oberkampf, J. C. Helton, C. A. Joslyn, S. F. Wojtkiewicz, and S. ferson, "Challenge problems: uncertainty in system reponse given uncertain parameters", *Reliability Engineering and System Safety* (this issue).

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

[25] Ramas website http://www.ramas.com

[26] R. Trejo and V. Kreinovich, "Error Estimations for Indirect Measurements: Randomized vs. Deterministic Algorithms For 'Black-Box' Programs", In: S. Rajasekaran, P. Pardalos, J. Reif, and J. Rolim (eds.), *Handbook on Randomized Computing*, Kluwer, 2001, pp. 673–729.

[27] S. A. Vavasis, *Nonlinear optimization: complexity issues*, Oxford University Press, N.Y., 1991.

[28] P. Walley, *Statistical reasoning with imprecise probabilities*, Chapman and Hall, N.Y., 1991.

Appendix 1: Numerical differentiation program

```
// program diffgen.c

// This program uses numerical differentiation to estimate
// errors of indirect measurements

#include <stdio.h>      // standard input and output functions
#include <malloc.h>
#include <math.h>
#define NumberOfVariables 4       //number of input variables

// Instead of this sample function - the sum of 4 variables -
// here we should place the code of the actual data processing function;
// its inputs should form an array inputs[i]
double F(double inputs[NumberOfVariables+1])
{
   int i;
   double temp =0.0;
   for (i = 1;i<= NumberOfVariables; i++)
      temp = temp + inputs[i];
return temp;
};

int main(void)
{
   double inputs[NumberOfVariables + 1]; // measurement results
   double deltaX[NumberOfVariables + 1]; // bounds on measurement errors
   double delta;  // desired error bound for indirect measurement

   double tildeY; // the result of indirect measurement, i.e.,
          // the result of applying F to measurement results
   int i,j;       // auxiliary variables used for loops:
   double xsimul[NumberOfVariables + 1]; // simulated measurement results

   double change[NumberOfVariables+1];   // simulated errors of
                                         // indirect measurement

   // inputting the results of direct measurements and the
   // corresponding measurement errors
   for (i = 1; i <= NumberOfVariables; i++){
      inputs[i] = 1.0;
      deltaX[i] = 0.1;
   };

   // printing the header
   printf("Numerical differentiation method for the toy example %\n");

   // computing the result of indirect measurements by
   // applying F to the measurement results
   tildeY = F(inputs);

   // return the result of indirect measurement
   printf("The result of indirect measurement is: %lf\n",tildeY);

   // calculating derivatives
   for (i = 1; i <= NumberOfVariables; i++){
      // simulating measurement error in i-th input only
      for (j = 1; j <= NumberOfVariables; j++){
 if (j == i)
            xsimul[j] = inputs[j] + deltaX[j];
         else
    xsimul[j] = inputs[j];
      }
      // computing corresponding error of indirect measurement
      change[i] = fabs(F(xsimul) - tildeY);
   }

   // computing the desired upper bound on the indirect measurement error
   // as the sum of components corresponding to different variables
   delta = 0.0;
   for (i = 1;i <= NumberOfVariables;i++)
      delta = delta + change[i];

   // return the desired delta
   printf("The maximum error is: %lf\n",delta);
return 0;
};
```

Appendix 2: Cauchy method

```
// program cauchygen.c

// This program uses Cauchy method to estimate
// errors of indirect measurements.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#define NumberOfVariables 4
#define N 200 //number of iterations
#define PI 3.14159265358979323846264338327950

// Instead of this sample function - the sum of 4 variables -
// here we should place the code of the actual data processing function;
// its inputs should form an array inputs[i]
double F(double inputs[NumberOfVariables+1])
{
   int i;
   double temp =0.0;
   for (i = 1;i<= NumberOfVariables; i++)
      temp = temp + inputs[i];
return temp;
};

int main(void)
{
   double inputs[NumberOfVariables + 1]; // measurement results
   double deltaX[NumberOfVariables + 1]; // bounds on measurement errors
   double deltaest;   // desired error bound for indirect measurement: estimate
   double deltabound; // desired error bound for indirect measurement: bound
```

```
  double tildeY; // the result of indirect measurement, i.e.,
         // the result of applying F to measurement results
  int i,k;       // auxiliary variables used for loops:
                 //   i is the number of a variable,
         //   k is the number of an iteration
  double r;      // uniform random variable on [0,1]]
  double cauchy[NumberOfVariables + 1]; // array of Cauchy distributed
                              // random variables with
                                      // parameter 1
  double largest;                     // normalization for
                              // Cauchy distribution
  double deltak[NumberOfVariables + 1]; // simulated measurement errors
  double xsimul[NumberOfVariables + 1]; // simulated measurement results
  double c[N + 1];    // simulated errors of indirect measurement
  double deltaminus;  // lower bound for the desired delta
  double deltaplus;   // upper bound for the desired delta
  double sum;         // auxiliary variable for the sum that is
                  // supposed to be equal to N/2 in the
              // equation for determining delta

  // seeding the random number generator
  srand( (unsigned)time( NULL ) );

  // inputting the result of direct measurements and the
  // corresponding measurement errors
  for (i = 1; i <= NumberOfVariables; i++){
     inputs[i] = 1.0;
     deltaX[i] = 0.1;
  };

  // printing the header
  printf("Numerical differentiation method for the toy example %\n");

  // computing the result of indirect measurements by
  // applying F to the measurement results
  tildeY = F(inputs);

  // return the result of indirect measurement
  printf("The result of indirect measurement is: %lf\n",tildeY);

  // generating N simulated results of indirect measurements
  // by using Cauchy distribution
  for (k = 1; k <= N; k++)
  {
     // simulating Cauchy distributed measurement errors
     for (i = 1; i <= NumberOfVariables; i++)
     {
// simulated uniform distribution on [0,1]
r = (double) rand() / (double) RAND_MAX;
        // transforming uniform distribution into Cauchy
        cauchy[i] = tan(PI * (r - 0.5));
     };

     // computing the largest value of |cauchy[i]| so that
     // we will be able to normalize simulated measurement
     // errors and apply F to the values that are within
     // the box of possible values
     largest = 0.0;
     for (i = 1; i <= NumberOfVariables; i++)
if (largest < fabs(cauchy[i]))
           largest = fabs(cauchy[i]);

     // computing simulated measurement errors
     for (i = 1; i <= NumberOfVariables; i++)
deltak[i] = deltaX[i] * cauchy[i] / largest;

     // computing simulated measurement results
     for (i = 1; i <= NumberOfVariables; i++)
xsimul[i] = inputs[i] + deltak[i];

     // computing simulated error of indirect measurement
     c[k] = fabs(largest * (F(xsimul) - tildeY));
  };

  // lower bound for delta is set at 0
  deltaminus = 0.0;

  // upper bound for delta is set at max of c[k]
  deltaplus = 0.0;
  for (k=1; k <= N; k++)
     if (deltaplus < c[k])
        deltaplus = c[k];

  // finding delta estimate by bisection; we stop when the lower bound
  // for delta is within 5% of the upper bound
  while (deltaminus < 0.95 * deltaplus)
  {
     // compute the midpoint of the interval [deltaminus,deltaplus]
     deltaest = (deltaminus + deltaplus) / 2;

     // compute the value of the sum at this midpoint
     sum = 0.0;
     for (k = 1; k <= N; k++)
        sum = sum + (deltaest * deltaest)/((deltaest * deltaest)
           + (c[k]) * (c[k]));

     // depending on whether this sum is > N/2 or < N/2 conclude
     // that delta belongs to the corresponding half-interval
     if (sum > (N / 2))
deltaplus = deltaest;
     else
deltaminus = deltaest;
   };

   // return the estimate for the desired delta
   printf("The estimate for delta is: %lf\n",deltaest);

   // compute the upper bound for delta
   deltabound = deltaest * (1 + 2 * sqrt(2.0 / N));
```

```
    // return the estimate for the desired delta
    printf("The 95 per cent bound for delta is: %lf\n",deltabound);
return 0;
};
```

Appendix 3: multiple oscillator function for the left half-interval

```
#define NumberOfVariables 1201

// single oscillator function
double osc(double m, double k, double c, double omega)
{
   double diff,temp;
   diff = k - m * omega * omega;
   temp = k / sqrt(diff * diff + c * c * omega * omega);
return temp;
};

// multiple oscillator function
double F(double inputs[NumberOfVariables + 1])
{
   double m,k,c,omega,sum;
   int i;
   omega = inputs[NumberOfVariables];
   sum = 0.0;
   for (i = 0; i < (NumberOfVariables / 3); i++){
      m = inputs[3 * i + 1];
      k = inputs[3 * i + 2];
      c = inputs[3 * i + 3];
      sum = sum + osc(m,k,c,omega);
   }
return sum;
}

   // inputting the result of direct measurements and the
   // corresponding measurement errors
   numbOsc = (NumberOfVariables - 1) / 3;
   for (i = 0; i < numbOsc; i++){
      inputs[3 * i + 1] = 10.0 + (2.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 1] = 1.0 / numbOsc;
      inputs[3 * i + 2] = 60.0 + (170.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 2] = 85.0 / numbOsc;
      inputs[3 * i + 3] = 5.0 + (20.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 3] = 10.0 / numbOsc;
   }
   inputs[NumberOfVariables] = 2.375;
   deltaX[NumberOfVariables] = 0.375;
```

Appendix 4: results of using Cauchy method for the left half-interval

```
Cauchy method for the multiple oscillator
for the first half of the frequency interval
The result of indirect measurement is: 766.658240
The estimate for delta is: 184.304499
The 95 per cent bound for delta is: 221.165399
```

Appendix 5: computing actual minimum of the multiple oscillator function for the left half-interval

```
// program actualmin1.c

// This program computes the actual minimum of the oscillator
// function for the case of multiple oxcillators
// for the left half of frequency interval
#include <stdio.h>      // standard input and output functions
#include <malloc.h>
#include <math.h>

#define NumberOfVariables  1201      //number of input variables
#define NumberOfOmega      20      //number of values of omega

// single oscillator function
double osc(double m, double k, double c, double omega)
{
   double diff,temp;
   diff = k - m * omega * omega;
   temp = k / sqrt(diff * diff + c * c * omega * omega);
return temp;
};

// computing minimum for each oscillator for given omega
double minosc(double m, double k, double c,
   double deltam, double deltak, double deltac, double omega)
{
   double ymin,ysimul;
   double simulm,simulk,simulc;
   int i1,i2;

   // as a value of c, we always take the upper endpoint
   simulc = c + deltac;

   ymin = osc(m,k,c,omega);

   // testing all endpoints and finding the smallest value
   for (i1 = 0; i1 <= 1; i1++){
      simulm = m + (2 * i1 - 1) * deltam;
      for (i2 = 0; i2 <= 1; i2++){
         simulk = k + (2 * i2 - 1) * deltak;
         ysimul = osc(simulm,simulk,simulc,omega);
         if (ysimul < ymin)
    ymin = ysimul;
      }
   };
return ymin;
};

int main(void)
{
   double inputs[NumberOfVariables + 1]; // measurement results
   double deltaX[NumberOfVariables + 1]; // bounds on measurement errors
   double ymin;  // desired minimum
   double m,k,c,deltam,deltak,deltac,omega;
   double ysimul; // minimum for each omega
   int i,j;       // auxiliary variables used for loops:
   int numbOsc;    // number of oscillators

   // inputting the result of direct measurements and the
   // corresponding measurement errors
   numbOsc = (NumberOfVariables - 1) / 3;
   for (i = 0; i < numbOsc; i++){
      inputs[3 * i + 1] = 10.0 + (2.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 1] = 1.0 / numbOsc;
      inputs[3 * i + 2] = 60.0 + (170.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 2] = 85.0 / numbOsc;
      inputs[3 * i + 3] = 5.0 + (20.0 / numbOsc) * (i + 0.5);
      deltaX[3 * i + 3] = 10.0 / numbOsc;
   }
   inputs[NumberOfVariables] = 2.375;
   deltaX[NumberOfVariables] = 0.375;

   // printing the header
   printf("Computing the actual minimum for the mutiple oscillator %\n");
   printf("for the first half of frequency interval %\n");

   // we first set ymin to some large value
   ymin = 1000000.0;

   // for each frequency, we compute the tital minimum as the
   // sum of minima of all oscillators
   for (j = 0; j < NumberOfOmega; j++){
      ysimul = 0.0;
      omega = inputs[NumberOfVariables] - deltaX[NumberOfVariables]
         + (2.0 * deltaX[NumberOfVariables] / NumberOfOmega) * (j + 0.5);
      for (i = 0; i < numbOsc; i++){
   m = inputs[3 * i + 1];
         k = inputs[3 * i + 2];
         c = inputs[3 * i + 3];
   deltam = deltaX[3 * i + 1];
   deltak = deltaX[3 * i + 2];
   deltac = deltaX[3 * i + 3];
         ysimul = ysimul + minosc(m,k,c,deltam,deltak,deltac,omega);
      }

      // if for some frequency, the minimal value ysimul is smaller
      // than minimum-so-far (ymin), then this new minimal value is
      // the new minimum-so-far
      if (ysimul < ymin)
  ymin = ysimul;
   };

   // printing the result
   printf("The minimal value is: %lf\n",ymin);
return 0;
};
```

```
Computing the actual minimum for the mutiple oscillator
```

```
for the first half of frequency interval
The minimal value is: 605.761832
```