COMBINING INTERVAL AND PROBABILISTIC UNCERTAINTY IN ENGINEERING APPLICATIONS

ANDREW POWNUK

Master's Program in Computational Science

APPROVED:

Vladik Kreinovich, Ph.D., Chair

Jack Chessa, Ph.D.

Aaron Velasco, Ph.D.

Piotr Wojciechowski, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

COMBINING INTERVAL AND PROBABILISTIC UNCERTAINTY IN ENGINEERING APPLICATIONS

by

ANDREW POWNUK

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Master's Program in Computational Science

THE UNIVERSITY OF TEXAS AT EL PASO

August 2016

Abstract

In many practical application, we process measurement results and expert estimates. Measurements and expert estimates are never absolutely accurate, their result are slightly different from the actual (unknown) values of the corresponding quantities. It is therefore desirable to analyze how this measurement and estimation inaccuracy affects the results of data processing.

There exist numerous methods for estimating the accuracy of the results of data processing under different models of measurement and estimation inaccuracies: probabilistic, interval, and fuzzy. To be useful in engineering applications, these methods should provide accurate estimate for the resulting uncertainty, should not take too much computation time, should be understandable to engineers, and should be sufficiently general to cover all kinds of uncertainty.

In this thesis, on several case studies, we show how we can achieve these four objectives. We show that we can get more accurate estimates by properly taking model inaccuracy into account. We show that we can speed up computations by processing different types of uncertainty differently. We show that we can make uncertainty-estimating algorithms more understandable by explaining the need for non-realistic Monte-Carlo simulations. We also analyze how general uncertainty-estimating algorithms can be.

Table of Contents

Chapter 1

Introduction

1.1 Need for Data Processing

One of the main objectives of science is to predict future values of physical quantities. For example:

- in meteorology, we need to predict future weather;
- in airplane control, we need to predict the location and the velocity of the plane under current control, etc.

To make these predictions, we need to know how each of these future values y depends on the current values $x_1, \ldots, x_n$ of the related quantities, i.e., we need to know an algorithm $y = f(x_1, \ldots, x_n)$ that relates y to x_i.

Once we find this information, we can then use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring the quantities x_i to compute an estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for the desired future value y. In these terms, the prediction consists of two stages:

- first, we measure or estimate the values of the quantities $x_1, \ldots, x_n$;
- then, we use the results $\widetilde{x}_i$ of measurement or estimation to compute an estimate $\widetilde{y}$ of the desired future value y as

$$\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n). \tag{1.1.1}$$

This computation is an important case of *data processing*.

For example, to predict tomorrow's temperature in El Paso y, we need to know today's temperature, wind speed and direction, and humidity in different locations inside El Paso and near El Paso; these values $x_1, \ldots, x_n$ are what we can use for this prediction. We can then use an appropriate method for solving the corresponding partial differential equation as the desired prediction algorithm $y = f(x_1, \ldots, x_n)$.

The weather example shows that the corresponding prediction algorithms can be very complicated; thus, we need to use high-performance computers for this data processing.

Other situations when we need data processing come from the fact that we also want to know the current state of the world, i.e., we want to know the current values of all the quantities that describe this state. Some of these quantities – like temperature in El Paso – we can measure directly. Other quantities, such as the temperature or the density deep inside the Earth, are difficult or even impossible to measure directly. To find the values of each such difficult-to-measure quantity y, a natural idea is to find related easier-to-measure quantities $x_1, \ldots, x_n$ that are related to the desired quantity y by a known dependence $y = f(x_1, \ldots, x_n)$, and then use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring x_i to compute an estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for y.

1.2 Need to Take Uncertainty Into Account When Processing Data

In general, data processing means applying some algorithm $f(x_1, \ldots, x_n)$ to the values of the quantities $x_1, \ldots, x_n$, resulting in a value $y = f(x_1, \ldots, x_n)$.

Values x_i usually come from measurements. Measurement are never absolutely accurate; the measurement result $\widetilde{x}_i$ is, in general, different from the actual (unknown) value x_i of the corresponding quantity: $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i \neq 0$; see, e.g., [25].

Because of the this, the computed value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ is, in general, different from the ideal value $y = f(x_1, \ldots, x_n)$.

It is therefore desirable to estimate the accuracy $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$. To estimate Δy, we need to have some information about the measurement errors Δx_i.

1.3 How to Gauge the Accuracy of the Estimates $\widetilde{x}_i$

In this thesis, we consider two types of estimates: measurements and expert estimates.

For *measurements*, we usually know the upper bound Δ_i on the absolute value of the measurement error $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$; see, e.g., [25]. This upper bound is usually provided by the manufacturer of the measurement instrument. The existence of such an upper bound comes from the very nature of measurement: if no upper bound is guaranteed, this means that whatever result the "measuring instrument" produces, the actual value can be any number from $-\infty$ to $+\infty$; this would be not a measurement result, it would be a wild guess.

Once we know the upper bound Δ_i for which $|\Delta x_i| \le \Delta_i$, and we know the measurement result $\widetilde{x}_i$, then we know that the actual value x_i is located in the interval $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

Different values x_i from these intervals can lead, in general, to different values of $y = f(x_1, \ldots, x_n)$. Our goal is then to find the range $\mathbf{y}$ of all possible values of y:

$$\mathbf{y} = \{f(x_1, \ldots, x_n) : x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}. \tag{1.3.1}$$

The problem of computing this range $\mathbf{y}$ is one of the main problems of *interval computations*; see, e.g., [7, 19, 25].

Often, in addition to the upper bound Δ_i on the measurement error, we also know the probabilities of different values $\Delta x_i \in [-\Delta_i, \Delta_i]$; see, e.g., [10, 11, 25].

To gauge the accuracy of (fuzzy) *estimates*, it is reasonable to use fuzzy techniques, techniques specifically designed to describe imprecise ("fuzzy") expert estimates in precise computer-understandable terms; see, e.g., [8, 21, 33]. In these techniques, the uncertainty of each estimate is described by a *membership function* $\mu_i(x_i)$ that describes, for all possible real numbers x_i, the degree to which the expert believes that this number is a possible value

of the corresponding quantity.

1.4 Measurement and Estimation Inaccuracies Are Usually Small

In many practical situations, the measurement and estimation inaccuracies Δx_i are relatively small, so that we can safely ignore terms which are quadratic (or of higher order) in terms of Δx_i [25]. We can use this fact to simplify the expression for the inaccuracy $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$.

Here, by definition of data processing, $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$. Thus,

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n). \tag{1.4.1}$$

From the definition of the measurement uncertainty Δx_i, we conclude that $x_i = \widetilde{x}_i - \Delta x_i$. Substituting this expression into the above formula (1.4.1) for Δy, we conclude that

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n). \tag{1.4.2}$$

Expanding the expression $f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n)$ in Taylor series in terms of small values Δx_i, and using the fact that terms quadratic in Δx_i can be safely ignored, we conclude that

$$f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_n - \Delta x_n) = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - \sum_{i=1}^{n} c_i \cdot \Delta x_i, \tag{1.4.3}$$

where we denoted

$$c_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}. \tag{1.4.4}$$

Substituting the expression (1.4.3) into the formula (1.4.2) and cancelling out the terms $+f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $-f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, we conclude that

$$\Delta y = \sum_{i=1}^{n} c_i \cdot \Delta x_i. \tag{1.4.5}$$

This is the main formula used to estimate the accuracy of estimating the desired quantity y.

1.5 How to Estimate Partial Derivatives c_i

Case of analytical differentiation. In some cases, we have explicit expressions – or efficient algorithms – for the partial derivatives (1.4.4).

Numerical differentiation: idea. In many practical situations, we do not have algorithms for computing the derivatives c_i. This happens, e.g., when we use proprietary software in our computations – in this case, we cannot use neither formula for differentiation, nor automatical differentiation tools.

In such situations, we can use the fact that we are under the linearization assumption that thus, that for each i and $h_i \neq 0$, we have

$$f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) \approx f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) + h_i \cdot c_i. \quad (1.5.1)$$

If we move the term $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$ to the left-hand side of the formula (1.5.1) and divide both sides of the resulting approximate equality by h_i, we conclude that

$$c_i \approx \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}}{h_i}. \quad (1.5.2)$$

This is a known formula for numerical differentiation.

Numerical differentiation: algorithm. We select some values $h_i \neq 0$. Then, we compute the values

$$c_i = \frac{f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}}{h_i}. \quad (1.5.3)$$

Numerical differentiation: computation time. The above algorithm contains $n + 1$ calls to the original data processing algorithm f:

- one call to compute $\widetilde{y}$ and
- n calls to compute n partial derivatives $c_1, \ldots, c_n$.

As we have mentioned earlier, the data processing algorithm f itself can be very time-consuming. The same weather prediction example shows that the number n of input variables can also be large, in hundreds or even thousands. As a result, the computation time needed for the numerical differentiation method can be very large.

1.6 Existing Methods for Computing the Probabilistic Uncertainty: Linearization Case

Analysis of the problem. In the probabilistic approach, it is assumed that we know the probability distributions of the measurement errors Δx_i, Usually, it is assumed that the measurement errors Δx_i are independent, and that each of them is normally distributed with 0 mean and known standard deviation σ_i.

Under these assumptions, the linear combination $\Delta y = \sum\limits_{i=1}^{n} c_i \cdot \Delta x_i$ is also normally distributed, with 0 mean and variance

$$\sigma^2 = \sum_{i=1}^{n} c_i^2 \cdot \sigma_i^2. \tag{1.6.1}$$

First method: numerical differentiation. In principle, we can estimate all the partial derivatives c_i and then apply the above formula.

Need for a faster method. As we have mentioned, the numerical differentiation method takes too long time. It is therefore desirable to have a faster method for computing σ.

Monte-Carlo method. Such a faster method is known: it is the method of Monte-Carlo simulations. In this method, to find the desired distribution for Δy, we several times $k = 1, \ldots, N$, do the following:

- we simulate n variables $\Delta x_i^{(k)}$ according to the corresponding probability distribution $\rho_i(\Delta x_i)$ (usually, normal);

- then we simulate $x_i^{(k)} = \widetilde{x}_i - \Delta x_i^{(k)}$ for each i;
- we apply the data processing algorithm $f(x_1, \ldots, x_n)$ to the simulated values, resulting in $y^{(k)} = f(x_1^{(k)}, \ldots, x_n^{(k)})$;
- finally, we compute $\Delta y^{(k)} = \widetilde{y} - y^{(k)}$.

One can easily check that these differences $\Delta y^{(k)}$ have the same distribution as Δy. So, we can determine the desired probability distribution from the sample $\Delta y^{(1)}$, ..., $\Delta y^{(N)}$.

1.7 Existing Methods for Computing the Interval Range: Linearization Case

Analysis of the problem. The expression (1.4.5) for Δy attains its largest value when each of the terms $c_i \cdot \Delta x_i$ attains its largest possible value.

Each of these terms is a linear function of Δx_i on the interval $[-\Delta_i, \Delta_i]$. When $c_i \geq 0$, this linear function is increasing and thus, it attains its largest possible value when Δx_i is the largest, i.e., when $\Delta x_i = \Delta_i$. The corresponding value of the term $c_i \cdot \Delta x_i$ is $c_i \cdot \Delta_i$.

When $c_i < 0$, the linear function $c_i \cdot \Delta x_i$ is decreasing and thus, it attains its largest possible value when Δx_i is the smallest, i.e., when $\Delta x_i = -\Delta_i$. The corresponding value of the term $c_i \cdot \Delta x_i$ is $c_i \cdot (-\Delta_i) = (-c_i) \cdot \Delta_i$.

In both cases, the largest possible value of each term $c_i \cdot \Delta x_i$ is equal to $|c_i| \cdot \Delta_i$. Thus, the largest possible value Δ of the sum (1.4.5) is equal to

$$\Delta = \sum_{i=1}^{n} |c_i| \cdot \Delta_i. \tag{1.7.1}$$

Similarly, one can show that the smallest possible value of the sum (1.4.5) is equal to $-\Delta$. Thus, the range of possible values of Δy is the interval $[-\Delta, \Delta]$, and the range $\mathbf{y}$ of possible values of $y = \widetilde{y} - \Delta y$ is equal to

$$\mathbf{y} = [\widetilde{y} - \Delta, \widetilde{y} + \Delta]. \tag{1.7.2}$$

Thus, once we have the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing, to compute the desired range $\mathbf{y}$, it is sufficient to be able to compute the value Δ.

First method: numerical differentiation. In principle, we can estimate all the partial derivatives c_i and then apply the above formula.

In particular, for $h_i = \Delta_i$, we get

$$\Delta = \sum_{i=1}^{n} |f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + \Delta_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) - \widetilde{y}|. \tag{1.7.3}$$

Need for a faster method. As we have mentioned, the numerical differentiation method takes too long time. It is therefore desirable to have a faster method for computing σ.

Monte-Carlo method. Such a faster method is indeed known; see, e.g., [13]. This method is based on using Cauchy distribution, with the probability density function

$$\rho_\Delta(x) = \frac{\Delta}{\pi} \cdot \frac{1}{1 + \dfrac{x^2}{\Delta^2}}. \tag{1.7.4}$$

Specifically, there is a known result about this distribution: that

- when we have several independent random variables Δx_i distributed according to Cauchy distribution with parameter Δ_i,
- then their linear combination $\sum\limits_{i=1}^{n} c_i \cdot \Delta x_i$ is also Cauchy distributed, with parameter $\Delta = \sum\limits_{i=1}^{n} |c_i| \cdot \Delta_i$.

This is exactly the desired formula (1.7.1). Thus, we can find Δ as follows:

- first, we several times simulate the inputs $\Delta x_i^{(k)}$ according to the Cauchy distribution;
- then, we plug in the corresponding simulated values $x_i^{(k)} = \widetilde{x}_i - \Delta x_i^{(k)}$ into the data processing algorithm $f(x_1, \ldots, x_n)$, producing the values $y^{(k)} = f(x_1^{(k)}, \ldots, x_n^{(k)})$;
- then, the differences $\Delta y^{(k)} = \widetilde{y} - y^{(k)}$ are also Cauchy distributed, with the desired parameter Δ.

The desired value Δ can then be determined, e.g., by using the Maximum Likelihood method, i.e., from the condition that

$$L \stackrel{\text{def}}{=} \prod_{k=1}^{N} \rho_\Delta(\Delta y^{(k)}) = \prod_{k=1}^{N} \frac{\Delta}{\pi} \cdot \frac{1}{1 + \dfrac{(\Delta y^{(k)})^2}{\Delta^2}} \to \max \tag{1.7.5}$$

Maximizing the likelihood L is equivalent to minimizing its negative logarithm $\psi \stackrel{\text{def}}{=} -\ln(L)$. Differentiating L with respect to Δ and equating the derivative to 0, we get the following formula:

$$\sum_{k=1}^{N} \frac{1}{1 + \dfrac{(\Delta y^{(k)})^2}{\Delta^2}} = \frac{N}{2}. \tag{1.7.6}$$

To find Δ from this equation, we can use, e.g., the bisection method. Thus, we arrive at the following algorithm.

Monte-Carlo method for estimating the interval uncertainty: algorithm. We select the number of iterations N. For each iteration $k = 1, \ldots, N$, we do the following:

- First, we simulate $\Delta x_i^{(k)}$ based on Cauchy with parameter Δ_i. We can do this, e.g., by computing $\Delta_i^{(k)} = \Delta_i \cdot \tan(\pi \cdot (r_{ik} - 0.5)$, where r_{ik} is the result of a standard random number generator that generates the numbers uniformly distributed on the interval $[0, 1]$.

- After that, we compute the difference

$$\Delta y^{(k)} \stackrel{\text{def}}{=} \widetilde{y} - f(\widetilde{x}_1 - \Delta x_1^{(k)}, \ldots, x_n - \Delta x_n^{(k)}). \tag{1.7.7}$$

Now, we can find Δ by using bisection to solve the equation (1.7.6). Specifically, we start with $\underline{\Delta} = 0$ and $\overline{\Delta} = \max\limits_{1 \le k \le N} |\Delta y^{(k)}|$. For $\Delta = \underline{\Delta}$, the left-hand side of the formula (**??**) is smaller than $N/2$, while for $\Delta = \underline{\Delta}$, this left-hand side is larger than $N/2$. Thus, if we want to get Δ with the desired accuracy ε, while $\overline{\Delta} - \underline{\Delta} > \varepsilon$, we do the following:

- we compute $\Delta_{\text{mid}} = \dfrac{\underline{\Delta} + \overline{\Delta}}{2}$;

- we check whether

$$\sum_{k=1}^{N} \frac{1}{1+\dfrac{\left(\Delta y^{(k)}\right)^2}{\Delta_{\rm mid}^2}} < \frac{N}{2}; \tag{1.7.8}$$

- if this inequality is true, we replace $\underline{\Delta}$ with the new value $\Delta_{\rm mid}$, leaving $\overline{\Delta}$ unchanged;
- if this inequality is not true, we replace $\overline{\Delta}$ with the new value $\Delta_{\rm mid}$, leaving $\underline{\Delta}$ unchanged.

In both cases, on each iteration, the width of the interval $\left[\underline{\Delta}, \overline{\Delta}\right]$ becomes twice smaller. Thus, in s steps, we decrease this width by a factor of 2^s. So, in a few steps, we get the desired value Δ. For example, to get the width $\le 0.1\%$ of the original one, it is sufficient to perform only 10 iterations of the bisection procedure.

Monte-Carlo method: computation time. In the Monte-Carlo approach, we need $N+1$ calls to the data processing algorithm f, where N is the number of simulations.

Good news is that, as in statistical methods in general, the needed number of simulation N is determined only by the desired accuracy ε and does not depend on the number of inputs n. For example, to find Δ with relative accuracy 20% and certainty 95% (i.e., in 95% of the cases), it is sufficient to take $n = 200$ [13].

Thus, when the number of inputs n of the data processing algorithm f is large, the Monte-Carlo method for estimating interval uncertainty is much faster than numerical differentiation.

For many practical problems, we can achieve an even faster speed-up. In both methods described in this section, we apply the original algorithm $f(x_1, \ldots, x_n)$ several times: first, to the tuple of nominal values $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and then, to several other tuples $(\widetilde{x}_1 + \eta_1, \ldots, \widetilde{x}_n + \eta_n)$ for some η_i. For example, in the numerical differentiation method, we apply the algorithm f to tuples $(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + h_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$ corresponding to $i = 1, \ldots, n$.

In many practical cases, once we have computed the value $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, we can then compute the values

$$f(\widetilde{x}_1 + \eta_1, \ldots, \widetilde{x}_n + \eta_n) \tag{1.7.9}$$

faster than by directly applying the algorithm f to the corresponding tuple. This happens, for example, if the algorithm for computing $f(x_1,\dots,x_n)$ solves a system of nonlinear equations $F_j(y_1,\dots,y_k,x_1,\dots,x_n)=0$, $1\le j\le k$, and then returns $y=y_1$.

In this case, once we know the values $\widetilde{y}_j$ for which $F_j(\widetilde{y}_1,\dots,\widetilde{y}_k,\widetilde{x}_1,\dots,\widetilde{x}_n)=0$, we can find the values $y_j=\widetilde{y}_j+\Delta y_j$ for which

$$F_j(\widetilde{y}_1+\Delta y_1,\dots,\widetilde{y}_k+\Delta y_k,\widetilde{x}_1+\eta_1,\dots,\widetilde{x}_n+\eta_n)=0 \tag{1.7.10}$$

by linearizing this system into an easy-to-solve system of linear equations

$$\sum_{j'=1}^{k} a_{jj'}\cdot\Delta y_{j'}+\sum_{i=1}^{n} b_{ji}\cdot\eta_i=0, \tag{1.7.11}$$

where $a_{jj'}\stackrel{\text{def}}{=}\dfrac{\partial F_j}{\partial y_{j'}}$ and $b_{ji}\stackrel{\text{def}}{=}\dfrac{\partial F_j}{\partial x_i}$.

A similar simplification is possible when the value y corresponding to given values $x_1,\dots,x_n$ comes from solving a system of nonlinear differential equations

$$\frac{dy_j}{dt}=f_j(y_1,\dots,y_k,x_1,\dots,x_n). \tag{1.7.12}$$

In this case, once we find the solution $\widetilde{y}_j(t)$ to the system of differential equations

$$\frac{d\widetilde{y}_j}{dt}=f_j(\widetilde{y}_1,\dots,\widetilde{y}_k,\widetilde{x}_1,\dots,\widetilde{x}_n) \tag{1.7.13}$$

corresponding to the nominal values, we do not need to explicitly solve the modified system of differential equations

$$\frac{dy_j}{dt}=f_j(y_1,\dots,y_k,\widetilde{x}_1+\eta_1,\dots,\widetilde{x}_n+\eta_n) \tag{1.7.14}$$

to find the corresponding solution. Instead, we can take into account that the differences η_i are small; thus, the resulting differences $\Delta y_j(t)\stackrel{\text{def}}{=}y_j(t)-\widetilde{y}_j(t)$ are small. So, we can linearize the resulting differential equations

$$\frac{\Delta y_j(t)}{dt}=f_j(\widetilde{y}_1+\Delta y_1,\dots,\widetilde{y}_k+\Delta y_k,\widetilde{x}_1+\eta_1,\dots,\widetilde{x}_n+\eta_n)-$$
$$f_j(\widetilde{y}_1,\dots,\widetilde{y}_k,\widetilde{x}_1,\dots,\widetilde{x}_n) \tag{1.7.15}$$

into easier-to-solve *linear* equations

$$\frac{d\Delta y_j}{dt}=\sum_{j'=1}^{k} a_{jj'}\cdot\Delta y_{j'}+\sum_{i=1}^{n} b_{ji}\cdot\eta_i, \tag{1.7.16}$$

where $a_{jj'} \stackrel{\text{def}}{=} \dfrac{\partial f_j}{\partial y_{j'}}$ and $b_{ji} \stackrel{\text{def}}{=} \dfrac{\partial f_j}{\partial x_i}$.

This idea – known as *local sensitivity analysis* – is successfully used in many practical applications; see, e.g., [4, 28].

Comment. In some practical situations, some of these deviations are not small. In such situations, we can no longer use linearization, we need to use global optimization techniques of *global sensitivity*; see, e.g., [4, 28].

1.8 Existing Methods for Estimating Fuzzy Uncertainty: Linearization Case

Analysis of the problem. Let us assume that we have fuzzy estimates $\mu_i(\Delta x_i)$ for all the estimation errors Δx_i. In this case, we want to estimate, for every real number Δy, the degree $\mu(\Delta y)$ to which this number is a possible value of data processing inaccuracy.

The value Δy is a possible value of inaccuracy if there exist values Δx_i

- which are possible as inaccuracies of input estimation, and
- for which $= f(x_1, \ldots, x_n)$.

For simplicity, let us use $\min(a, b)$ to describe "and", and $\max(a, b)$ to describe "or". Then, for each combination of values Δx_i, the degree to which all these values are possible is equal to the minimum of the degrees to which each of them is possible:

$$\min(\mu_1(\Delta x_1), \ldots \mu_n(\Delta x_n)), \tag{1.8.1}$$

and the desired degree Δy is equal to the maximum of these expressions over all possible combinations of Δx_i:

$$\mu(\Delta y) = \max \min(\mu_1(\Delta x_1), \ldots \mu_n(\Delta x_n)), \tag{1.8.2}$$

where the maximum is taken over all tuples for which $y = f(x_1, \ldots, x_n)$; this expression is known as *Zadeh's extension principle.*

It is know that the computation of this membership function can be simplified if instead of each membership function $\mu(z)$, we consider its α-*cuts*, i.e., sets ${}^{\alpha}z \stackrel{\text{def}}{=} \{z : \mu(z) \geq \alpha\}$ corresponding to different values $\alpha \in [0,1]$.

It should be mentioned that since $\mu(z)$ is always non-negative, the above definition (with non-strict inequality $\mu(z) \geq \alpha$) does not work for $\alpha = 0$: strictly speaking, all real numbers z satisfy the corresponding inequality. Thus, for $\alpha = 0$, the α-cut is defined slightly differently: as the closure of the set $\{z : \mu(z) > 0\}$ corresponding to strict inequality.

Usually, membership functions correspond to *fuzzy numbers*, i.e., all α-cuts are intervals. Moreover, the α-cuts corresponding to Δx_i are usually symmetric, i.e., have the form ${}^{\alpha}\Delta x_i = [-{}^{\alpha}\Delta_i, {}^{\alpha}\Delta_i]$ for appropriate values ${}^{\alpha}\Delta_i$.

One can easily check, based on the formula (1.8.2), that $\mu(\Delta) \geq \alpha$ if and only if there exists a tuple $(\Delta x_1, \ldots, \Delta x_n)$ for which

$$\min(\mu_1(\Delta x_1), \ldots \mu_n(\Delta x_n)) \geq \alpha, \tag{1.8.3}$$

i.e., equivalently, for which $\mu_i(\Delta_i) \geq \alpha$ for all i.

In other words, Δy belongs to the α-cut if and only if it is a possible value of the expression (1.4.5) when Δx_i belongs to the corresponding α-cut $[-{}^{\alpha}\Delta_i, {}^{\alpha}\Delta_i]$.

We already know how to compute the range of such values Δy. Thus, we arrive at the following algorithm for computing the desired α-cut $[-{}^{\alpha}\Delta, {}^{\alpha}\Delta]$.

How to estimate Δy: case of fuzzy uncertainty – resulting formula. In case of fuzzy uncertainty, for every $\alpha \in [0,1]$, we are given the α-cuts $[-{}^{\alpha}\Delta_i, {}^{\alpha}\Delta_i]$ describing the expert's uncertainty about the estimates $\widetilde{x}_i$ and about the model used in data processing.

Based on these α-cuts, we can compute the α-cuts $[-{}^{\alpha}\Delta, {}^{\alpha}\Delta]$ for Δy as follows:

$${}^{\alpha}\Delta = \sum_{i=1}^{n} |c_i| \cdot {}^{\alpha}\Delta_i. \tag{1.8.4}$$

Comment. In principle, there are infinitely many different values α in the interval $[0,1]$. However, we should take into account that the values α correspond to experts' degrees of confidence, and experts cannot describe their degrees with too much accuracy.

Usually, it is sufficient to consider only eleven values $\alpha = 0.0$, $\alpha = 0.1$, $\alpha = 0.2$, ..., $\alpha = 0.9$, and $\alpha = 1.0$. Thus, we need to apply the formula (1.8.4) eleven times.

This is in line with the fact that, as psychologists have found, we usually divide each quantity into 7 plus plus minus 2 categories – this is the largest number of categories whose meaning we can immediately grasp; see, e.g., [18, 27]. For some people, this "magical number" is $7 + 2 = 9$, for some it is $7 - 2 = 5$. So, to make sure that do not miss on some people's subtle divisions, it is sufficient to have at least 9 different categories. From this viewpoint, 11 categories are sufficient; usually the above eleven values are chosen since for us, it is easier to understand decimal numbers.

1.9 Open Problems and What We Do in This Thesis

What we want. In engineering applications, we want methods for estimating uncertainty which are:

- *accurate* – this is most important in most engineering applications;
- *fast*: this is important in some engineering applications where we need real-time computations,
- *understandable* to engineers – otherwise, engineers will be reluctant to use them, and
- sufficiently *general* – so that they can be applied in all kinds of situations.

It is therefore desirable to try to modify the existing methods for estimating uncertainty so that these methods will become more accurate, faster, more understandable, and/or more general.

This thesis contains the preliminary results of our research:

- In Chapter 2, we show how to make these methods more accurate.
- In Chapter 3, we show how to make these methods faster.
- In Chapter 4, we show how to make these methods more understandable to engineers.
- In Chapter 5, we analyze how to make these methods more general.

The final Chapter 6 contains remaining open problems and our plan for future work.

Chapter 2

How to Get More Accurate Estimates – by Properly Taking Model Inaccuracy into Account

2.1 What If We Take Into Account Model Inaccuracy

Models are rarely exact. Engineering systems are usually complex. As a result, it is rarely possible to find explicit expressions for y as a function of the parameters $x_1, \ldots, x_n$. Usually, we have some approximate computations. For example, if y is obtained by solving a system of partial differential equations, we use, e.g., the Finite Element method to find the approximate solution and thus, the approximate value of the quantity y.

How model inaccuracy is usually described. In most practical situations, at best, we know the upper bound ε on the accuracy of the computational model. In such cases, for each tuple of parameters $x_1, \ldots, x_n$, once we apply the computational model and get the value $F(x_1, \ldots, x_n)$, the actual (unknown) value $f(x_1, \ldots, x_n)$ of the quantity y satisfies the inequality

$$|F(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)| \le \varepsilon. \tag{2.1.1}$$

How this model inaccuracy affects the above checking algorithms: analysis of the problem. Let us start with the formula (1.7.3). This formula assumes that we know the exact values of $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $y_i \stackrel{\text{def}}{=} f(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + \Delta_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n)$. Instead, we know the values

$$\widetilde{Y} \stackrel{\text{def}}{=} F(\widetilde{x}_1, \ldots, \widetilde{x}_n) \tag{2.1.2}$$

and

$$Y_i \stackrel{\text{def}}{=} F(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + \Delta_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n) \tag{2.1.3}$$

which are ε-close to the values $\widetilde{y}$ and y_i. We can apply the formula (1.7.3) to these approximate values, and get the upper and lower bounds:

$$\overline{Y} = \widetilde{y} + \sum_{i=1}^{n} |Y_i - \widetilde{y}\,|\,; \quad \underline{Y} = \widetilde{y} - \sum_{i=1}^{n} |Y_i - \widetilde{y}\,|\,. \tag{2.1.4}$$

Here, $\left|\widetilde{Y} - \widetilde{y}\right| \leq \varepsilon$ and $|Y_i - y_i| \leq \varepsilon$, hence $\left|\left(Y_i - \widetilde{Y}\right) - (y_i - \widetilde{y}\,)\right| \leq 2\varepsilon$ and $\left|\left|Y_i - \widetilde{Y}\right| - |y_i - \widetilde{y}\,|\right| \leq 2\varepsilon$. By adding up all these inequalities, we conclude that

$$\left|\,\overline{y} - \overline{Y}\,\right| \leq (2n+1) \cdot \varepsilon \text{ and } \left|\,\underline{y} - \underline{Y}\,\right| \leq (2n+1) \cdot \varepsilon. \tag{2.1.5}$$

Thus, the only information that we have about the desired upper bound $\overline{y}$ is that $\overline{y} \leq B$, where

$$B \stackrel{\text{def}}{=} \overline{Y} + (2n+1) \cdot \varepsilon. \tag{2.1.6}$$

Hence, we arrive at the following method.

Resulting method. We know:

- an algorithm $F(x_1, \ldots, x_n)$ that, given the values of the parameters $x_1, \ldots, x_n$, computes the value of the quantity y with a known accuracy ε;
- for each parameter x_i, we know its nominal value $\widetilde{x}_i$ and the largest possible deviation Δ_i from this nominal value.

Based on this information, we need to find an enclosure for the range of all the values values $f(x_1, \ldots, x_n)$ for all possible combinations of values x_i from the corresponding intervals

$$[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i].$$

We can perform this computation as follows:

1) first, we apply the algorithm F to compute the value

$$\widetilde{Y} = F(\widetilde{x}_1, \ldots, \widetilde{x}_n); \tag{2.1.7}$$

2) then, for each i from 1 to n, we apply the algorithm F to compute the value

$$Y_i = F(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + \Delta_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n); \tag{2.1.8}$$

3) after that, we compute the interval $\left[\widetilde{Y} - \Delta, \widetilde{Y} + \Delta\right]$, where

$$\Delta = \sum_{i=1}^{n} \left| Y_i - \widetilde{Y} \right| + (2n+1) \cdot \varepsilon. \tag{2.1.9}$$

Comment 1. Please note that, in contrast to the case of the exact model, if, e.g., the upper bound $\widetilde{Y} + \Delta$ is larger than a given threshold t, this does not necessarily mean that the corresponding specification $y \le t$ is not satisfied: maybe it is satisfied, but we cannot check that since we only know approximate values of y.

Comment 2. Similar bounds can be found for the estimates based on the Cauchy distribution.

Comment 3. The above estimate B is not the best that we can get, but it has been proven that computing the best estimate would require un-realistic exponential time [9, 15] – i.e., time which grows as 2^s with the size s of the input; thus, when we only consider feasible algorithms, overestimation is inevitable.

Comment 4. Similar to the methods described in Chapter 1, instead of directly applying the algorithm F to the modified tuples, we can, wherever appropriate, to use the above-mentioned local sensitivity analysis technique.

Problem. When n is large, then, even for reasonably small inaccuracy ε, the value $(2n+1) \cdot \varepsilon$ is large.

In this chapter, we show how we can get better estimates for the difference between the desired bounds $\widetilde{y}$ and $\underline{y}$ and the computed bounds $\overline{Y}$ and $\underline{Y}$. The results from this chapter first appeared in [14].

2.2 How to get Better Estimates

Main idea. As we have mentioned earlier, usually, we know the partial differential equations that describe the engineering system. Model inaccuracy comes from the fact that we do not have

an analytical solution to this system of equations, so we have to use numerical (approximate) methods.

Usual numerical methods for solving systems of partial differential equations involve discretization of space – e.g., the use of Finite Element Methods.

Strictly speaking, the resulting inaccuracy is deterministic. However, in most cases, for all practical purposes, this inaccuracy can be viewed as random: when we select a different combination of parameters, we get an unrelated value of discretization-based inaccuracy.

In other words, we can view the differences

$$F(x_1, \ldots, x_n) - f(x_1, \ldots, x_n) \tag{2.2.1}$$

corresponding to different tuples $(x_1, \ldots, x_n)$ as independent random variables. In particular, this means that the differences $\widetilde{Y} - \widetilde{y}$ and $Y_i - y_i$ are independent random variables.

Technical details. What is a probability distribution for these random variables?

All we know about each of these variables is that its values are located somewhere in the interval $[-\varepsilon, \varepsilon]$. We do not have any reason to assume that some values from this interval are more probable than others, so it is reasonable to assume that all the values are equally probable, i.e., that we have a uniform distribution on this interval.

For this uniform distribution, the mean is 0, and the standard deviation is $\sigma = \dfrac{\varepsilon}{\sqrt{3}}$.

Auxiliary idea: how to get a better estimate for $\widetilde{y}$. In our main algorithm, we apply the computational model F to $n+1$ different tuples. What we suggest it to apply it to one more tuple (making it $n+2$ tuples), namely, computing an approximation

$$M \stackrel{\text{def}}{=} F(\widetilde{x}_1 - \Delta_1, \ldots, \widetilde{x}_n - \Delta_n) \tag{2.2.2}$$

to the value

$$m \stackrel{\text{def}}{=} f(\widetilde{x}_1 - \Delta_1, \ldots, \widetilde{x}_n - \Delta_n). \tag{2.2.3}$$

In the linearized case, one can easily check that

$$\widetilde{y} + \sum_{i=1}^{n} y_i + m = (n+2) \cdot \widetilde{y}, \tag{2.2.4}$$

i.e.,

$$\widetilde{y} = \frac{1}{n+2} \cdot \left(\widetilde{y} + \sum_{i=1}^{n} y_i + m \right). \tag{2.2.5}$$

Thus, we can use the following formula to come up with a new estimate $\widetilde{Y}_{\text{new}}$ for $\widetilde{y}$:

$$\widetilde{Y}_{\text{new}} = \frac{1}{n+2} \cdot \left(\widetilde{Y} + \sum_{i=1}^{n} Y_i + m \right). \tag{2.2.6}$$

For the differences $\Delta\widetilde{y}_{\text{new}} \stackrel{\text{def}}{=} \widetilde{Y}_{\text{new}} - \widetilde{y}$, $\Delta\widetilde{y} \stackrel{\text{def}}{=} \widetilde{y} - \widetilde{y}$, $\Delta y_i \stackrel{\text{def}}{=} Y_i - y_i$, and $\Delta m \stackrel{\text{def}}{=} M - m$, we have the following formula:

$$\Delta\widetilde{y}_{\text{new}} = \frac{1}{n+2} \cdot \left(\Delta\widetilde{y} + \sum_{i=1}^{n} \Delta y_i + \Delta m \right). \tag{2.2.7}$$

The left-hand side is the arithmetic average of $n+2$ independent identically distributed random variables, with mean 0 and variance $\sigma^2 = \dfrac{\varepsilon^2}{3}$. Hence (see, e.g., [29]), the mean of this average $\Delta\widetilde{y}_{\text{new}}$ is the average of the means, i.e., 0, and the variance is equal to $\sigma^2 = \dfrac{\varepsilon^2}{3 \cdot (n+2)} \ll \dfrac{\varepsilon^2}{3} = \sigma^2[\Delta\widetilde{y}]$.

Thus, this average $\widetilde{Y}_{\text{new}}$ is a more accurate estimation of the quantity $\widetilde{y}$ than $\widetilde{Y}$.

Let us use this better estimate for $\widetilde{y}$ when estimating the range of y. Since the average $\widetilde{Y}_{\text{new}}$ is a more accurate estimation of the quantity $\widetilde{y}$ than $\widetilde{Y}$, let us use this average instead of $\widetilde{y}$ when estimating $\overline{Y}$ and $\underline{Y}$. In other words, instead of the estimate (2.1.9), let us use a new estimate $[\widetilde{Y}_{\text{new}} - \Delta_{\text{new}}, \widetilde{Y}_{\text{new}} + \Delta_{\text{new}}]$, where

$$\Delta_{\text{new}} = \sum_{i=1}^{n} \left| Y_i - \widetilde{Y}_{\text{new}} \right|. \tag{2.2.8}$$

Let us estimate the accuracy of this new approximation.

The formula (1.7.3) can be described in the following equivalent form:

$$\overline{y} = \widetilde{y} + \sum_{i=1}^{n} s_i \cdot (y_i - \widetilde{y}\,) = \left(1 - \sum_{i=1}^{n} s_i \right) \cdot \widetilde{y} + \sum_{i=1}^{n} s_i \cdot y_i, \tag{2.2.9}$$

where $s_i \in \{-1, 1\}$ are the signs of the differences $y_i - \widetilde{y}$.

Similarly, we get

$$\overline{Y}_{\text{new}} = \left(1 - \sum_{i=1}^{n} s_i \right) \cdot \widetilde{Y}_{\text{new}} + \sum_{i=1}^{n} s_i \cdot Y_i. \tag{2.2.10}$$

Thus, e.g., for the difference $\Delta\overline{y} \stackrel{\text{def}}{=} \overline{Y}_{\text{new}} - \overline{y}$, we have

$$\Delta\overline{y}_{\text{new}} = \left(1 - \sum_{i=1}^{n} s_i\right) \cdot \Delta\widetilde{y}_{\text{new}} + \sum_{i=1}^{n} s_i \cdot \Delta y_i. \tag{2.2.11}$$

Here, the differences $\Delta\widetilde{y}_{\text{new}}$ and Δy_i are independent random variables. According to the Central Limit Theorem (see, e.g., [29]), for large n, the distribution of a linear combination of many independent random variables is close to Gaussian. The mean of the resulting distribution is the linear combination of the means, thus equal to 0.

The variance of a linear combination $\sum\limits_i k_i \cdot \eta_i$ of independent random variables η_i with variances σ_i^2 is equal to $\sum\limits_i k_i^2 \cdot \sigma_i^2$. Thus, in our case, the variance σ^2 of the difference $\Delta\overline{y}$ is equal to

$$\sigma^2 = \left(1 - \sum_{i=1}^{n} s_i\right)^2 \cdot \frac{\varepsilon^2}{3 \cdot (n+2)} + \sum_{i=1}^{n} \frac{\varepsilon^2}{3}. \tag{2.2.12}$$

Here, since $|s_i| \le 1$, we have $\left|1 - \sum\limits_{i=1}^{n} s_i\right| \le n+1$, so (2.2.12) implies that

$$\sigma^2 \le \frac{\varepsilon^2}{3} \cdot \left(\frac{(n+1)^2}{n+2} + n\right). \tag{2.2.13}$$

Here, $\dfrac{(n+1)^2}{n+2} \le \dfrac{(n+1)^2}{n+1} = n+1$, hence

$$\sigma^2 \le \frac{\varepsilon^2}{3} \cdot (2n+1). \tag{2.2.14}$$

For a normal distribution, with almost complete certainty, all the values are concentrated within k_0 standard deviations away from the mean: within 2σ with confidence 0.95, within 3σ with degree of confidence 0.999, within 6σ with degree of confidence $1 - 10^{-8}$. Thus, we can safely conclude that

$$\overline{y} \le \overline{Y}_{\text{new}} + k_0 \cdot \sigma \le \overline{Y}_{\text{new}} + k_0 \cdot \frac{\varepsilon}{\sqrt{3}} \cdot \sqrt{2n+1}. \tag{2.2.15}$$

Here, inaccuracy grows as $\sqrt{2n+1}$, which is much better than in the traditional approach, where it grows proportionally to $2n+1$ – and we achieve this drastic reduction of the overestimation, basically by using one more run of the program F in addition to the previously used $n+1$ runs.

Similar estimates can be made for the lower bound $\underline{y}$. So, we arrive at the following method.

Resulting method. We know:

- an algorithm $F(x_1, \ldots, x_n)$ that, given the values of the parameters $x_1, \ldots, x_n$, computes the value of the quantity y with a known accuracy ε;
- for each parameter x_i, we know its nominal value $\widetilde{x}_i$ and the largest possible deviation Δ_i from this nominal value.

Based on this information, we need to estimate the range of $y = f(x_1, \ldots, x_n)$ when x_i is from the corresponding intervals $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

We can estimate this range as follows:

1) first, we apply the algorithm F to compute the value

$$\widetilde{Y} = F(\widetilde{x}_1, \ldots, \widetilde{x}_n); \tag{2.2.16}$$

2) then, for each i from 1 to n, we apply the algorithm F to compute the value

$$Y_i = F(\widetilde{x}_1, \ldots, \widetilde{x}_{i-1}, \widetilde{x}_i + \Delta_i, \widetilde{x}_{i+1}, \ldots, \widetilde{x}_n); \tag{2.2.17}$$

3) then, we compute

$$M = F(\widetilde{x}_1 - \Delta_1, \ldots, \widetilde{x}_n - \Delta_n); \tag{2.2.18}$$

4) we compute

$$\widetilde{Y}_{\text{new}} = \frac{1}{n+2} \cdot \left(\widetilde{Y} + \sum_{i=1}^{n} Y_i + M \right); \tag{2.2.19}$$

5) finally, we compute $\left[\widetilde{Y}_{\text{new}} - \widetilde{\Delta}_{\text{new}}, \widetilde{Y}_{\text{new}} + \widetilde{\Delta}_{\text{new}} \right]$, where

$$\widetilde{\Delta}_{\text{new}} = \sum_{i=1}^{n} \left| Y_i - \widetilde{Y}_{\text{new}} \right| + k_0 \cdot \sqrt{2n+1} \cdot \frac{\varepsilon}{\sqrt{3}}, \tag{2.2.20}$$

where k_0 depends on the level of confidence that we can achieve.

Comment. For the Cauchy method, similarly, after computing $\widetilde{Y} = F(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and $Y^{(k)} = F(\widetilde{x}_1 + \eta_1^{(k)}, \ldots, \widetilde{x}_n + \eta_n^{(k)})$, we can compute the improved estimate $\widetilde{Y}_{\text{new}}$ for $\widetilde{y}$ as

$$\widetilde{Y}_{\text{new}} = \frac{1}{N+1} \cdot \left(\widetilde{Y} + \sum_{k=1}^{N} Y^{(k)} \right), \tag{2.2.21}$$

and estimate the parameter Δ based on the more accurate differences $\Delta Y_{\rm new}^{(k)} = Y^{(k)} - \widetilde{Y}_{\rm new}$.

Experimental testing. We tested our approach on the example of the seismic inverse problem in geophysics, where we need to reconstruct the velocity of sound at different spatial locations and at different depths based on the times that it takes for a seismic signal to get from the set-up explosion to different seismic stations. In this reconstruction, we used (a somewhat improved version of) the finite element technique that was originated by John Hole [6]; the resulting techniques are described in [2, 12, 22].

According to Hole's algorithm, we divide the 3-D volume of interest (in which we want to find the corresponding velocities) into a rectangular 3-D grid of N small cubic cells. We assume that the velocity is constant within each cube; the value of velocity in the j-th cube is denoted by v_j. Each observation j means that we know the time t_j that it took the seismic wave to go from the site of the corresponding explosion to the location of the observing sensor.

This algorithm is iterative. We start with the first-approximation model of the Earth, namely, with geology-motivated approximate values $v_i^{(1)}$. At each iteration k, we start with the values $v_i^{(k)}$ and produce the next approximation $v_i^{(k+1)}$ as follows.

First, based on the latest approximation $v_i^{(k)}$, we simulate how the seismic waves propagate from the explosion site to the sensor locations. In the cube that contains the explosion site, the seismic signal propagates in all directions. When the signal's trajectory approaches the border between the two cubes i and i', the direction of the seismic wave changes in accordance with the Snell's law $\dfrac{\sin(\theta_i)}{\sin(\theta_{i'})} = \dfrac{v_i^{(k)}}{v_{i'}^{(k)}}$, where θ_i is the angle between the seismic wave's trajectory in the i-th cube and the vector orthogonal to the plane separating the two cubes. Snell's law enables us to find the trajectory's direction in the next cube i'. Once the way reaches the location of the sensor, we can estimate the travel time as $t_j^{(k)} = \sum\limits_i \dfrac{\ell_{ji}}{v_i^{(k)}}$, where the sum is taken over all the cubes through which this trajectory passes, and ℓ_{ji} is the length of the part of the trajectory that lies in the i-th cube.

Each predicted value $t_j^{(k)}$ is, in general, different from the observed value t_j. To compensate for this difference, the velocity model $v_i^{(k)}$ is corrected: namely, the inverse value $s_i^{(k)} \stackrel{\rm def}{=} \dfrac{1}{v_i^{(k)}}$ is

replaced with an updated value

$$s_i^{(k+1)} = s_i^{(k)} + \frac{1}{n_i} \cdot \sum_j \frac{t_j - t_j^{(k)}}{L_j}, \tag{2.2.22}$$

where the sum is taken over all trajectories that pass through the i-th cube, n_i is the overall number of such trajectories, and $L_j = \sum\limits_i \ell_{ji}$ is the overall length of the j-th trajectory.

Iterations stop when the process converges; for example, it is reasonable to stop the process when the velocity models obtained on two consecutive iterations becomes close:

$$\sum_i \left(v_i^{(k+1)} - v_i^{(k)} \right)^2 \le \varepsilon \tag{2.2.23}$$

for some small value $\varepsilon > 0$. The quality of the resulting solution can be gauged by how well the predicted travel times $t_i^{(k)}$ match the observations; usually, by the root mean square (rms) approximation error $\sqrt{\frac{1}{N} \cdot \sum\limits_i \left(t_i^{(k)} - t_i \right)^2}$.

In this problem, there are two sources of uncertainty. The first is the uncertainty with which we can measure each travel time t_j. The travel time is the difference between the time when the signal arrives at the sensor location and the time of the artificially set explosion. The explosion time is known with a very high accuracy, but the arrival time is not. In the ideal situation, when the only seismic signal comes from the our explosion, we could exactly pinpoint the arrival time as the time when the sensor starts detecting a signal. In real life, there is always a background noise, so we can only determine the arrival time with some accuracy.

The second source of uncertainty comes from the fact that our discrete model is only an approximate description of the continuous real Earth.

In [2, 12, 22], we used the formula (1.7.3) and the Cauchy-based techniques to estimate how the measurement uncertainty affects the results of data processing. To test the method described in this section, we used the above formulas to compute the new value $\widetilde{y}_{\rm new}$. These new values indeed lead to a better fit with data than the original values $\widetilde{y}$: in our 16 experiments, we only in one case, the rms approximation error decreased, on average, by 15%. It should also be mentioned that only in one case the rms approximation error increased (and not much, only by 7%); in all other 15 cases, the rms approximation error decreased.

2.3 Future Work: Can We Further Improve the Accuracy?

How to improve the accuracy: a straightforward approach. As we have mentioned, the inaccuracy $F \neq f$ is caused by the fact that we are using a Finite Element method with a finite size elements. In the traditional Finite Element method, when we assume that the values of each quantity within each element are constant, this inaccuracy comes from the fact that we ignore the difference between the values of the corresponding parameters within each element. For elements of linear size h, this inaccuracy Δy is proportional to $y' \cdot h$, where y' is the spatial derivative of y. In other words, the inaccuracy is proportional to the linear size h.

A straightforward way to improve the accuracy is to decrease h. For example, if we reduce h to $\dfrac{h}{2}$, then we decrease the resulting model inaccuracy ε to $\dfrac{\varepsilon}{2}$.

This decrease requires more computations. The number of computations is, crudely speaking, proportional to the number of nodes. Since the elements fill the original area, and each element has volume h^3, the number of such elements is proportional to h^{-3}.

So, if we go from the original value h to the smaller value h', then we increase the number of computations by a factor of

$$K \stackrel{\text{def}}{=} \frac{h^3}{(h')^3}. \tag{2.3.1}$$

This leads to decreasing the inaccuracy by a factor of $\dfrac{h}{h'}$, which is equal to $\sqrt[3]{K}$.

For example, in this straightforward approach, if we want to decrease the accuracy in half $\left(\sqrt[3]{K} = 2\right)$, we will have to increase the number of computation steps by a factor of $K = 8$.

An alternative approach: description. An alternative approach is as follows. We select K small vectors $\left(\Delta_1^{(k)}, \ldots, \Delta_n^{(k)}\right)$, $1 \le k \le K$, which add up to 0. For example, we can arbitrarily select the first $K-1$ vectors and take $\Delta x_i^{(K)} = -\sum\limits_{k=1}^{K-1} \Delta_i^{(k)}$.

Then, every time we need to estimate the value $f(x_1, \ldots, x_n)$, instead of computing $F(x_1, \ldots, x_n)$, we compute the average

$$F_K(x_1, \ldots, x_n) = \frac{1}{K} \cdot \sum_{k=1}^{K} F\left(x_1 + \Delta_1^{(k)}, \ldots, x_n + \Delta_n^{(k)}\right). \tag{2.3.2}$$

Why this approach decreases inaccuracy. We know that $F(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) = f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) + \delta y$, where, in the small vicinity of the original tuple $(x_1, \ldots, x_n)$, the expression $f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n)$ is linear, and the differences δy are independent random variables with zero mean.

Thus, we have

$$F_K(x_1, \ldots, x_n) = \frac{1}{K} \cdot \sum_{k=1}^{K} f\left(x_1 + \Delta_1^{(k)}, \ldots, x_n + \Delta_n^{(k)}\right) + \frac{1}{K} \cdot \sum_{k=1}^{K} \delta y^{(k)}. \tag{2.3.3}$$

Due to linearity and the fact that $\sum\limits_{k=1}^{K} \Delta_i^{(k)} = 0$, the first average in (2.3.3) is equal to $f(x_1, \ldots, x_n)$. The second average is the average of K independent identically distributed random variables, and we have already recalled that this averaging decreases the inaccuracy by a factor of $\sqrt{K}$.

Thus, in this alternative approach, we increase the amount of computations by a factor of K, and as a result, we decrease the inaccuracy by a factor of $\sqrt{K}$.

The new approach is better than the straightforward one. In general, $\sqrt{K} > \sqrt[3]{K}$. Thus, with the same increase in computation time, the new method provides a better improvement in accuracy than the straightforward approach.

Comment. The above computations only refer to the traditional Finite Element approach, when we approximate each quantity within each element by a *constant*. In many real-life situations, it is useful to approximate each quantity within each element not by a constant, but rather by a *polynomial* of a given order (see, e.g., [30]): by a linear function, by a quadratic function, etc. In this case, for each element size h, we have smaller approximation error but larger amount of computations. It is desirable to extend the above analysis to such techniques as well.

Chapter 3

How to Speed Up Computations – by Processing Different Types of Uncertainty Separately

3.1 Case for Which A Speed-Up Is Possible: A Description

In this section, we describe the cases of fuzzy uncertainty for which the computations can be made faster.

Simplest case: when all fuzzy numbers are of the same type. Sometimes, all membership functions are "of the same type", i.e., they all have the form $\mu(z) = \mu_0(k \cdot z)$ for some fixed symmetric function $\mu_0(z)$.

For example, frequently, we consider symmetric triangular functions; all these functions can be obtained from the standard triangular function $\mu_0(z) = \max(1 - |z|, 0)$ by using an appropriate constant $k > 0$.

In this case, we can speed up computations. Let us show that in this simple case, we can drastically reduce the computation time that is needed to compute the desired α-cuts ${}^{\alpha}\Delta$.

Indeed, let $[-{}^{\alpha}\Delta_0, {}^{\alpha}\Delta_0]$ denote an α-ut corresponding to the membership function $\mu_0(z)$. This means that the inequality $\mu_0(z) \geq \alpha$ is equivalent to $|z| \leq {}^{\alpha}\Delta_0$. Then, for the membership function $\mu(z) = \mu_0(k \cdot z)$, the inequality $\mu(z) \geq \alpha$ describing its α-cut is equivalent to $\mu_0(k \cdot z) \geq \alpha$, i.e., to to $k \cdot |z| \leq {}^{\alpha}\Delta_0$ and thus, $|z| \leq \dfrac{1}{k} \cdot {}^{\alpha}\Delta_0$. Hence, the half-widths of the corresponding α-cuts are

equal to

$$^{\alpha}\Delta = \frac{1}{k} \cdot {}^{\alpha}\Delta_0. \tag{3.1.1}$$

This equality holds for all α, in particular, for $\alpha = 0$, when we get

$$^{0}\Delta = \frac{1}{k} \cdot {}^{0}\Delta_0. \tag{3.1.2}$$

By dividing (3.1.1) by (3.1.2), we conclude that

$$\frac{^{\alpha}\Delta}{^{0}\Delta} = f(\alpha), \tag{3.1.3}$$

where we denoted

$$f(\alpha) \stackrel{\text{def}}{=} \frac{^{\alpha}\Delta_0}{^{0}\Delta_0}. \tag{3.1.4}$$

For example, for a triangular membership function, we have

$$f(\alpha) = 1 - \alpha. \tag{3.1.5}$$

Thus, if we know the type of the membership function (and hence, the corresponding function $f(\alpha)$), and we know the 0-cut, then we can reconstruct all α-cuts as

$$^{\alpha}\Delta = f(\alpha) \cdot {}^{0}\Delta, \tag{3.1.6}$$

i.e., by simply multiplying the 0-cuts by an appropriate factor $f(\alpha)$.

So, if all the membership functions $\mu_i(\Delta x_i)$ and $\mu_m(\Delta m)$ are of the same type, then, for every α, we have $^{\alpha}\Delta_i = f(\alpha) \cdot {}^{0}\Delta_i$. Substituting these expressions into the formula (3.1.1), we conclude that

$$^{\alpha}\Delta = \sum_{i=1}^{n} |c_i| \cdot f(\alpha) \cdot {}^{0}\Delta_i = f(\alpha) \cdot \sum_{i=1}^{n} |c_i| \cdot {}^{0}\Delta_i, \tag{3.1.7}$$

i.e., that

$$^{\alpha}\Delta = f(\alpha) \cdot {}^{0}\Delta. \tag{3.1.8}$$

Thus, if all the membership functions are of the same type $\mu_0(z)$, there is no need to apply the formula (**??**) eleven times: it is sufficient to compute it only once, e.g., for $\alpha = 0$. To find all other values $^{\alpha}\Delta$, we can then simply multiply the resulting value $^{0}\Delta$ by the factors $f(\alpha)$ corresponding to the type $\mu_0(z)$.

A more general case. A more general case is when we have a list of T different types of uncertainty – i.e., types of membership functions – and each approximation error Δx_i consists of $\le T$ components of the corresponding type. In other words, for each i, we have

$$\Delta x_i = \sum_{t=1}^{T} \Delta x_{i,t}, \tag{3.1.9}$$

where $\Delta x_{i,t}$ are uncertainties of the t-th type, and we know the corresponding membership functions $\mu_{i,t}(\Delta x_{i,t})$.

For example, type $t = 1$ may correspond to intervals (which are, of course, a particular case of fuzzy uncertainty), type $t = 2$ may correspond to triangular membership functions, etc.

How this case is processed now.

- First, we use the known membership functions $\mu_{i,t}(\Delta x_{i,t})$ to find the memberships functions $\mu_i(\Delta x_i)$ that correspond to the sums (3.1.9).
- Then, we use these membership functions to compute the desired membership function $\mu(\Delta y)$.

On the second stage, we apply the formula (1.8.4) eleven times.

3.2 Main Idea of This Chapter

Main idea. As we have mentioned, at present, to find the membership function for Δy, we use the formula (1.4.5), in which each of the terms Δx_i is computed by using the formula (3.1.9).

A natural alternative idea is:

- to substitute the expressions (3.1.9) into the formula (1.4.5), and then
- to regroup the resulting terms by combining all the components of the same type t.

Comment. This idea was published in [31].

Technical details. Substituting the expressions (3.1.9) into the formula (1.8.4), we conclude that

$$\Delta y = \sum_{i=1}^{n} c_i \cdot \left(\sum_{t=1}^{T} \Delta x_{i,t} \right). \tag{3.2.1}$$

Now, grouping together all the terms corresponding to each type t, we conclude that

$$\Delta y = \sum_{t=1}^{T} \Delta y_t, \tag{3.2.2}$$

where

$$\Delta y_t \stackrel{\text{def}}{=} \sum_{i=1}^{n} c_i \cdot \Delta x_{i,t}. \tag{3.2.3}$$

This representation suggests the following new algorithm.

New algorithm: idea. For each t, since we are combining membership functions of the same type, computing these membership functions requires a single application of the formula (1.8.4), to compute the value ${}^0\Delta_t$ corresponding to $\alpha = 0$. The values corresponding to other values α, we simply multiply this value ${}^0\Delta_t$ by the coefficients $f_t(\alpha)$ corresponding to membership functions of type t.

Then, we add the resulting membership functions – by adding the corresponding α-cuts. Let us describe the resulting algorithm in detail.

New algorithm: in detail. We start with the values ${}^0\Delta_{i,t}$ for which the corresponding symmetric intervals $[-{}^0\Delta_{i,t}, {}^0\Delta_{i,t}]$ describe the 0-cuts of the corresponding membership functions $\mu_{i,t}(\Delta x_{i,t})$.

Based on these 0-cuts, we compute, for each type t, the values

$${}^0\Delta_t = \sum_{i=1}^{n} |c_i| \cdot {}^0\Delta_{i,t}. \tag{3.2.4}$$

Then, for $\alpha = 0$, $\alpha = 0.1$, $\ldots$, and for $\alpha = 1.0$, we compute the values

$${}^\alpha\Delta_t = f_t(\alpha) \cdot {}^0\Delta_t, \tag{3.2.5}$$

where the function $f_t(\alpha)$ is known for each type t. Finally, we add up α-cuts corresponding to different types t, to come up with the expression

$${}^\alpha\Delta = \sum_{t=1}^{T} {}^\alpha\Delta_t. \tag{3.2.6}$$

Comment. We can combine the steps (3.2.5) and (3.2.6) into a single step, in which we use the following formula:

$$^{\alpha}\Delta = \sum_{t=1}^{T} f_t(\alpha) \cdot {}^{0}\Delta_t. \tag{3.2.7}$$

This new algorithm is much faster. The original algorithm computed the formula (1.8.4) eleven times. The new algorithm uses the corresponding formula (3.2.4) (the analogue of the formula (1.8.4)) T times, i.e., as many times as there are types. All the other computations are much faster, since they do not grow with the input size n.

Thus, if the number T of different types is smaller than eleven, the new methods is much faster.

For example, if we have $T = 2$ different types, e.g., intervals and triangular membership functions, then we get a $\dfrac{11}{2} = 5.5$ times speedup.

Conclusion. We can therefore conclude that sometimes, it is beneficial to process different types of uncertainty separately – namely, it is beneficial when we have ten or fewer different types of uncertainty. The fewer types of uncertainty we have, the faster the resulting algorithm.

Future work. In this paper, we show the handling uncertainty by types of uncertainty can make computations faster. We plan to test this idea of several actual data processing examples, to measure the actual gain, and thus, to make proper recommendations on when this idea is indeed beneficial. We also plan to extend this idea to other types of uncertainty, in particular, to different types of probabilistic uncertainty.

Chapter 4

Towards a Better Understandability of Uncertainty-Estimating Algorithms: Explaining the Need for Non-Realistic Monte-Carlo Simulations

4.1 Problem: The Existing Monte-Carlo Method is Not Realistic

Monte-Carlo method for the case of probabilistic uncertainty is realistic. The probabilistic Monte-Carlo method is *realistic* in the following sense:

- we know that each measurement error Δx_i is distributed according to the probability distribution $\rho_i(\Delta x_i)$, and
- this is exactly how we simulate the measurement errors: to simulate each value $\Delta_i^{(k)}$, we use the exact same distribution $\rho_i(\Delta x_i)$.

In contrast, the Monte-Carlo method for the case of interval uncertainty is not realistic. In the case of uncertainty, all we know is that the measurement errors are always located within the corresponding interval $[-\Delta_i, \Delta_i]$. We do not know how frequently measurement errors

will be observed in different parts of this interval. In other words, we do not know the probability distribution of the measurement errors – we only know that this (unknown) probability distribution is located on the interval $[-\Delta_i, \Delta_i]$ *with probability 1.*

With this in mind, a *realistic* Monte-Carlo simulation would mean that for simulating the values $\Delta_i^{(k)}$, we select a probability distribution is located on the corresponding interval $[-\Delta_i, \Delta_i]$ with probability 1. Instead, the existing Monte-Carlo method for interval uncertainty uses Cauchy distribution – and it is known that for this distribution, for any interval, there is a non-zero probability to be outside this interval, and thus, the *probability* to be inside the interval $[-\Delta_i, \Delta_i]$ *is smaller than 1.*

A natural question. A natural question is:

- is this a limitation of the existing method, and an alternative realistic Monte-Carlo method is possible for the case of interval uncertainty,
- or this is a limitation of the problem, and no realistic Monte-Carlo method is possible for interval uncertainty.

What we do in this chapter. In the two following sections, we prove that the non-realistic character of the existing Monte-Carlo method for interval uncertainty is a limitation of the problem. In other words, we prove that no realistic Monte-Carlo is possible for the case of interval uncertainty.

In an additional section, we explain why Cauchy distribution should be used.

The results from this chapter first appeared in [24].

4.2 Proof That Realistic Interval Monte-Carlo Techniques Are Not Possible: Case of Independent Variables

To prove the desired result, it is sufficient to consider a simple case. To prove the desired impossibility result – that no realistic Monte-Carlo algorithm is possible that would *always*

compute the desired range $\mathbf{y}$ – it is sufficient to prove that we cannot get the correct estimate for *one* specific function $f(x_1, \ldots, x_n)$.

As such a function, let us consider the simple function $f(x_1, \ldots, x_n) = x_1 + \ldots + x_n$. In this case, all the partial derivatives are equal to 1, i.e., $c_1 = \ldots = c_n = 1$ and thus,

$$\Delta y = \Delta x_1 + \ldots + \Delta x_n. \tag{4.2.1}$$

If we assume that each variables Δx_i takes value from the interval $[-\delta, \delta]$, then the range of possible values of the sum is $[-\Delta, \Delta]$, where $\Delta = n \cdot \delta$.

Analysis of the problem. Under Monte-Carlo simulations, we have

$$\Delta y^{(k)} = \Delta x_1^{(k)} + \ldots + \Delta x_n^{(k)}. \tag{4.2.2}$$

We assumed that the probability distributions corresponding to all i are independent.

Since the original problem is symmetric with respect to permutations, the corresponding distribution is also symmetric, so all $\Delta_i^{(k)}$ are identically distributed. Thus, the value Δy is the sum of several (n) independent identically distributed random variables.

It is known that due to the Central Limit Theorem (see, e.g., [29]), when n increases, the distribution of the sum tends to Gaussian. So, for large n, this distribution is close to Gaussian.

The Gaussian distribution is uniquely determined by its mean μ and variance $V = \sigma^2$. The mean of the sum is equal to the sum of the means, so $\mu = n \cdot \mu_0$, where μ_0 is the mean of the distribution used to simulate each Δx_i. For independent random variables, the variance of the sum is equal to the sum of the variances, so $V = n \cdot V_0$, where V_0 is the variance of the distribution used to simulate each Δx_i. Thus, $\sigma = \sqrt{V} = \sqrt{V_0} \cdot \sqrt{n}$.

It is well known that for a normal distribution, with very high confidence, all the values are contained in a k-sigma interval $[\mu - k \cdot \sigma, \mu + k \cdot \sigma]$:

- with probability $\approx 99.9\%$, the value will be in 3-sigma interval,
- with probability $\approx 1 - 10^{-8}$, the value will be in the 6-sigma interval, etc.

Thus, with high confidence, all the values obtained from simulation are contained in the interval $[\mu - k \cdot \sigma, \mu + k \cdot \sigma]$ of width $2k \cdot \sigma = 2k \cdot \sqrt{V_0} \cdot \sqrt{n}$.

For large n, this interval has the size const $\cdot \sqrt{n}$. On the other hand, we want the range $[-\Delta, \Delta]$ whose width is $2\Delta = 2\delta \cdot n$. So, when n is large, the simulated values occupy a part of the desired interval that tends to 0:

$$\frac{2k \cdot \sqrt{V_0} \cdot \sqrt{n}}{2\delta \cdot n} = \frac{\text{const}}{\sqrt{n}} \to 0. \tag{4.2.3}$$

So, in the independence case, the impossibility is proven.

4.3 Proof That Realistic Interval Monte-Carlo Techniques Are Not Possible: General Case

To prove the desired negative result, it is sufficient to consider a simple case. Similarly to the previous section, to prove the impossibility result in the *general* case, it is also sufficient to prove the impossibility for *some* of the functions.

In this proof, we will consider functions

$$f(x_1, \ldots, x_n) = s_1 \cdot x_1 + \ldots + s_n \cdot x_n, \tag{4.3.1}$$

where $s_i \in \{-1, 1\}$.

For each of these functions,

$$\Delta y = s_1 \cdot \Delta x_1 + \ldots + s_n \cdot \Delta x_n, \tag{4.3.2}$$

so we have $c_i = s_i$. Similarly to the previous section, we assume that each of the unknowns Δx_i takes value from the interval $[-\delta, \delta]$, for some known value $\delta > 0$.

For each of these functions, $|c_i| = |s_i| = 1$, so the desired range is the same for all these functions and is equal to $[-\Delta, \Delta]$, where

$$\Delta = \sum_{i=1}^{n} |c_i| \cdot \Delta_i = n \cdot \delta. \tag{4.3.3}$$

Towards a precise formulation of the problem. Suppose that we want to find the range $[-\Delta, \Delta]$ with some relative accuracy ε. To get the range from simulations, we need to make sure

that some of the simulated results are ε-close to Δ, i.e., that

$$\left|\sum_{i=1}^{n} s_i \cdot \Delta x_i^{(k)} - n \cdot \delta\right| \le \varepsilon \cdot n \cdot \delta, \tag{4.3.4}$$

or, equivalently,

$$n \cdot \delta \cdot (1-\varepsilon) \le \sum_{i=1}^{n} s_i \cdot \Delta x_i^{(k)} \le n \cdot \delta \cdot (1+\varepsilon). \tag{4.3.5}$$

We are interested in realistic Monte-Carlo simulations, for which $|\Delta_i^{(k)}| \le \delta$ for all i. Thus, we always have

$$\sum_{i=1}^{n} s_i \cdot \Delta x_i^{(k)} \le n \cdot \delta < n \cdot \delta \cdot (1+\varepsilon). \tag{4.3.6}$$

So, the right-hand inequality is always satisfied, and it is thus sufficient to make sure that we have

$$\sum_{i=1}^{n} s_i \cdot \Delta x_i^{(k)} \ge n \cdot \delta \cdot (1-\varepsilon) \tag{4.3.7}$$

for some simulation k.

For this inequality to be true with some certainty, we need to make sure that the probability of this inequality exceed some constant $p > 0$. Then, if we run $1/p$ simulations, then with high probability, the inequality will be satisfied for at least one of these simulations. Thus, we arrive at the following condition.

Definition 4.3.1. *Let $\varepsilon > 0$, $\delta > 0$, and $p \in (0,1)$. We say that a probability distribution on the set of all vectors*

$$(\Delta_1 \ldots, \Delta x_n) \in [-\delta, \delta] \times \ldots \times [-\delta, \delta] \tag{4.3.8}$$

is a (p,ε)-realistic Monte-Carlo estimation of interval uncertainty *if for every set of values $s_i \in \{-1, 1\}$, we have*

$$\mathrm{Prob}(s_1 \cdot \Delta x_1 + \ldots + s_n \cdot \Delta x_n \ge n \cdot \delta \cdot (1-\varepsilon)) \ge p. \tag{4.3.9}$$

Proposition 4.3.1. *Let $\delta > 0$ and $\varepsilon > 0$. If for every n, we have a (p_n, ε)-realistic Monte-Carlo estimation of interval uncertainty, then $p_n \le \beta \cdot n \cdot c^n$ for some $\beta > 0$ and $c < 1$..*

Comments.

- As we have mentioned, when the probability is equal to p, we need $1/p$ simulations to get the desired estimates. Due to Proposition 4.3.1, to get a realistic Monte-Carlo estimate for the interval uncertainty, we thus need

$$\frac{1}{p_n} \sim \frac{c^{-n}}{\beta \cdot n} \tag{4.3.10}$$

simulations. For large n, we have

$$\frac{c^{-n}}{\beta \cdot n} \gg n + 1. \tag{4.3.11}$$

Thus, the above results shows that realistic Monte-Carlo simulations require even more computational time than numerical differentiation. This defeats the main purpose for using Monte-Carlo techniques, which is – for our problem – to decrease the computation time.

- It is worth mentioning that if we allow p_n to be exponentially decreasing, then a realistic Monte-Carlo estimation of interval uncertainty is possible: e.g., we can take Δx_i to be independent and equal to δ or to $-\delta$ with equal probability 0.5. In this case, with probability 2^{-n}, we get the values $\Delta x_i = s_i \cdot \delta$ for which

$$\sum_{i=1}^{n} s_i \cdot \Delta x_i = \sum_{i=1}^{n} \delta = n \cdot \delta > n \cdot \delta \cdot (1 - \varepsilon). \tag{4.3.12}$$

Thus, for this probability distribution, for each combination of signs s_i, we have

$$\text{Prob}(s_1 \cdot \Delta x_1 + \ldots + s_n \cdot \Delta x_n \geq n \cdot \delta \cdot (1 - \varepsilon)) = p_n = 2^{-n}. \tag{4.3.13}$$

Proof of Proposition 4.3.1. Let us pick some $\alpha \in (0, 1)$. Let us denote, by m, the number of indices i or which $s_i \cdot \Delta x_i > \alpha \cdot \delta$. Then, if we have

$$s_1 \cdot \Delta x_1 + \ldots + s_n \cdot \Delta x_n \geq n \cdot \delta \cdot (1 - \varepsilon), \tag{4.3.14}$$

then for $n - m$ indices, we have $s_i \cdot \Delta x_i \leq \alpha \cdot \delta$ and for the other m indices, we have $s_i \cdot \Delta x_i \leq \delta$. Thus,

$$n \cdot \delta \cdot (1 - \varepsilon) \leq \sum_{i=1}^{n} s_i \cdot \Delta x_i \leq m \cdot \delta + (n - m) \cdot \alpha \cdot \delta. \tag{4.3.15}$$

Dividing both sides of this inequality by δ, we get

$$n \cdot (1 - \varepsilon) \leq m + (n - m) \cdot \alpha, \tag{4.3.16}$$

hence $n \cdot (1-\alpha-\varepsilon) \le m \cdot (1-\alpha)$ and thus,

$$m \ge n \cdot \frac{1-\alpha-\varepsilon}{1-\alpha}. \tag{4.3.17}$$

So, we have at least

$$n \cdot \frac{1-\alpha-\varepsilon}{1-\alpha} \tag{4.3.18}$$

indices for which Δx_i has the same sign as s_i (and for which $|\Delta x_i| > \alpha \cdot \delta$). This means that for the vector corresponding to a tuple $(s_1, \ldots, s_n)$, at most

$$n \cdot \frac{\varepsilon}{1-\alpha-\varepsilon} \tag{4.3.19}$$

indices have a different sign than s_i.

It is, in principle, possible that the same tuple $(\Delta x_1, \ldots, \Delta x_n)$ can serve two different tuples $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$. However, in this case:

- going from s_i to $\mathrm{sign}(\Delta x_i)$ changes at most $n \cdot \frac{\varepsilon}{1-\alpha-\varepsilon}$ signs, and
- going from $\mathrm{sign}(\Delta x_i)$ to s'_i also changes at most $n \cdot \frac{\varepsilon}{1-\alpha-\varepsilon}$ signs.

Thus, between the tuples s and s', at most $2 \cdot \frac{\varepsilon}{1-\alpha-\varepsilon}$ signs are different. In other words, for the Hamming distance

$$d(s, s') \stackrel{\text{def}}{=} \#\{i : s_i \ne s'_i\}, \tag{4.3.20}$$

we have

$$d(s, s') \le 2 \cdot n \cdot \frac{\varepsilon}{1-\alpha-\varepsilon}. \tag{4.3.21}$$

Thus, if

$$d(s, s') > 2 \cdot n \cdot \frac{\varepsilon}{1-\alpha-\varepsilon}, \tag{4.3.22}$$

then no tuples $(\Delta x_1, \ldots, \Delta x_n)$ can serve both sign tuples s and s'. In this case, the corresponding sets of tuples for which

$$s_1 \cdot \Delta x_1 + \ldots + s_n \cdot \Delta x_n \ge n \cdot \delta \cdot (1-\varepsilon) \tag{4.3.23}$$

and

$$s'_1 \cdot \Delta x_1 + \ldots + s'_n \cdot \Delta x_n \ge n \cdot \delta \cdot (1-\varepsilon) \tag{4.3.24}$$

do not intersect. Hence, the probability that the randomly selected tuple belongs to one of these sets is equal to the sum of the corresponding probabilities. Since each of the probabilities is greater than or equal to p, the resulting probability is equal to $2p$.

If we have M sign tuples $s^{(1)}, \ldots, s^{(M)}$ for which

$$d(s^{(i)}, s^{(j)}) > 2 \cdot \frac{\varepsilon}{1 - \alpha - \varepsilon} \tag{4.3.25}$$

for all $i \neq j$, then similarly, the probability that the tuple $(\Delta x_1, \ldots, \Delta x_n)$ serves one of these sign tuples is greater than or equal to $M \cdot p$. On the other hand, this probability is ≤ 1, so we conclude that $M \cdot p \leq 1$ and $p \leq \frac{1}{M}$.

So, to prove that p_n is exponentially decreasing, it is sufficient to find the sign tuples whose number M is exponentially increasing.

Let us denote $\beta \stackrel{\text{def}}{=} \frac{\varepsilon}{1 - \alpha - \varepsilon}$. Then, for each sign tuple s, the number t of all sign tuples s' for which $d(s, s') \leq \beta \cdot n$ is equal to the sum of:

- the number of tuples $\binom{n}{0}$ that differ from s in 0 places,
- the number of tuples $\binom{n}{1}$ that differ from s in 1 place, $\ldots$,
- the number of tuples $\binom{n}{\beta \cdot n}$ that differ from s in $\beta \cdot n$ places,

i.e.,

$$t = \binom{n}{0} + \binom{n}{1} + \ldots + \binom{n}{n \cdot \beta}. \tag{4.3.26}$$

When $\beta < 0.5$ and $\beta \cdot n < \frac{n}{2}$, the number of combinations $\binom{n}{k}$ increases with k, so $t \leq \beta \cdot n \cdot \binom{n}{\beta \cdot n}$. Here,

$$\binom{a}{b} = \frac{a!}{b! \cdot (a - b)!}. \tag{4.3.27}$$

Asymptotically,

$$n! \sim \left(\frac{n}{e}\right)^n, \tag{4.3.28}$$

so

$$t \leq \beta \cdot n \cdot \frac{\left(\frac{n}{e}\right)^n}{\left(\frac{\beta \cdot n}{e}\right)^{\beta \cdot n} \cdot \left(\frac{(1 - \beta) \cdot n}{e}\right)^{(1-\beta) \cdot n}}. \tag{4.3.29}$$

One can see that the term n^n in the numerator cancels with the term $n^{\beta \cdot n} \cdot n^{(1-\beta) \cdot n} = n^n$ in the denominator. Similarly, the terms e^n and $e^{\beta \cdot n} \cdot e^{(1-\beta) \cdot n} = e^n$ cancel each other, so we conclude that

$$t \leq \beta \cdot n \cdot \left(\frac{1}{\beta^\beta \cdot (1-\beta)^{1-\beta}} \right)^n. \tag{4.3.30}$$

Here,

$$\gamma \stackrel{\text{def}}{=} \frac{1}{\beta^\beta \cdot (1-\beta)^{1-\beta}} = \exp(S), \tag{4.3.31}$$

where

$$S \stackrel{\text{def}}{=} -\beta \cdot \ln(\beta) - (1-\beta) \cdot \ln(1-\beta) \tag{4.3.32}$$

is Shannon's entropy. It is well known (and easy to check by differentiation) that its largest possible values is attained when $\beta = 0.5$, in which case $S = \ln(2)$ and $\gamma = \exp(S) = 2$. When $\beta < 0.5$, we have $S < \ln(2)$, thus, $\gamma < 2$, and $t \leq \beta \cdot n \cdot \gamma^n$ for some $\gamma < 2$.

Let us now construct the desired collection of sign tuples $s^{(1)}, \ldots, s^{(M)}$.

- We start with some sign tuple $s^{(1)}$, e.g., $s^{(1)} = (1, \ldots, 1)$.
- Then, we dismiss $t \leq \gamma^n$ tuples which are $\leq \beta$-close to s, and select one of the remaining tuples as $s^{(2)}$.
- We then dismiss $t \leq \gamma^n$ tuples which are $\leq \beta$-close to $s^{(2)}$. Among the remaining tuples, we select the tuple $s^{(3)}$, etc.

Once we have selected M tuples, we have thus dismissed $t \cdot M \leq \beta \cdot n \cdot \gamma^n \cdot M$ sign tuples. So, as long as this number is smaller than the overall number 2^n of sign tuples, we can continue selecting.

This procedure ends when we have selected M tuples for which $\beta \cdot n \cdot \gamma^n \cdot M \geq 2^n$. Thus, we have selected

$$M \geq \left(\frac{2}{\gamma} \right)^n \cdot \frac{1}{\beta \cdot n} \tag{4.3.33}$$

tuples. So, we have indeed selected exponentially many tuples.

Hence,

$$p_n \leq \frac{1}{M} \leq \beta \cdot n \cdot \left(\frac{\gamma}{2} \right)^n, \tag{4.3.34}$$

i.e.,

$$p_n \leq \beta \cdot n \cdot c^n, \tag{4.3.35}$$

where

$$c \stackrel{\text{def}}{=} \frac{\gamma}{2} < 1. \tag{4.3.36}$$

So, the probability p_n is indeed exponentially decreasing. The proposition is proven.

4.4 Why Cauchy Distribution

Formulation of the problem. We want to find a family of probability distributions with the following property:

- when we have several independent variables $X_1, \ldots, X_n$ distributed according to a distribution with this family with parameters $\Delta_1, \ldots, \Delta_n$,
- then each linear combination $Y = c_1 \cdot X_1 + \ldots + c_n \cdot X_n$ has the same distribution as $\Delta \cdot X$, where X corresponds to parameter 1, and $\Delta = \sum_{i=1}^{n} |c_i| \cdot \Delta_i$.

In particular, for the case when $\Delta_1 = \ldots = \Delta_n = 1$, the problem becomes even easier to describe, since then, we only need to find *one* probability distribution: corresponding to the value 1. In this case, the desired property of this probability distribution is as follows:

- if we have n independent identically distributed random variables $X_1, \ldots, X_n$,
- then each linear combination $Y = c_1 \cdot X_1 + \ldots + c_n \cdot X_n$ has the same distribution as $\Delta \cdot X_i$, where $\Delta = \sum_{i=1}^{n} |c_i|$.

Let us describe all probability distributions that satisfy this property.

Analysis of the problem. First, we observe that from the above condition, for $n = 1$ and $c_1 = -1$, we conclude that $-X$ and X should have exactly the same probability distribution, i.e., that the desired probability distribution be symmetric with respect to 0 (even).

A usual way to describe a probability distribution is to use a probability density function $\rho(x)$, but often, it is more convenient to use its Fourier transform, i.e., in probabilistic terms, the *characteristic function* $\chi_X(\omega) \stackrel{\text{def}}{=} E[\exp(\mathrm{i} \cdot \omega \cdot X)]$, where $E[.]$ indicates the expected value of the corresponding quantity and $\mathrm{i} \stackrel{\text{def}}{=} \sqrt{-1}$.

The advantage of using a characteristic function is that for the sum $S = X_1 + X_2$ of two independent variables $X_1 + X_2$, we have

$$\chi_S(\omega) = E[\exp(\mathrm{i} \cdot \omega \cdot S)] = E[\exp(\mathrm{i} \cdot \omega \cdot (X_1 + X_2)] = E[\exp(\mathrm{i} \cdot \omega \cdot X_1 + \mathrm{i} \cdot \omega \cdot X_2)] =$$

$$E[\exp(\mathrm{i} \cdot \omega \cdot X_1) \cdot \exp(\mathrm{i} \cdot \omega \cdot X_2)]. \tag{4.4.1}$$

Since X_1 and X_2 are independent, the variables $\exp(\mathrm{i} \cdot \omega \cdot X_1)$ and $\exp(\mathrm{i} \cdot \omega \cdot X_2)$ are also independent, and thus,

$$\chi_S(\omega) = E[\exp(\mathrm{i} \cdot \omega \cdot X_1) \cdot \exp(\mathrm{i} \cdot \omega \cdot X_2)] = E[\exp(\mathrm{i} \cdot \omega \cdot X_1)] \cdot E[\exp(\mathrm{i} \cdot \omega \cdot X_2)] =$$

$$\chi_{X_1}(\omega) \cdot \chi_{X_2}(\omega). \tag{4.4.2}$$

Similarly, for a linear combination $Y = \sum\limits_{i=1}^{n} c_i \cdot X_i$, we have

$$\chi_Y(\omega) = E[\exp(\mathrm{i} \cdot \omega \cdot Y)] = E\left[\exp\left(\mathrm{i} \cdot \omega \cdot \sum_{i=1}^{n} c_i \cdot X_i\right)\right] = E\left[\exp\left(\sum_{i=1}^{n} \mathrm{i} \cdot \omega \cdot c_i \cdot X_i\right)\right] =$$

$$E\left[\prod_{i=1}^{n} \exp\left(\mathrm{i} \cdot \omega \cdot c_i \cdot X_i\right)\right] = \prod_{i=1}^{n} E[\exp(\mathrm{i} \cdot (\omega \cdot c_i) \cdot X_i] = \prod_{i=1}^{n} \chi_X(\omega \cdot c_i). \tag{4.4.3}$$

The desired property is that the linear combination Y should have the same distribution as $\Delta \cdot X$. Thus, the characteristic function $\chi_Y(\omega)$ should be equal to the characteristic function of $\Delta \cdot X$, i.e., to

$$\chi_{\Delta \cdot X}(\omega) = E[\exp(\mathrm{i} \cdot \omega \cdot (\Delta \cdot X))] = E[\exp(\mathrm{i} \cdot (\omega \cdot \Delta) \cdot X)] = \chi_X(\omega \cdot \Delta). \tag{4.4.4}$$

By comparing expressions (4.4.3) and (4.4.4), we conclude that for all possible combinations $c_1, \ldots, c_n$, the desired characteristic function $\chi_X(\omega)$ should satisfy the equality

$$\chi_X(c_1 \cdot \omega) \cdot \ldots \cdot \chi_X(c_n \cdot \omega) = \chi_X((|c_1| + \ldots + |c_n|) \cdot \omega). \tag{4.4.5}$$

In particular, for $n = 1$, $c_1 = -1$, we get $\chi_X(-\omega) = \chi_X(\omega)$, so $\chi_X(\omega)$ should be an even function. For $n = 2$, $c_1 > 0$, $c_2 > 0$, and $\omega = 1$, we get

$$\chi_X(c_1 + c_2) = \chi_X(c_1) \cdot \chi_X(c_2). \tag{4.4.6}$$

The characteristic function should be measurable, and it is known that the only measurable function with the property (4.4.6) has the form $\chi_X(\omega) = \exp(-k \cdot \omega)$ for some k; see, e.g., [1].

Due to evenness, for a general ω, we get $\chi_X(\omega) = \exp(-k \cdot |\omega|)$. By applying the inverse Fourier transform, we conclude that X is Cauchy distributed.

Conclusion. The only distribution for which the independent-case Monte Carlo simulations lead to correct estimate of the interval uncertainty is the Cauchy distribution.

Chapter 5

How General Can We Go: What Is Computable and What Is Not

5.1 Formulation of the Problem

Need to take uncertainty into account when processing data: reminder. The resulting estimates are never 100% accurate:

- measurements are never absolutely accurate,
- physical models used for predictions are usually only approximate, and
- sometimes (like in quantum physics) these models only predict the probabilities of different events.

It is desirable to take this uncertainty into account when processing data.

In some cases, we know all the related probabilities; in this case, we can potentially determine the values of all statistical characteristics of interest: mean, standard deviation, correlations, etc.

Need to combine probabilistic and interval uncertainty. In most practical situations, however, we only have partial information about the corresponding probabilities. For example, for measurement uncertainties, often, the only information that we have about this uncertainty is the upper bound Δ on its absolute value; in this case, after we get a measurement result $\widetilde{X}$, the only information that we have about the actual (unknown) value of the corresponding quantity X is that it belongs to the interval $[\widetilde{X} - \Delta, \widetilde{X} + \Delta]$. We may know intervals containing the actual (unknown) cumulative distribution function, we may know bounds on moments, etc. In

such situations of partial knowledge, for each statistical characteristic of interest, we can have several possible values. In such cases, we are interested in the interval of possible values of this characteristic, i.e., in the smallest and the largest possible values of this characteristic. In some cases, there are efficient algorithms for computing these intervals, in other cases, the corresponding general problem is known to be NP-hard or even not algorithmically computable; see, e.g., [15].

Studying computability – just like studying NP-hardness – is important, since it prevents us from vain attempts to solve the problem in too much generality, and helps us concentrate on doable cases. In view of this importance, in this chapter, we describe the most general related problems which are still algorithmically solvable.

Comment. The results from this chapter first appeared in [16].

5.2 What Is Computable: A Brief Reminder

What is computable: general idea. We are interested in processing uncertainty, i.e., in dealing with a difference between the exact models of physical reality and our approximate representation of this reality. In other words, we are interested in models of physical reality.

Why do we need mathematical models in the first place? One of our main objectives is to predict the results of different actions (or the result of not performing any action). Models enable us to predict these results without the need to actually perform these actions, thus often drastically decreasing potential costs. For example, it is theoretically possible to determine the stability limits of an airplane by applying different stresses to several copies of this airplane until each copy breaks, but, if we have an adequate computer-based model, it is cheaper and faster to simulate different stresses on this model without having to destroy actual airplane frames.

From this viewpoint, a model is computable if it has algorithms that allow us to make the corresponding predictions. Let us recall how this general idea can be applied to different mathematical objects.

What is computable: case of real numbers. In modeling, real numbers usually represent values of physical quantities. This is what real numbers were originally invented for – to describe quantities like length, weight, etc., this is still one of the main practical applications of real

numbers.

The simplest thing that we can do with a physical quantity is measure its value. In line with the above general idea, we can say that a real number is computable if we can predict the results of measuring the corresponding quantity.

A measurement is practically never absolutely accurate, it only produces an approximation $\widetilde{x}$ to the actual (unknown) value x; see, e.g., [25]. In modern computer-based measuring instruments, such an approximate value $\widetilde{x}$ is usually a binary fraction, i.e., a rational number.

For every measuring instrument, we usually know the upper bound Δ on the absolute value of the corresponding measurement error $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$: $|\Delta x| \le \Delta$. Indeed, without such a bound, the difference Δx could be arbitrary large, and so, we would not be able to make any conclusion about the actual value x; in other words, this would be a wild guess, not a measurement.

Once we know Δ, then, based on the measurement result $\widetilde{x}$, we can conclude that the actual value x is Δ-close to $\widetilde{x}$: $|x - \widetilde{x}| \le \Delta$. Thus, it is reasonable to say that a real number x is computable if for every given accuracy $\Delta > 0$, we can efficiently generate a rational number that approximates x with the given accuracy.

One can easily see that it is sufficient to be able to approximate x with the accuracy 2^{-k} corresponding to k binary digits. Thus, we arrive at the following definition of a computable real number (see, e.g., [32]):

Definition 5.2.1. *A real number x is called* computable *if there is an algorithm that, given a natural number k, generates a rational number r_k for which*

$$|x - r_k| \le 2^{-k}. \tag{5.2.1}$$

Comment. It is worth mentioning that not all real numbers are computable. The proof of this fact is straightforward.

Indeed, to every computable real number, there corresponds an algorithm, and different real numbers require different algorithms. An algorithm is a finite sequence of symbols. There are countably many finite sequences of symbols, so there are no more than countably many computable real numbers. And it is known that the set of all real numbers is *not* computable: this was one of the first results of set theory. Thus, there are real numbers which are not computable.

How to store a computable number in the computer. The above definition provides a straightforward way of storing a computable real number in the actual computer: namely, once we fix the accuracy 2^{-k}, all we need to store in the corresponding rational number r_k.

What is computable: case of functions from reals to reals. In the real world, there are many dependencies between the values of different quantities. Sometimes, the corresponding dependence is *functional*, in the sense that the values $x_1, \ldots, x_n$ of some quantities x_i uniquely determine the value of some other quantity y. For example, according to the Ohm's Law $V = I \cdot R$, the voltage V is uniquely determined by the values of the current I and the resistance R.

It is reasonable to say that the corresponding function $y = f(x_1, \ldots, x_n)$ is computable if, based on the results of measuring the quantities x_i, we can predict the results of measuring y. We may not know beforehand how accurately we need to measure the quantities x_i to predict y with a given accuracy k. If the original accuracy of measuring x_i is not enough, the prediction scheme can ask for more accurate measurement results. In other words, the algorithm can ask, for each pair of natural numbers $i \le n$ and k, for a rational number r_{ik} such that $|x_i - r_{ik}| \le 2^{-k}$. The algorithm can ask for these values r_{ik} as many times as it needs, all we require is that at the end, we always get the desired prediction. Thus, we arrive at the following definition [32]:

Definition 5.2.2. *We say that a function $y = f(x_1, \ldots, x_n)$ from real numbers to real numbers is* computable *if there is an algorithm that, for all possible values x_i, given a natural number ℓ, computes a rational number s_ℓ for which $|f(x_1, \ldots, x_n) - s_\ell| \le 2^{-\ell}$. This algorithm,*

- *in addition to the usual computational steps,*
- *can also generate* requests*, i.e., pairs of natural numbers (i, k) with $i \le n$.*

As a reply to a request, the algorithm then gets a rational number r_{ik} for which $|x_i - r_{ik}| \le 2^{-k}$; this number can be used in further computations.

It is known that most usual mathematical functions are computable in this sense.

How to store a computable function in a computer. In contrast to the case of a computable real number, here, even if we know the accuracy $2^{-\ell}$ with which we need to compute the results, it is not immediately clear how we can store the corresponding function without explicitly storing the while algorithm.

To make storage easier, it is possible to take into account that in practice, for each physical quantity X_i, there are natural bounds $\underline{X}_i$ and $\overline{X}_i$: velocities are bounded by the speed of light, distances on Earth are bounded by the Earth's size, etc. Thus, for all practical purposes, it is sufficient to only consider values $x_i \in [\underline{X}_i, \overline{X}_i]$. It turns out that for such functions, the definition of a computable function can be simplified:

Proposition 5.2.1. *For every computable function $f(x_1, \ldots, x_n)$ on a rational-valued box $[\underline{X}_1, \overline{X}_1] \times \ldots \times [\underline{X}_n, \overline{X}_n]$, there exists an algorithm that, given a natural number ℓ, computes a natural number k such that if $|x_i - x_i'| \le 2^{-k}$ for all i, then*

$$|f(x_1, \ldots, x_n) - f(x_1', \ldots, x_n')| \le 2^{-\ell}. \tag{5.2.2}$$

This "ℓ to k" algorithm can be effectively constructed based on the original one.

Comment. For reader's convenience, all the proofs are placed in the special Proofs section.

Because of this result, for each ℓ, to be able to compute all the values $f(x_1, \ldots, x_n)$ with the accuracy $2^{-\ell}$, it is no longer necessary to describe the whole algorithm, it is sufficient to store finitely many rational numbers. Namely:

- We use Proposition 1 to find select a value k corresponding to the accuracy $2^{-(\ell+1)}$.
- Then, for each i, we consider a finite list of rational values

$$r_i = \underline{X}_i, \quad r_i = \underline{X}_i + 2^{-k}, \quad r_i = \underline{X}_i + 2 \cdot 2^{-k}, \ldots, r_i = \overline{X}_i. \tag{5.2.3}$$

- For each combination of such rational values, we use the original function's algorithm to compute the value $f(r_1, \ldots, r_n)$ with accuracy $2^{-(\ell+1)}$.

These are the values we store.

Based on these stored values, we can compute all the values of the function $f(x_1, \ldots, x_n)$ with the given accuracy $2^{-\ell}$. Specifically, for each combination of computable values $(x_1, \ldots, x_n)$, we can:

- compute 2^{-k}-close rational value $r_1, \ldots, r_n$, and then
- find, in the stored list, the corresponding approximation $\widetilde{y}$ to $f(r_1, \ldots, r_n)$, i.e., the value $\widetilde{y}$ for which $|f(r_1, \ldots, r_n) - \widetilde{y}| \le 2^{-(\ell+1)}$.

Let us show that this value $\widetilde{y}$ is indeed the $2^{-\ell}$-approximation to $f(x_1, \ldots, x_n)$.

Indeed, because of our choice of ℓ, from the fact that $|x_i - r_i| \le 2^{-k}$, we conclude that $|f(x_1, \ldots, x_n) - f(r_1, \ldots, r_n)| \le 2^{-(\ell+1)}$. Thus,

$$|f(x_1, \ldots, x_n) - y| \le |f(x_1, \ldots, x_n) - f(r_1, \ldots, r_n)| + |f(r_1, \ldots, r_n) - \widetilde{y}| \le$$

$$2^{-(\ell+1)} + 2^{-(\ell+1)} = 2^{-\ell}, \tag{5.2.4}$$

i.e., that the value $\widetilde{y}$ is indeed the desired $2^{-\ell}$-approximation to $f(x_1, \ldots, x_n)$.

A useful equivalent definition of a computable function. Proposition 1 allows us to use the following equivalent definition of a computable function.

Definition 5.2.2′. *We say that a function $y = f(x_1, \ldots, x_n)$ defined on a rational-valued box $[\underline{X}_1, \overline{X}_1] \times \ldots \times [\underline{X}_n, \overline{X}_n]$ is* computable *if there exist two algorithms:*

- *the first algorithm, given a natural number ℓ and rational values $r_1, \ldots, r_n$, computes a $2^{-\ell}$-approximation to $f(r_1, \ldots, r_n)$;*
- *the second algorithm, given a natural number ℓ, computes a natural number k such that if $|x_i - x_i'| \le 2^{-k}$ for all i, then*

$$|f(x_1, \ldots, x_n) - f(x_1', \ldots, x_n')| \le 2^{-\ell}. \tag{5.2.5}$$

Comment. As a corollary of Definition 5.2.2′, we conclude that every computable function is continuous. It should be mentioned, however, that not all continuous function is computable. For example, if a is a non-computable real number, then a linear function $f(x) = a \cdot x$ is clearly continuous but not computable. Indeed, if the function $f(x)$ was computable, we would be able to compute its value $f(1) = a$, and we know that the number a is not computable.

Not all usual mathematical functions are computable. According to Definition 5.2.2′, every computable function is continuous. Thus, discontinuous functions are not continuous, in particular, the following function:

Definition 5.2.3. *By a* step function*, we mean a function $f(x_1)$ for which:*

- *$f(x_1) = 0$ for $x < 0$ and*

- $f(x_1) = 1$ *for* $x_1 \geq 0$.

Corollary. *The step function* $f(x_1)$ *is not computable.*

Comment. This corollary can be proven directly, without referring to a (rather complex) proof of Proposition 5.2.1. This direct proof is also given in the Proofs section.

Consequences for representing a probability distribution: we need to go beyond computable functions. We would like to represent a general probability distribution by its cdf $F(x)$. From the purely mathematical viewpoint, this is indeed the most general representation – as opposed, e.g., to a representation that uses a probability density function, which is not defined if we have a discrete variable.

Since the cdf $F(x)$ is a function, at first glance, it may make sense to say that the cdf is computable if the corresponding function $F(x)$ is computable. For many distributions, this definition makes perfect sense: the cdfs corresponding to uniform, Gaussian, and many other distributions are indeed computable functions.

However, for the degenerate random variable which is equal to $x = 0$ with probability 1, the cdf is exactly the step-function, and we have just proven that the step-function is not computable. Thus, we need to find an alternative way to represent cdfs, beyond computable functions.

What we do in this chapter. In this chapter, we provide the corresponding general description:

- first for case when we know the exact probability distribution, and
- then for the general case, when we only have a partial information about the probability distribution.

5.3 What We Need to Compute: A Even Briefer Reminder

The ultimate goal of all data processing is to make decision. It is known that a rational decision maker maximizes the expected value of his/her utility $u(x)$; see, e.g., [5, 17, 20, 26]. Thus, we need to be able to compute the expected values of different functions $u(x)$.

There are known procedures for eliciting from the decision maker, with any given accuracy, the utility value $u(x)$ for each x [5, 17, 20, 26]. Thus, the utility function is *computable*. We therefore need to be able to compute expected values of computable functions.

Comment. Once we are able to compute the expected values $E[u(x)]$ of different computable functions, we will thus be able to compute other statistical characteristic such as variance. Indeed, variance V can be computed as $V = E[x^2] - (E[x])^2$.

5.4 Simplest Case: A Single Random Variable

Description of the case. Let us start with the simplest case of a single random variable X. We would like to understand in what sense its cdf $F(x)$ is computable.

According to our general application-based approach to computability, this means that we would like to find out what we can compute about this random variable based on the observations.

What can we compute about $F(x)$? By definition, each value $F(x)$ is the *probability* that $X \le x$. So, in order to decide what we can compute about the value $F(x)$, let us recall what we can compute about probabilities in general.

What can we compute about probabilities: case of an easy-to-check event. Let us first consider the simplest situation, when we consider a probability of an easy-to-check event, i.e., an event for which, from each observation, we can tell whether this event occurred or not. Such events – like observing head when tossing a coin or getting a total of seven points when throwing two dice – are what probability textbooks start with.

In general, we cannot empirically find the exact probabilities p of such an event. Empirically, we can only estimate *frequencies* f, by observing samples of different size N. It is known that for large N, the difference $d = p - f$ between the (ideal) probability and the observed frequency is asymptotically normal, with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\frac{p \cdot (1-p)}{N}}$. We also know that for a normal distribution, situations when $|d - \mu| < 6\sigma$ are negligibly rare (with probability $< 10^{-8}$), so for all practical purposes, we can conclude that $|f - p| \le 6\sigma$.

If we believe that the probability of 10^{-8} is too high to ignore, we can take 7σ, 8σ, or $k_0 \cdot \sigma$

for an even larger value k_0. No matter what value k_0 we choose, for any given value $\delta > 0$, for sufficiently large N, we get $k_0 \cdot \sigma \le \delta$.

Thus, for each well-defined event and for each desired accuracy δ, we can find the frequency f for which $|f - p| \le \delta$. This is exactly the definition of a computable real number, so we can conclude that the probability of a well-defined event should be a computable real number.

What about the probability that $X \le x$? The desired cdf is the probability that $X \le x$. The corresponding event $X \le x$ is *not* easy to check, since we do not observe the actual value X, we only observe the measurement result $\widetilde{X}$ which is close to X.

In other words, after repeating the experiment N times, instead of N actual values $X_1, \ldots, X_n$, we only know approximate values $\widetilde{X}_1, \ldots, \widetilde{X}_n$ for which

$$|\widetilde{X}_i - X_i| \le \varepsilon \tag{5.4.1}$$

for some accuracy ε. Thus, instead of the "ideal" frequency $f = \mathrm{Freq}(X_i \le x)$ – which is close to the desired probability $F(x) = \mathrm{Prob}(X \le x)$ – based on the observations, we get a slightly different frequency $f = \mathrm{Freq}(\widetilde{X}_i \le x)$.

What can we say about $F(x)$ based on this frequency? Since $|\widetilde{X}_i - X_i| \le \varepsilon$, the inequality $\widetilde{X}_i \le x$ implies that $X_i \le x + \varepsilon$. Similarly, if if $X_i \le x - \varepsilon$, then we can conclude that $\widetilde{X}_i \le x$. Thus, we have:

$$\mathrm{Freq}(X_i \le x - \varepsilon) \le f = \mathrm{Freq}(\widetilde{X}_i \le x) \le \mathrm{Freq}(X_i \le x + \varepsilon). \tag{5.4.2}$$

We have already discussed that for a sufficiently large sample, frequencies are δ-close to probabilities, so we conclude that

$$\mathrm{Prob}(X \le x - \varepsilon) - \delta \le f \le \mathrm{Prob}(\widetilde{X}_i \le x) \le \mathrm{Prob}(X_i \le x + \varepsilon) + \varepsilon. \tag{5.4.3}$$

So, we arrive at the following definition.

Definition 5.4.1. *We say that a cdf $F(x)$ is* computable *if there is an algorithm that, given rational values x, $\varepsilon > 0$, and $\delta > 0$, returns a rational number f for which*

$$F(x - \varepsilon) - \delta \le f \le F(x + \varepsilon) + \delta. \tag{5.4.5}$$

How to describe a computable cdf in a computer. How can we describe a computable cdf in a computer? The above definition kinds of prompts us to store the algorithm computing f, but algorithms may take a long time to compute. It is desirable to avoid such time-consuming computations and store only the pre-computed values – at least the pre-computed values corresponding to the given accuracy.

We cannot do this by directly following the above definition, since this definition requires us to produce an appropriate f for all infinitely many possible rational values x. Let us show, however, that a simple and natural modification of this idea makes storing finitely many values possible.

Indeed, for two natural numbers k and ℓ, let us take $\varepsilon_0 = 2^{-k}$ and $\delta_0 = 2^{-\ell}$. On the interval $[\underline{T}, \overline{T}]$, we then select a grid $x_1 = \underline{T}$, $x_2 = \underline{T} + \varepsilon_0$, ... Due to Definition 5.4.1, for every point x_i from this grid, we can then find the value f_i for which

$$F(x_i - \varepsilon_0) - \delta_0 \le f_i \le F(x_i + \varepsilon_0) + \delta_0. \tag{5.4.5}$$

Let us also set up a grid 0, δ_0, $2\delta_0$, etc., on the interval $[0, 1]$ of possible values f_i, and instead of the original values f_i, let us store the closest values $\widetilde{f}_i$ from this grid.

Thus, for each pair (k, ℓ), we store a finite number of rational numbers $\widetilde{f}_i$ each of which take finite number of possible values (clearly not exceeding $1 + 1/\delta_0 = 2^\ell + 1$). Thus, for each k and ℓ, we have finitely many possible approximations of this type.

Let us show that this information is indeed sufficient to reconstruct the computable cdf, i.e., that if we have such finite-sets-of-values for all k and ℓ, then, for each rational x, $\varepsilon > 0$, and $\delta > 0$, we can algorithmically compute the value f needed in the Definition 5.4.1.

Indeed, for each ε_0 and δ_0, we can find the value x_i from the corresponding grid which is ε_0-close to x. For this x_i, we have a value $\widetilde{f}_i$ which is δ_0-close to the f_i for which

$$F(x_i - \varepsilon_0) - \delta_0 \le f_i \le F(x_i + \varepsilon_0) + \delta_0. \tag{5.4.6}$$

Thus, we have

$$F(x_i - \varepsilon_0) - 2\delta_0 \le \widetilde{f}_i \le F(x_i + \varepsilon_0) + 2\delta_0. \tag{5.4.7}$$

From $|x_i - x| \le \varepsilon_0$, we conclude that $x_i + \varepsilon_0 \le x + 2\varepsilon_0$ and $x - 2\varepsilon_0 \le x_i - \varepsilon_0$ and thus, that $F(x - 2\varepsilon_0) \le F(x_i - \varepsilon_0)$ and $F(x_i + \varepsilon_0) \le F(x + 2\varepsilon_0)$. Hence,

$$F(x - 2\varepsilon_0) - 2\delta_0 \le \widetilde{f}_i \le F(x + 2\varepsilon_0) + 2\delta_0. \tag{5.4.8}$$

So, if we take ε_0 and δ_0 for which $2\varepsilon_0 \le \varepsilon$ and $2\delta_0 \le \delta$, then we get

$$F(x-\varepsilon) \le F(x-2\varepsilon_0) - 2\delta_0 \le \widetilde{f}_i \le F(x+2\varepsilon_0) + 2\delta_0 \le F(x+\varepsilon) + \delta, \tag{5.4.9}$$

i.e., we have the desired double inequality

$$F(x-\varepsilon) - \delta \le \widetilde{f}_i \le F(x+\varepsilon) + \delta, \tag{5.4.10}$$

with $f = \widetilde{f}_i$.

Equivalent definitions. Anyone who seriously studied mathematical papers and books have probably noticed that, in addition to definitions of different notions and theorems describing properties of these notions, these papers and books often have, for many of these notions, several different but mathematically equivalent definitions. The motivation for having several definitions is easy to understand: if we have several equivalent definitions, then in each case, instead of trying to use the original definition, we can select the one which is the most convenient to use. In view of this, let us formulate several equivalent definitions of a computable cdf.

Definition 5.4.1′. *We say that a cdf $F(x)$ is* computable *if there is an algorithm that, given rational values x, $\varepsilon > 0$, and $\delta > 0$, returns a rational number f which is δ-close to $F(x')$ for some x' for which $|x' - x| \le \varepsilon$.*

Proposition 5.4.1. *Definitions* 5.4.1 *and* 5.4.1′ *are equivalent to each other.*

To get the second equivalent definition, we start with the pairs $(x_i, \widetilde{f}_i)$ that we decided to use to store the computable cdf. When $f_{i+1} - f_i > \delta$, we add intermediate pairs

$$(x_i, f_i + \delta), (x_i, f_i + 2\delta), \ldots, (x_i, f_{i+1}). \tag{5.4.11}$$

We can say that the resulting finite set of pairs is (ε, δ)-*close* to the graph

$$\{(x, y) : F(x-0) \le y \le F(x)\} \tag{5.4.12}$$

in the following sense.

Definition 5.4.2. *Let $\varepsilon > 0$ and $\delta > 0$ be two rational numbers.*

- *We say that pairs (x, y) and (x', y') are* (ε, δ)-close *if $|x - x'| \le \varepsilon$ and $|y - y'| \le \delta$.*

- *We say that the sets S and S' are* (ε, δ)-close *if:*
  - *for every $s \in S$, there is a (ε, δ)-close point $s' \in S'$;*
  - *for every $s' \in S'$, there is a (ε, δ)-close point $s \in S$.*

Comment. This definition is similar to the definition of ε-closeness in *Hausdorff metric*, where the two sets S and S' are ε-close if:

- for every $s \in S$, there is a ε-close point $s' \in S'$;
- for every $s' \in S'$, there is a ε-close point $s \in S$.

Definition 5.4.1″. *We say that a cdf $F(x)$ is* computable *if there is an algorithm that, given rational values $\varepsilon > 0$ and $\delta > 0$, produces a finite list of pairs which is* (ε, δ)-close *to the graph* $\{(x, y) : F(x - 0) \le y \le F(x)\}$.

Proposition 5.4.2. *Definition* 5.4.1″ *is equivalent to Definitions* 5.4.1 *and* 5.4.1′.

Comment. Proof of Proposition 5.4.2 is similar to the above argument that our computer representation is sufficient for describing a computable cdf.

What can be computed: a positive result for the 1-D case. We are interested in computing the expected value $E_{F(x)}[u(x)]$ for computable functions $u(x)$. For this problem, we have the following result:

Theorem 5.4.1. *There is an algorithm that:*

- *given a computable cdf $F(x)$,*
- *given a computable function $u(x)$, and*
- *given (rational) accuracy $\delta > 0$,*

computes $E_{F(x)}[u(x)]$ with accuracy δ.

5.5 What If We Only Have Partial Information about the Probability Distribution?

Need to consider mixtures of probability distributions. The above result deals with the case when we have a single probability distribution, and by observing larger and larger samples we can get a better and better understanding of the corresponding probabilities. This corresponds to the ideal situation when all sub-samples have the same statistical characteristics. In practice, this is rarely the case. What we often observe is, in effect, a mixture of several samples with slightly different probabilities. For example, if we observe measurement errors, we need to take into account that a minor change in manufacturing a measuring instrument can cause a slight difference in the resulting probability distribution of measurement errors.

In such situations, instead of a *single* probability distribution, we need to consider a *set* of possible probability distributions.

Another case when we need to consider a set of distributions is when we only have partial knowledge about the probabilities. In all such cases, we need to process sets of probability distributions. To come up with an idea of how to process such sets, let us first recall how sets are dealt with in computations. For that, we will start with the simplest case: sets of numbers (or tuples).

Computational approach to sets of numbers: reminder. In the previous sections, we considered computable numbers and computable tuples (and computable functions). A number (or a tuple) corresponds to the case when we have a complete information about the value of the corresponding quantity (quantities). In practice, we often only have *partial* information about the actual value. In this case, instead of *single* value, we have a *set* of possible values. How can we represent such sets in a computer?

At first glance, this problem is complex, since there are usually infinitely many possible numbers – e.g., all numbers from an interval, and it is not clear how to represent infinitely many number in a computer – which is only capable of storing finite number of bits.

However, a more detailed analysis shows that the situation is not that hopeless: infinite number of values only appears in the idealized case when we assume that all the measurements

are absolutely accurate and thus, produce the exact value. In practice, as we have mentioned, measurements have uncertainty and thus, with each measuring instrument, we can only distinguish between finitely many possible outcomes.

So, for each set S of possible values, for each accuracy ε, we can represent this set by a finite list S_ε of possible ε-accurate measurement results. This finite list has the following two properties:

- each value $s_i \in S_\varepsilon$ is the result of an ε-accurate measurement and is, thus, ε-close to some value $s \in S$;
- vice versa, each possible value $s \in S$ is represented by one of the possible measurement results, i.e., for each $s \in S$, there exists an ε-close value $s_i \in S_\varepsilon$.

Comment. An attentive reader may recognize that these two conditions have already been mentioned earlier – they correspond to ε-closeness of the sets S and S_ε in terms of Hausdorff metric.

Thus, we naturally arrive at the following definition.

Definition 5.5.1. *A set S is called* computable *if there is an algorithm that, given a rational number $\varepsilon > 0$, generates a finite list S_ε for which:*

- *each element $s \in S$ is ε-close to some element from this list, and*
- *each element from this list is ε-close to some element from the set S.*

Comment. In mathematics, sets which can be approximated by finite sets are known as *compact sets*. Because of this, computable sets are also known as *computable compacts*; see, e.g., [3].

So how do we describe partial information about the probability distribution. We have mentioned that for each accuracy (ε, δ), all possible probability distributions can be represented by the corresponding finite lists – e.g., if we use Definition 5.4.1″, as lists which are (ε, δ)-close to the corresponding cdf $F(x)$.

It is therefore reasonable to represent a set of probability distributions – corresponding to partial knowledge about probabilities – by finite lists of such distributions.

Definition 5.5.2. *A set* $\mathbf{S}$ *of probability distributions is called* computable *if there is an algorithm that, given rational numbers* $\varepsilon > 0$ *and* $\delta > 0$*, generates a finite list* $\mathbf{S}_{\varepsilon,\delta}$ *of computable cdfs for which:*

- *each element* $s \in \mathbf{S}$ *is* (ε, δ)*-close to some element from this list, and*
- *each element from this list is* (ε, δ)*-close to some element from the set* $\mathbf{S}$*.*

What can be computed? For the same utility function $u(x)$, different possible probability distributions lead, in general, to different expected values. In such a situation, it is desirable to find the *range* $E_{\mathbf{S}}[u(x)] = [\underline{E}_{\mathbf{S}}[u(x), \overline{E}_S[u(x)]]$ of possible values of $E_{F(x)}[u(x)]$ corresponding to all possible probability distributions $F(x) \in \mathbf{S}$:

$$\underline{E}_{\mathbf{S}}[u(x)] = \min_{F(x)\in\mathbf{S}} E_{F(x)}[u(x)]; \quad \overline{E}_{\mathbf{S}}[u(x)] = \max_{F(x)\in\mathbf{S}} E_{F(x)}[u(x)].$$

It turns out that, in general, this range is also computable:

Theorem 5.5.1. *There is an algorithm that:*

- *given a computable set* $\mathbf{S}$ *of probability distributions,*
- *given a computable function* $u(x)$*, and*
- *given (rational) accuracy* $\delta > 0$*,*

computes the endpoints of the range $E_{\mathbf{S}}[u(x)]$ *with accuracy* δ*.*

Comment. This result follows from Theorem 5.4.1 and from the known fact that there is a general algorithm for computing maximum and minimum of a computable function on a computable compact; see, e.g., [3].

5.6 What to Do in a General (Not Necessarily 1-D) Case

Need to consider a general case. What if we have a joint distribution of several variable? A random process – i.e., a distribution on the set of functions of one variable? A random field

– a probability distribution on the set of functions of several variables? A random operator? A random set?

In all these cases, we have a natural notion of a distance (metric) which is computable, so we have probability distribution on a computable metric space M.

Situations when we know the exact probability distribution: main idea. In the general case, the underlying metric space M is not always ordered, so we cannot use cdf

$$F(x) = \mathrm{Prob}(X \le x)$$

to describe the corresponding probability distribution.

However, what we observe and measure are still numbers – namely, each measurement can be described by a computable function $g : M \to \mathbb{R}$ that maps each state $m \in M$ into a real number. By performing such measurements many times, we can get the frequencies of different values of $g(x)$. Thus, we arrive at the following definition.

Definition 5.6.1. *We say that a probability distribution on a computable metric space is* computable *if there exists an algorithm, that, given:*

- *a computable real-valued function $g(x)$ on M, and*
- *rational numbers y, $\varepsilon > 0$, and $\delta > 0$,*

returns a rational number f which is ε-close to the probability $\mathrm{Prob}(g(x) \le y')$ *for some y' which is δ-close to y.*

How can we represent this information in a computer? Since M is a computable set, for every ε, there exists an *ε-net* $x_1, \ldots, x_n$ for M, i.e., a finite list of points for which, for every $x \in M$, there exists an ε-close point x_i from this list, thus

$$X = \bigcup_i B_\varepsilon(x_i), \text{ where } B_\varepsilon(x) \stackrel{\mathrm{def}}{=} \{x' : d(x, x') \le \varepsilon\}. \tag{5.6.1}$$

For each computable element x_0, by applying the algorithm from Definition 8 to a function $g(x) = d(x, x_0)$, we can compute, for each ε_0 and δ_0, a value f which is close to $\mathrm{Prob}(B_{\varepsilon'}(x_0))$ for some ε' which is δ_0-close to ε_0.

In particular, by taking $\delta_0 = 2^{-k}$ and $\varepsilon_0 = \varepsilon + 2 \cdot 2^{-k}$, we can find a value f' which is 2^{-k}-close to $\mathrm{Prob}(B_{\varepsilon'}(x_0))$ for some $\varepsilon' \in [\varepsilon + 2^{-k}, \varepsilon + 3 \cdot 2^{-k}]$. Similarly, by taking $\varepsilon_0' = \varepsilon + 5 \cdot 2^{-k}$, we can find a value f'' which is 2^{-k}-close to $\mathrm{Prob}(B_{\varepsilon''}(x_0))$ for some $\varepsilon'' \in [\varepsilon + 4 \cdot 2^{-k}, \varepsilon + 6 \cdot 2^{-k}]$.

We know that when we have $\varepsilon < \varepsilon' < \varepsilon''$ and $\varepsilon'' \to \varepsilon$, then

$$\mathrm{Prob}(B_{\varepsilon''}(x_0) - B_{\varepsilon'}(x_0)) \to 0, \tag{5.6.2}$$

so the values f' and f'' will eventually become close. Thus, by taking $k = 1, 2, \ldots$, we will eventually compute the number f_1 which is close to $\mathrm{Prob}(B_{\varepsilon'}(x_1))$ for all ε' from some interval $[\underline{\varepsilon}_1, \overline{\varepsilon}_1]$ which is close to ε (and for which $\underline{\varepsilon} > \varepsilon$).

We then:

- select f_2 which is close to $\mathrm{Prob}(B_{\varepsilon'}(x_1) \cup B_{\varepsilon'}(x_2))$ for all ε' from some interval $[\underline{\varepsilon}_2, \overline{\varepsilon}_2] \subseteq [\underline{\varepsilon}_1, \overline{\varepsilon}_1]$,
- select f_3 which is close to $\mathrm{Prob}(B_{\varepsilon'}(x_1) \cup B_{\varepsilon'}(x_2) \cup B_{\varepsilon'}(x_3))$ for all ε' from some interval $[\underline{\varepsilon}_3, \overline{\varepsilon}_3] \subseteq [\underline{\varepsilon}_2, \overline{\varepsilon}_2]$,
- etc.

At the end, we get approximations $f_i - f_{i-1}$ to probabilities of the sets

$$S_i \stackrel{\mathrm{def}}{=} B_\varepsilon(x_i) - (B_\varepsilon(x_1) \cup \ldots \cup B_\varepsilon(x_{i-1})) \tag{5.6.3}$$

for all ε from the last interval $[\underline{\varepsilon}_n, \overline{\varepsilon}_n]$.

These approximations $f_i - f_{i-1}$ form the information that we store about the probability distribution – as well as the values x_i.

What can we compute? It turns out that we can compute the expected value $E[u(x)]$ of any computable function:

Theorem 5.6.1. *There is an algorithm that:*

- *given a computable probability distribution on a computable metric space,*
- *given a computable function $u(x)$, and*
- *given (rational) accuracy $\delta > 0$,*

computes the expected value $E[u(x)]$ with accuracy δ.

What if we have a set of possible probability distributions? In the case of partial information about the probabilities, we have a set $\mathbf{S}$ of possible probability distributions.

In the computer, for any given accuracies ε and δ, each computable probability distribution is represented by the values $f_1, \ldots, f_n$. A computable set of distributions can be then defined by assuming that, for every ε and δ, instead of a single tuple $(f_1, \ldots, f_n)$, we have a *computable set* of such tuples.

In this case, similar to the 1-D situation, it is desirable to find the *range* $E_{\mathbf{S}}[u(x)] = [\underline{E}_{\mathbf{S}}[u(x)], \overline{E}_S[u(x)]]$ of possible values of $E_P[u(x)]$ corresponding to all possible probability distributions $P \in \mathbf{S}$:

$$\underline{E}_{\mathbf{S}}[u(x)] = \min_{P \in \mathbf{S}} E_{F(x)}[u(x)]; \quad \overline{E}_{\mathbf{S}}[u(x)] = \max_{P \in \mathbf{S}} E_{F(x)}[u(x)].$$

In general, this range is also computable:

Theorem 5.6.2. *There is an algorithm that:*

- *given a computable set $\mathbf{S}$ of probability distributions,*
- *given a computable function $u(x)$, and*
- *given (rational) accuracy $\delta > 0$,*

computes the endpoints of the range $E_{\mathbf{S}}[u(x)]$ with accuracy δ.

Comment. Similarly to Theorem 5.5.1, this result follows from Theorem 5.6.1 and from the known fact that there is a general algorithm for computing maximum and minimum of a computable function on a computable compact [3].

5.7 Proofs

Proof of Proposition 5.2.1.

1°. Once we can approximate a real number x with an arbitrary accuracy, we can always find, for each k, a 2^{-k}-approximation r_k of the type $\dfrac{n_k}{2^k}$ for some integer n_k.

Indeed, we can first find a rational number r_{k+1} for which $|x - r_{k+1}| \le 2^{-(k+1)}$, and then take $r_k = \dfrac{n_k}{2^k}$ where n_k is the integer which is the closest to the rational number $2^k \cdot r_{k+1}$. Indeed, for this closest integer, we have $|2^k \cdot r_{k+1} - n_k| \le 0.5$. By dividing both sides of this inequality by 2^k, we get $|r_{k+1} - r_k| = \left|r_{k+1} - \dfrac{n_k}{2^k}\right| \le 2^{-(k+1)}$, and thus, indeed,

$$|x - r_k| \le |x - r_{k+1}| + |r_{k+1} - r_k| \le 2^{-(k+1)} + 2^{-(k+1)} = 2^{-k}. \tag{5.7.1}$$

2°. Because of Part 1 of this proof, it is sufficient to consider situations in which, as a reply to all its requests (i, k), the algorithm receives the approximate value r_{ik} of the type $\dfrac{n_{ik}}{2^k}$.

3°. Let us prove, by contradiction, that for given ℓ, there exists a value $k_{\max}$ that bounds, from above, the indices k in the all the requests (i, k) that this algorithm makes when computing a $2^{-\ell}$-approximation to $f(x_1, \ldots, x_n)$ on all possible inputs.

If this statement is not true, this means that for every natural number x, there exist a tuple $x^{(k)} = (x_1^{(k)}, \ldots, x_n^{(k)})$ for which this algorithm requests an approximation of accuracy at least 2^{-k} to at least one of the values $x_i^{(k)}$.

Overall, we have infinitely many tuples corresponding to infinitely many natural numbers. As a reply to each request (i, k), we get a rational number of the type $r_{ik} = \dfrac{n_{ik}}{2^k}$. For each natural number m, let us consider the value $\dfrac{p_i}{2^m}$ which is the closest to r_{ik}. There are finitely many possible tuples $(p_1, \ldots, p_n)$, so at least one of these tuples occurs infinitely many times.

Let us select such a tuple t_1 corresponding to $m = 1$. Out of infinitely many cases when we get an approximation to this tuple, we can select, on the level $m = 2$, a tuple t_2 for which we infinitely many times request the values which are 2^{-2}-close to this tuple, etc. As a result, we get a sequence of tuples t_m for which $|t_m - t_{m+1}| \le 2^{-m} + 2^{-(m+1)}$.

This sequence of tuples converges. Let us denote its limit by $t = (t_1, \ldots, t_n)$. For this limit, for each k, the algorithm follows the same computation as the k-th tuple and thus, will request some value with accuracy $\le 2^{-k}$. Since this is true for every k, this means that this algorithm will never stop – and we assumed that our algorithm always stops. This contradiction proves that there indeed exists an upper bound $k_{\max}$.

4°. How can we actually find this $k_{\max}$? For that, let us try values $m = 1, 2, \ldots$ For each m, we apply the algorithm $f(r_1, \ldots, r_n)$ to all possible combinations of values of the type $r_i = \dfrac{p_i}{2^m}$;

in the original box, for each m, there are finitely many such tuples. For each request (i,k), we return the number of the type $\frac{n_{ik}}{2^k}$ which is the closest to t_i. When we reach the value $m = k_{\max}$, then, by definition of $k_{\max}$, this would mean that our algorithm never requires approximations which are more accurate than 2^{-m}-accurate ones.

In this case, we can then be sure that we have reached the desired value $k_{\max}$: indeed, for all possible tuples $(x_1, \ldots, x_n)$, this algorithm will never request values beyond this m-th approximation – and we have shown it for all possible combinations of such approximations. The proposition is proven.

Direct proof of the Corollary to Proposition 5.2.1. The non-computability of the step function can be easily proven by contradiction. Indeed, suppose that there exists an algorithm that computes this function. Then, for $x_1 = 0$ and $\ell = 2$, this algorithm produces a rational number s_ℓ which is 2^{-2}-close to the value $f(0) = 1$ and for which, thus, $s_\ell \geq 0.75$. This algorithm should work no matter which approximate values r_{1k} it gets – as long as these values are 2^{-k}-close to x_1. For simplicity, let us consider the case when all these approximate values are 0s: $r_{1k} = 0$.

This algorithm finishes computations in finitely many steps, during which it can only ask for the values of finitely many such approximations; let us denote the corresponding accuracies by $k_1, \ldots, k_m$, and let $K = \max(k_1, \ldots, k_m)$ be the largest of these natural numbers. In this case, all the information that this algorithm uses about the actual value x is that this value satisfies all the corresponding inequalities $|x_1 - r_{1k_j}| \leq 2^{-k_j}$, i.e., $|x_1| \leq 2^{-k_j}$. Thus, for any other value x_1' that satisfies all these inequalities, this algorithm returns the exact same value $s_\ell \geq 0.75$. In particular, this will be true for the value $x_1' = -2^{-K}$. However, for this negative value x_1', we should get $f(x_1') = 0$, and thus, the desired inequality $|f(x_1') - y_\ell| \leq 2^{-2}$ is no longer satisfied. This contradiction proves that the step function is not computable.

Proof of Proposition 5.4.1. It is easy to show that Definition 5.4.1′ implies Definition 5.4.1. Indeed, if f is δ-close to $F(x')$ for some $x' \in [x - \varepsilon, x + \varepsilon]$, i.e., if $F(x') - \delta \leq f \leq F(x') + \delta$, then, due to $x - \varepsilon \leq x' \leq x + \varepsilon$, we get $F(x - \varepsilon) \leq F(x')$ and $F(x') \leq F(x + \varepsilon)$ and thus, that

$$F(x - \varepsilon) \leq F(x') - \delta \leq f \leq F(x') + \delta \leq F(x + \varepsilon) + \delta, \tag{5.7.2}$$

i.e., the desired inequality

$$F(x - \varepsilon) \leq f \leq F(x + \varepsilon) + \delta. \tag{5.7.3}$$

Vice versa, let us show that Definition 5.4.1 implies Definition 5.4.1′. Indeed, we know that $F(x+\varepsilon) - F(x+\varepsilon/3) \to 0$ as $\varepsilon \to 0$. Indeed, this difference is the probability of X being in the set $\{X : x+\varepsilon/3 \le X \le x+\varepsilon\}$, which is a subset of the set $S_\varepsilon \stackrel{\text{def}}{=} \{X : x < X \le x+\varepsilon\}$. The sets S_ε form a nested family with an empty intersection, thus their probabilities tend to 0 and thus, the probabilities of their subsets also tend to 0.

Due to Definition 5.4.1, for each $k = 1, 2, \ldots$, we can take $\varepsilon_k = \varepsilon \cdot 2^{-k}$ and find f_k and f'_k for which

$$F(x+\varepsilon_k/3) - \delta/4 \le f_k \le F(x+(2/3)\cdot\varepsilon_k) + \delta/4 \tag{5.7.4}$$

and

$$F(x+(2/3)\cdot\varepsilon_k) - \delta/4 \le f'_k \le F(x+\varepsilon_k) + \delta/4. \tag{5.7.5}$$

From these inequalities, we conclude that

$$-\delta/2 \le f'_k - f_k \le F(x+\varepsilon_k) - F(x+\varepsilon_k/3) + \delta/2. \tag{5.7.6}$$

Since $F(x+\varepsilon_k) - F(x+\varepsilon_k/3) \to 0$ as $k \to \infty$, for sufficiently large k, we will have $F(x+\varepsilon_k) - F(x+\varepsilon_k/3) \le \delta/4$ and thus, $|f'_k - f_k| \le (3/4)\cdot\delta$. By computing the values f_k and f'_k for $k = 1, 2, \ldots$, we will eventually reach an index k for which this inequality is true. Let us show that this f_k is then δ-close to $F(x')$ for $x' = x + (2/3)\cdot\varepsilon_k$ (which is ε_k-close – and thus, ε-close – to x).

Indeed, we have

$$f_k \le F(x+(2/3)\cdot\varepsilon_k) + \delta/4 \le F(x+(2/3)\cdot\varepsilon_k) + \delta. \tag{5.7.7}$$

On the other hand, we have

$$F(x+(2/3)\cdot\varepsilon_k) - \delta/4 \le f'_k \le f_k + (3/4)\cdot\delta \tag{5.7.8}$$

and thus,

$$F(x+(2/3)\cdot\varepsilon_k) - \delta \le f_k \le F(x+(2/3)\cdot\varepsilon_k) + \delta. \tag{5.7.9}$$

The equivalence is proven.

Proof of Theorem 5.5.1. We have shown, in Proposition 5.2.1, that every computable function $u(x)$ is computably continuous, in the sense that for every $\delta_0 > 0$, we can compute $\varepsilon > 0$ for which $|x - x'| \le \varepsilon$ implies $|u(x) - u(x')| \le \delta_0$.

In particular, if we take ε corresponding to $\delta_0 = 1$, and take the ε-grid $x_1, \ldots, x_i, \ldots$, then we conclude that each value $u(x)$ is 1-close to one of the values $u(x_i)$ on this grid. So, if we compute the 1-approximations $\widetilde{u}_i$ to the values $u(x_i)$, then each value $u(x)$ is 2-close to one of these values $\widetilde{u}_i$. Thus, $\max_x |u(x)| \le U \stackrel{\text{def}}{=} \max_i \widetilde{u}_i + 2$. So, we have a computable bound $U \ge 2$ for the (absolute value) of the computable function $u(x)$.

Let us once again use computable continuity. This time, we select ε corresponding to $\delta_0 = \delta/4$, and take an x-grid $x_1, \ldots, x_i, \ldots$ with step $\varepsilon/4$. Let G be the number of points in this grid.

According to the equivalent form (Definition 5.4.1′) of the definition of computable cdf, for each of these grid points x_i, we can compute the value f_i which is
$(\delta/(4U \cdot G))$-close to $F(x_i')$ for some x_i' which is $(\varepsilon/4)$-close to x_i.

The function $u(x)$ is $(\delta/4)$-close to a piece-wise constant function $u'(x)$ which is equal to $u(x_i)$ for $x \in (x_i', x_{i+1}']$. Thus, their expected values are also $(\delta/4)$-close:

$$|E[u(x)] - E[u'(x)]| \le \delta/4. \tag{5.7.10}$$

Here, $E[u'(x)] = \sum_i u(x_i) \cdot (F(x_{i+1}') - F(x_i'))$. But $F(x_i')$ is $(\delta/(4U \cdot G))$-close to f_i and $F(x_{i+1}')$ is $(\delta/(4U \cdot G))$-close to f_{i+1}. Thus, each difference $F(x_{i+1}') - F(x_i')$ is $(\delta/(2U \cdot G))$-close to the difference $f_{i+1} - f_i$.

Since $|u(x_i)| \le U$, we conclude that each term $u(x_i) \cdot (F(x_{i+1}') - F(x_i'))$ is $(\delta/(2G))$-close to the computable term $u(x_i) \cdot (f_{i+1} - f_i)$. Thus, the sum of G such terms – which is equal to $E[u'(x)]$ – is $(\delta/2)$-close to the computable sum

$$\sum_i u(x_i) \cdot (f_{i+1} - f_i). \tag{5.7.11}$$

Since $E[u'(x)]$ is, in its turn, $(\delta/4)$-close to desired expected value $E[u(x)]$, we thus conclude that the above computable sum

$$\sum_i u(x_i) \cdot (f_{i+1} - f_i) \tag{5.7.12}$$

is indeed a δ-approximation to the desired expected value.

The theorem is proven.

Proof of Theorem 5.6.1. The proof is similar to the proof of Theorem 5.5.1: we approximate the function $u(x)$ by a $(\delta/2)$-close function $u'(x)$ which is piece-wise constant, namely, which is

equal to a constant $u_i = u(x_i)$ on each set

$$S_i = B_\varepsilon(x_i) - (B_\varepsilon(x_1) \cup \ldots \cup B_\varepsilon(x_{i-1})). \tag{5.7.13}$$

The expected value of the function $u'(x)$ is equal to $E[u'(x)] = \sum_i u_i \cdot \mathrm{Prob}(S_i)$.

The probabilities $\mathrm{Prob}(S_i)$ can be computed with any given accuracy, in particular, with accuracy $\delta/(2U \cdot n)$, thus enabling us to compute $E[u'(x)]$ with accuracy $\delta/2$.

Since the functions $u(x)$ and $u'(x)$ are $(\delta/2)$-close, their expected values are also $(\delta/2)$-close. So, a $(\delta/2)$-approximation to $E[u'(x)]$ is the desired δ-approximation to $E[u(x)]$.

The theorem is proven.

5.8 Conclusions

When processing data, it is important to take into account that data comes from measurements and is, therefore, imprecise. In some cases, we know the probabilities of different possible values of measurement error – in this case, we have a probabilistic uncertainty. In other cases, we only know the upper bounds on the measurement error; in this case, we have interval uncertainty.

In general, we have a partial information about the corresponding probabilities: e.g., instead of knowing the exact values of the cumulative distribution function (cdf) $F(x) = \mathrm{Prob}(X \le x)$, we only know bounds on these values – i.e., in other words, an interval containing such bounds. In such situations, we have a combination of probabilistic and interval uncertainty.

The ultimate goal of data processing under uncertainty is to have efficient algorithms for processing the corresponding uncertainty, algorithms which are as general as possible. To come up with such algorithms, it is reasonable to analyze which of the related problems are algorithmically solvable in the first place: e.g., is it possible to always compute the expected value of a given computable function?

In this chapter, we show that a straightforward (naive) formulation, most corresponding problems are not algorithmically solvable: for example, no algorithm can always, given the value x, compute the corresponding value $F(x)$ of the cdf.

However, we also show that if we instead formulate these problems in practice-related terms, then these problems become algorithmically solvable. For example, if we take into account that

the value x also comes from measurement and is, thus, only known with some accuracy, it no longer make sense to look for an approximation to $F(x)$; instead, it is sufficient to look for an approximation to $F(x')$ for some x' which is close to x, and it turns out that such an approximation is always computable.

Chapter 6

Conclusions and Future Work

Conclusions. In many practical application, we process measurement results and expert estimates. Measurements and expert estimates are never absolutely accurate, their result are slightly different from the actual (unknown) values of the corresponding quantities. It is therefore desirable to analyze how this measurement and estimation inaccuracy affects the results of data processing.

There exist numerous methods for estimating the accuracy of the results of data processing under different models of measurement and estimation inaccuracies: probabilistic, interval, and fuzzy. To be useful in engineering applications, these methods:

- should provide accurate estimate for the resulting uncertainty,
- should not take too much computation time,
- should be understandable to engineers, and
- should be sufficiently general to cover all kinds of uncertainty.

In this thesis, on several case studies, we show how we can achieve these four objectives:

- We show that we can get more accurate estimates by properly taking model inaccuracy into account.
- We show that we can speed up computations by processing different types of uncertainty differently.
- We show that we can make uncertainty-estimating algorithms more understandable by explaining the need for non-realistic Monte-Carlo simulations.
- We also analyze how general uncertainty-estimating algorithms can be.

Future work. In our future work, we plan to continue working in these four directions. In particular, we plan to extend our speed-up algorithms from fuzzy to probabilistic uncertainty.

Also, since one of the main reasons for estimation and data processing is to make decisions, we plan to analyze how the corresponding uncertainty affects decision making, and what is the best way to make decisions under different types of uncertainty.

We plan to apply these algorithms to practical engineering problems. In particular, we are currently analyzing the possibility of applying different uncertainty techniques to the analysis of the practically important process of pavement compaction.

References

[1] J. Aczél, *Lectures on Functional Equations and Their Applications*, Dover, New York, 2006.

[2] M. G. Averill, *Lithospheric Investigation of the Southern Rio Grande Rift*, University of Texas at El Paso, Department of Geological Sciences, PhD Dissertation, 2007.

[3] E. Bishop and D. Bridges, *Constructive Analysis*, Springer Verlag, Heidelberg, 1985.

[4] D. G. Cacuci, *Sensitivity and Uncertainty Analysis: Theory*, Chapman & Hall/CRC, Boca Raton, Florida, 2007.

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

[6] J. A. Hole, "Nonlinear high-resolution three-dimensional seismic travel time tomography", *Journal of Geophysical Research*, 192, Vol. 97, No. B5, pp. 6553–6562.

[7] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*, Springer, London, 2001.

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

[9] V. Kreinovich, "Error estimation for indirect measurements is exponentially hard", *Neural, Parallel, and Scientific Computations*, 1994, Vol. 2, No. 2, pp. 225–234.

[10] V. Kreinovich, "Relation between interval computing and soft computing", In: C. Hu, R. B. Kearfott, A. de Korvin, and V. Kreinovich (eds.), *Knowledge Processing with Interval and Soft Computing*, Springer Verlag, London, 2008, pp. 75–97.

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

[12] V. Kreinovich, J. Beck, C. Ferregut, A. Sanchez, G. R. Keller, M. Averill, and S. A. Starks, "Monte-Carlo-type techniques for processing interval uncertainty, and their potential engineering applications", *Reliable Computing*, 2007, Vol. 13, No. 1, pp. 25–69.

[13] V. Kreinovich and S. Ferson, "A new Cauchy-based black-box technique for uncertainty in risk analysis", *Reliability Engineering and Systems Safety*, 2004, Vol. 85, No. 1–3, pp. 267–279.

[14] V. Kreinovich, O. Kosheleva, A. Pownuk, and R. Romero, "How to take into account model inaccuracy when estimating the uncertainty of the result of data processing", *Proceedings of the ASME 2015 International Mechanical Engineering Congress & Exposition IMECE'2015*, Houston, Texas, November 13–19, 2015.

[15] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.

[16] V. Kreinovich, A. Pownuk, and O. Kosheleva, "Combining interval and probabilistic uncertainty: what is computable?", in: P. Pardalos, A. Zhigljavsky, and J. Zilinskas, *Advances in Stochastic and Deterministic Global Optimization*, Springer Verlag, to appear.

[17] R. D. Luce and R. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.

[18] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information", *Psychological Review*, 1956, Vol. 63, No. 2, pp. 81–97.

[19] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylvania, 2009.

[20] H. T. Nguyen, O. Kosheleva, and V. Kreinovich, "Decision making beyond Arrow's 'impossibility theorem', with the analysis of effects of collusion and mutual attraction", *International Journal of Intelligent Systems*, 2009, Vol. 24, No. 1, pp. 27–47.

[21] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, Chapman and Hall/CRC, Boca Raton, Florida, 2006.

[22] P. Pinheiro da Silva, A. Velasco, M. Ceberio, C. Servin, M. G. Averill, N. Del Rio, L. Longpré, and V. Kreinovich, "Propagation and provenance of probabilistic and interval uncertainty in cyberinfrastructure-related data processing and data fusion", In: R. L. Muhanna and R. L. Mullen (eds.), *Proceedings of the International Workshop on Reliable Engineering Computing REC'08*, Savannah, Georgia, February 20–22, 2008, pp. 199–234.

[23] A. Pownuk, "Approximate method for computing the sum of independent random variables", *Abstracts of the 17th Joint UTEP/NMSU Workshop on Mathematics, Computer Science, and Computational Sciences*, El Paso, Texas, November 7, 2015.

[24] A. Pownuk, O. Kosheleva, and V. Kreinovich, "Limitations of realistic Monte-Carlo techniques in estimating interval uncertainty", *Proceedings of the 7th International Workshop on Reliable Engineering Computing REC'2016*, Bochum, Germany, June 15–17, 2016, pp. 269–284.

[25] S. Rabinovich, *Measurement Errors and Uncertainties: Theory and Practice*, Springer Verlag, New York, 2005.

[26] H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, Massachusetts, 1970.

[27] S. K. Reed, *Cognition: Theories and application*, Wadsworth Cengage Learning, Belmont, California, 2010.

[28] A. Saltelli, K. Chan, and E. M. Scott, *Sensitivity Analysis*, Wiley, Chichester, UK, 2009.

[29] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, Chapman and Hall/CRC, Boca Raton, Florida, 2011.

[30] P. Solin, K. Segeth, and I. Dolezel, *Higher-Order Finite Element Methods*, Chapman & Hall/CRC, Boca Raton, Florida, 2003.

[31] C. D. Stylios, A. Pownuk, and V. Kreinovich, "Sometimes, it is beneficial to process different types of uncertainty separately", *Proceedings of the Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2015 and 5th World Conference on Soft Computing*, Redmond, Washington, August 17–19, 2015.

[32] K. Weihrauch, *Computable Analysis*, Springer Verlag, Berlin, 2000.

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

Curriculum Vitae

Andrew Pownuk was born on August 19, 1969 in the town of Lubliniec, Poland. He entered the Silesian Technical University (Gliwice, Poland) in the Fall of 1990, and graduated five years later with a BSc and MS degrees in Applied Mathematics and Mechanics. In 1998 he also received a professional Master's degree in Computer Science from the Silesian University of Technology, with a specialization in computer networks and databases. In 2001, he received a Ph.D. degree from the Department of Civil Engineering, Silesian Technical University. Between 1995 and 2010 he worked for the Department of Theoretical Mechanics, Silesian University of Technology as a lecturer and an assistant professor. He have been working as a lecturer and as a faculty at the Department of Mathematical Sciences at the University of Texas at El Paso since 2006. In 2014 he received his third MSc degree – in Mathematics from The University of Texas at El Paso.

Permanent address: 7244 Feather Hawk Dr.
El Paso, Texas 79912