From Interval Computations to Constraint-Related Set Computations: Towards Faster Estimation of Statistics and ODEs Under Interval, p-Box, and Fuzzy Uncertainty

Martine Ceberio[1], Vladik Kreinovich[1], Andrzej Pownuk[2], and Barnabás Bede[2]

[1] Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA
mceberio@cs.utep.edu, vladik@utep.edu
[2] Department of Mathematical Sciences, University of Texas at El Paso, El Paso, TX 79968, USA
ampownuk@utep.edu, bbede@utep.edu

Abstract. In interval computations, at each intermediate stage of the computation, we have intervals of possible values of the corresponding quantities. In our previous papers, we proposed an extension of this technique to *set computations*, where on each stage, in addition to intervals of possible values of the quantities, we also keep sets of possible values of pairs (triples, etc.). In this paper, we show that in several practical problems, such as estimating statistics (variance, correlation, etc.) and solutions to ordinary differential equations (ODEs) with given accuracy, this new formalism enables us to find estimates in feasible (polynomial) time.

1 Formulation of the Problem

Need for data processing. In many real-life situations, we are interested in the value of a physical quantity y that is difficult or impossible to measure directly. Examples of such quantities are the distance to a star and the amount of oil in a given well. Since we cannot measure y directly, a natural idea is to measure y *indirectly*. Specifically, we find some easier-to-measure quantities $x_1, \ldots, x_n$ which are related to y by a known relation $y = f(x_1, \ldots, x_n)$; this relation may be a simple functional transformation, or complex algorithm (e.g., for the amount of oil, numerical solution to a partial differential equation). Then, to estimate y, we first measure or estimate the values of the quantities $x_1, \ldots, x_n$, and then we use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of these measurements (estimations) to compute an estimate $\widetilde{y}$ for y as $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$

Computing an estimate for y based on the results of direct measurements is called *data processing*; data processing is the main reason why computers were invented in the first place, and data processing is still one of the main uses of computers as number crunching devices.

P. Melin et al. (Eds.): IFSA 2007, LNAI 4529, pp. 33–42, 2007.

Measurement uncertainty: from probabilities to intervals. Measurement are never 100% accurate, so in reality, the actual value x_i of i-th measured quantity can differ from the measurement result $\widetilde{x}_i$. Because of these *measurement errors* $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing is, in general, different from the actual value $y = f(x_1, \ldots, x_n)$ of the desired quantity y.

It is desirable to describe the error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ of the result of data processing. To do that, we must have some information about the errors of direct measurements.

What do we know about the errors Δx_i of direct measurements? First, the manufacturer of the measuring instrument must supply us with an upper bound Δ_i on the measurement error. If no such upper bound is supplied, this means that no accuracy is guaranteed, and the corresponding "measuring instrument" is practically useless. In this case, once we performed a measurement and got a measurement result $\widetilde{x}_i$, we know that the actual (unknown) value x_i of the measured quantity belongs to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$.

In many practical situations, we not only know the interval $[-\Delta_i, \Delta_i]$ of possible values of the measurement error; we also know the probability of different values Δx_i within this interval. This knowledge underlies the traditional engineering approach to estimating the error of indirect measurement, in which we assume that we know the probability distributions for measurement errors Δx_i.

In practice, we can determine the desired probabilities of different values of Δx_i by comparing the results of measuring with this instrument with the results of measuring the same quantity by a standard (much more accurate) measuring instrument. Since the standard measuring instrument is much more accurate than the one use, the difference between these two measurement results is practically equal to the measurement error; thus, the empirical distribution of this difference is close to the desired probability distribution for measurement error. There are two cases, however, when this determination is not done:

- First is the case of cutting-edge measurements, e.g., measurements in fundamental science. When we use the largest particle accelerator to measure the properties of elementary particles, there is no "standard" (much more accurate) located nearby that we can use for calibration: our accelerator is the best we have.
- The second case is the case of measurements in manufacturing. In principle, every sensor can be thoroughly calibrated, but sensor calibration is so costly – usually costing ten times more than the sensor itself – that manufacturers rarely do it.

In both cases, we have no information about the probabilities of Δx_i; the only information we have is the upper bound on the measurement error.

In this case, after we performed a measurement and got a measurement result $\widetilde{x}_i$, the only information that we have about the actual value x_i of the measured quantity is that it belongs to the interval $\mathbf{x}_i = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In such situations, the only information that we have about the (unknown) actual value of $y =$

$f(x_1,\ldots,x_n)$ is that y belongs to the range $\mathbf{y}=[\underline{y},\overline{y}]$ of the function f over the box $\mathbf{x}_1\times\ldots\times\mathbf{x}_n$:

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

The process of computing this interval range based on the input intervals $\mathbf{x}_i$ is called *interval computations*; see, e.g., [4].

Case of fuzzy uncertainty and its reduction to interval uncertainty. An expert usually describes his/her uncertainty by using words from the natural language, like "most probably, the value of the quantity is between 3 and 4". To formalize this knowledge, it is natural to use *fuzzy set theory*, a formalism specifically designed for describing this type of informal ("fuzzy") knowledge [5].

In fuzzy set theory, the expert's uncertainty about x_i is described by a fuzzy set, i.e., by a function $\mu_i(x_i)$ which assign, to each possible value x_i of the i-th quantity, the expert's degree of certainty that x_i is a possible value. A fuzzy set can also be described as a nested family of α-cuts $\mathbf{x}_i(\alpha)\stackrel{\text{def}}{=}\{x_i\,|\,\mu_i(x_i)>\alpha\}$.

Zadeh's extension principle can be used to transform the fuzzy sets for x_i into a fuzzy set for y. It is known that for continuous functions f on a bounded domain, this principle is equivalent to saying that for every α, $\mathbf{y}(\alpha)=f(\mathbf{x}_1(\alpha),\ldots,\mathbf{x}_n(\alpha))$. In other words, fuzzy data processing can be implemented as layer-by-layer interval computations.

In view of this reduction, in the following text, we will mainly concentrate on interval computations.

Outline. We start by recalling the basic techniques of interval computations and their drawbacks, then we will describe the new set computation techniques and describe a class of problems for which these techniques are efficient. Finally, we talk about how we can extend these techniques to other types of uncertainty (e.g., classes of probability distributions).

2 Interval Computations: Brief Reminder

Interval computations: main idea. Historically the first method for computing the enclosure for the range is the method which is sometimes called "straightforward" interval computations. This method is based on the fact that inside the computer, every algorithm consists of elementary operations (arithmetic operations, min, max, etc.). For each elementary operation $f(a,b)$, if we know the intervals $\mathbf{a}$ and $\mathbf{b}$ for a and b, we can compute the exact range $f(\mathbf{a},\mathbf{b})$. The corresponding formulas form the so-called *interval arithmetic*:

$$[\underline{a},\overline{a}]+[\underline{b},\overline{b}]=[\underline{a}+\underline{b},\overline{a}+\overline{b}];\quad [\underline{a},\overline{a}]-[\underline{b},\overline{b}]=[\underline{a}-\overline{b},\overline{a}-\underline{b}];$$

$$[\underline{a},\overline{a}]\cdot[\underline{b},\overline{b}]=[\min(\underline{a}\cdot\underline{b},\underline{a}\cdot\overline{b},\overline{a}\cdot\underline{b},\overline{a}\cdot\overline{b}),\max(\underline{a}\cdot\underline{b},\underline{a}\cdot\overline{b},\overline{a}\cdot\underline{b},\overline{a}\cdot\overline{b})];$$

$$1/[\underline{a},\overline{a}]=[1/\overline{a},1/\underline{a}]\text{ if }0\notin[\underline{a},\overline{a}];\quad [\underline{a},\overline{a}]/[\underline{b},\overline{b}]=[\underline{a},\overline{a}]\cdot(1/[\underline{b},\overline{b}]).$$

In straightforward interval computations, we repeat the computations forming the program f step-by-step, replacing each operation with real numbers by the corresponding operation of interval arithmetic. It is known that, as a result, we get an enclosure $\mathbf{Y} \supseteq \mathbf{y}$ for the desired range.

From main idea to actual computer implementation. Not every real number can be exactly implemented in a computer; thus, e.g., after implementing an operation of interval arithmetic, we must enclose the result $[r^-, r^+]$ in a computer-representable interval: namely, we must round-off r^- to a smaller computer-representable value $\underline{r}$, and round-off r^+ to a larger computer-representable value $\overline{r}$.

Sometimes, we get excess width. In some cases, the resulting enclosure is exact; in other cases, the enclosure has excess width. The excess width is inevitable since straightforward interval computations increase the computation time by at most a factor of 4, while computing the exact range is, in general, NP-hard [6], even for computing the population variance $V = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \overline{x})^2$, where $\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ [3]. If we get excess width, then we can use more sophisticated techniques to get a better estimate, such as centered form, bisection, etc. [4].

Reason for excess width. The main reason for excess width is that intermediate results are dependent on each other, and straightforward interval computations ignore this dependence. For example, the actual range of $f(x_1) = x_1 - x_1^2$ over $\mathbf{x}_1 = [0, 1]$ is $\mathbf{y} = [0, 0.25]$. Computing this f means that we first compute $x_2 := x_1^2$ and then subtract x_2 from x_1. According to straightforward interval computations, we compute $\mathbf{r} = [0, 1]^2 = [0, 1]$ and then $\mathbf{x}_1 - \mathbf{x}_2 = [0, 1] - [0, 1] = [-1, 1]$. This excess width comes from the fact that the formula for interval subtraction implicitly assumes that both a and b can take arbitrary values within the corresponding intervals $\mathbf{a}$ and $\mathbf{b}$, while in this case, the values of x_1 and x_2 are clearly not independent: x_2 is uniquely determined by x_1, as $x_2 = x_1^2$.

3 Constraint-Based Set Computations

Main idea. The main idea behind constraint-based set computations (see, e.g., [1]) is to remedy the above reason why interval computations lead to excess width. Specifically, at every stage of the computations, in addition to keeping the *intervals* $\mathbf{x}_i$ of possible values of all intermediate quantities x_i, we also keep several *sets*:

- sets $\mathbf{x}_{ij}$ of possible values of pairs (x_i, x_j);
- if needed, sets $\mathbf{x}_{ijk}$ of possible values of triples (x_i, x_j, x_k); etc.

In the above example, instead of just keeping two intervals $\mathbf{x}_1 = \mathbf{x}_2 = [0, 1]$, we would then also generate and keep the set $\mathbf{x}_{12} = \{(x_1, x_1^2) \,|\, x_1 \in [0, 1]\}$. Then,

the desired range is computed as the range of $x_1 - x_2$ over this set – which is exactly $[0, 0.25]$.

To the best of our knowledge, in interval computations context, the idea of representing dependence in terms of sets of possible values of tuples was first described by Shary; see, e.g., [7] and references therein.

How can we propagate this set uncertainty via arithmetic operations? Let us describe this on the example of addition, when, in the computation of f, we use two previously computed values x_i and x_j to compute a new value $x_k := x_i + x_j$. In this case, we set $\mathbf{x}_{ik} = \{(x_i, x_i + x_j) \,|\, (x_i, x_j) \in \mathbf{x}_{ij}\}$, $\mathbf{x}_{jk} = \{(x_j, x_i + x_j) \,|\, (x_i, x_j) \in \mathbf{x}_{ij}\}$, and for every $l \neq i, j$, we take

$$\mathbf{x}_{kl} = \{(x_i + x_j, x_l) \,|\, (x_i, x_j) \in \mathbf{x}_{ij}, (x_i, x_l) \in \mathbf{x}_{il}, (x_j, x_l) \in \mathbf{x}_{jl}\}.$$

From main idea to actual computer implementation. In interval computations, we cannot represent an arbitrary interval inside the computer, we need an enclosure. Similarly, we cannot represent an arbitrary set inside a computer, we need an enclosure.

To describe such enclosures, we fix the number C of granules (e.g., $C = 10$). We divide each interval $\mathbf{x}_i$ into C equal parts $\mathbf{X}_i$; thus each box $\mathbf{x}_i \times \mathbf{x}_j$ is divided into C^2 subboxes $\mathbf{X}_i \times \mathbf{X}_j$. We then describe each set $\mathbf{x}_{ij}$ by listing all subboxes $\mathbf{X}_i \times \mathbf{X}_j$ which have common elements with $\mathbf{x}_{ij}$; the union of such subboxes is an enclosure for the desired set $\mathbf{x}_{ij}$.

This implementation enables us to implement all above arithmetic operations. For example, to implement $\mathbf{x}_{ik} = \{(x_i, x_i + x_j) \,|\, (x_i, x_j) \in \mathbf{x}_{ij}\}$, we take all the subboxes $\mathbf{X}_i \times \mathbf{X}_j$ that form the set $\mathbf{x}_{ij}$; for each of these subboxes, we enclosure the corresponding set of pairs $\{(x_i, x_i + x_j) \,|\, (x_i, x_j) \in \mathbf{X}_i \times \mathbf{X}_j\}$ into a set $\mathbf{X}_i \times (\mathbf{X}_i + \mathbf{X}_j)$. This set may have non-empty intersection with several subboxes $\mathbf{X}_i \times \mathbf{X}_k$; all these subboxes are added to the computed enclosure for $\mathbf{x}_{ik}$. Once can easily see if we start with the exact range $\mathbf{x}_{ij}$, then the resulting enclosure for $\mathbf{x}_{ik}$ is an $(1/C)$-approximation to the actual set – and so when C increases, we get more and more accurate representations of the desired set.

Similarly, to find an enclosure for

$$\mathbf{x}_{kl} = \{(x_i + x_j, x_l) \,|\, (x_i, x_j) \in \mathbf{x}_{ij}, (x_i, x_l) \in \mathbf{x}_{il}, (x_j, x_l) \in \mathbf{x}_{jl}\},$$

we consider all the triples of subintervals $(\mathbf{X}_i, \mathbf{X}_j, \mathbf{X}_l)$ for which $\mathbf{X}_i \times \mathbf{X}_j \subseteq \mathbf{x}_{ij}$, $\mathbf{X}_i \times \mathbf{X}_l \subseteq \mathbf{x}_{il}$, and $\mathbf{X}_j \times \mathbf{X}_l \subseteq \mathbf{x}_{jl}$; for each such triple, we compute the box $(\mathbf{X}_i + \mathbf{X}_j) \times \mathbf{X}_l$; then, we add subboxes $\mathbf{X}_k \times \mathbf{X}_l$ which intersect with this box to the enclosure for $\mathbf{x}_{kl}$.

Limitations of this approach. The main limitation of this approach is that when we need an accuracy ε, we must use $\sim 1/\varepsilon$ granules; so, if we want to compute the result with k digits of accuracy, i.e., with accuracy $\varepsilon = 10^{-k}$, we must consider exponentially many boxes ($\sim 10^k$). In plain words, this method is only applicable when we want to know the desired quantity with a given accuracy (e.g., 10%).

Cases when this approach is applicable. In practice, there are many problems when it is sufficient to compute a quantity with a given accuracy: e.g., when we detect an outlier, we usually do not need to know the variance with a high accuracy, an accuracy of 10% is more than enough.

Let us describe the case when interval computations do not lead to the exact range, but set computations do – of course, the range is "exact" modulo accuracy of the actual computer implementations of these sets.

Example: estimating variance under interval uncertainty. Suppose that we know the intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of possible values of $x_1, \ldots, x_n$, and we need to compute the range of the variance $V = \frac{1}{n} \cdot M - \frac{1}{n^2} \cdot E^2$, where $M \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i^2$ and $E \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i$.

A natural way to to compute V is to compute the intermediate sums $M_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i^2$ and $E_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i$. We start with $M_0 = E_0 = 0$; once we know the pair (M_k, E_k), we compute $(M_{k+1}, E_{k+1}) = (M_k + x_{k+1}^2, E_k + x_{k+1})$. Since the values of M_k and E_k only depend on $x_1, \ldots, x_k$ and do not depend on x_{k+1}, we can conclude that if (M_k, E_k) is a possible value of the pair and x_{k+1} is a possible value of this variable, then $(M_k + x_{k+1}^2, E_k + x_{k+1})$ if a possible value of (M_{k+1}, E_{k+1}). So, the set $\mathbf{p}_0$ of possible values of (M_0, E_0) is the single point $(0, 0)$; once we know the set $\mathbf{p}_k$ of possible values of (M_k, E_k), we can compute $\mathbf{p}_{k+1}$ as $\{(M_k + x^2, E_k + x) \,|\, (M_k, E_k) \in \mathbf{p}_k, x \in \mathbf{x}_{k+1}\}$. For $k = n$, we will get the set $\mathbf{p}_n$ of possible values of (M, E); based on this set, we can then find the exact range of the variance $V = \frac{1}{n} \cdot M - \frac{1}{n^2} \cdot E^2$.

What C should we choose to get the results with an accuracy $\varepsilon \cdot \overline{V}$? On each step, we add the uncertainty of $1/C$; to, after n steps, we add the inaccuracy of n/C. Thus, to get the accuracy $n/C \approx \varepsilon$, we must choose $C = n/\varepsilon$.

What is the running time of the resulting algorithm? We have n steps; on each step, we need to analyze C^3 combinations of subintervals for E_k, M_k, and x_{k+1}. Thus, overall, we need $n \cdot C^3$ steps, i.e., n^4/ε^3 steps. For fixed accuracy $C \sim n$, so we need $O(n^4)$ steps – a polynomial time, and for $\varepsilon = 1/10$, the coefficient at n^4 is still 10^3 – quite feasible.

Comment. When the accuracy increases $\varepsilon = 10^{-k}$, we get an exponential increase in running time – but this is OK since, as we have mentioned, the problem of computing variance under interval uncertainty is, in general, NP-hard.

Other statistical characteristics. Similar algorithms can be presented for computing many other statistical characteristics. For example, for every integer $d > 2$, the corresponding higher-order central moment $C_d = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \overline{x})^d$ is a linear combination of d moments $M^{(j)} \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i^j$ for $j = 1, \ldots, d$; thus, to find the exact range for C_d, we can keep, for each k, the set of possible values of d-dimensional

tuples $(M_k^{(1)}, \ldots, M_k^{(d)})$, where $M_k^{(j)} \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i^j$. For these computations, we need $n \cdot C^{d+1} \sim n^{d+2}$ steps – still a polynomial time.

Another example is covariance $C = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \cdot y_i - \frac{1}{n^2} \cdot \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i$. To compute covariance, we need to keep the values of the triples (C_k, X_k, Y_k), where $C_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i \cdot y_i$, $X_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} x_i$, and $Y_k \stackrel{\text{def}}{=} \sum_{i=1}^{k} y_i$. At each step, to compute the range of

$$(C_{k+1}, X_{k+1}, Y_{k+1}) = (C_k + x_{k+1} \cdot y_{k+1}, X_k + x_{k+1}, Y_k + y_{k+1}),$$

we must consider all possible combinations of subintervals for C_k, X_k, Y_k, x_{k+1}, and y_{k+1} – to the total of C^5. Thus, we can compute covariance in time $n \cdot C^5 \sim n^6$.

Similarly, to compute correlation $\rho = C/\sqrt{V_x \cdot V_y}$, we can update, for each k, the values of $(C_k, X_k, Y_k, X_k^{(2)}, Y_k^{(2)})$, where $X_k^{(2)} = \sum_{i=1}^{k} x_i^2$ and $Y_k^{(2)} = \sum_{i=1}^{k} y_i^2$ are needed to compute the variances V_x and V_y. These computations require time $n \cdot C^7 \sim n^8$.

Systems of ordinary differential equations (ODEs) under interval uncertainty. A general system of ODEs has the form $\dot{x}_i = f_i(x_1, \ldots, x_m, t)$, $1 \le i \le m$. Interval uncertainty usually means that the exact functions f_i are unknown, we only know the expressions of f_i in terms of parameters, and we have interval bounds on these parameters.

There are two types of interval uncertainty: we may have global parameters whose values are the same for all moments t, and we may have noise-like parameters whose values may different at different moments of time – but always within given intervals. In general, we have a system of the type $\dot{x}_i = f_i(x_1, \ldots, x_m, t, a_1, \ldots, a_k, b_1(t), \ldots, b_l(t))$, where f_i is a known function, and we know the intervals $\mathbf{a}_j$ and $\mathbf{b}_j(t)$ of possible values of a_i and $b_j(t)$.

Example. For example, the case of a differential inequality when we only know the bounds $\underline{f}_i$ and $\overline{f}_i$ on f_i can be described as $\widetilde{f}_i(x_1, \ldots, x_n, t) + b_1(t) \cdot \Delta(x_1, \ldots, x_n, t)$, where $\widetilde{f}_i \stackrel{\text{def}}{=} (\underline{f}_i + \overline{f}_i)/2$, $\Delta(t) = (\overline{f}_i - \underline{f}_i)/2$, and $\mathbf{b}_1(t) = [-1, 1]$.

Solving systems of ordinary differential equations (ODEs) under interval uncertainty. For the general system of ODEs, Euler's equations take the form $x_i(t + \Delta t) = x_i(t) + \Delta t \cdot f_i(x_1(t), \ldots, x_m(t), t, a_1, \ldots, a_k, b_1(t), \ldots, b_l(t))$. Thus, if for every t, we keep the set of all possible values of a tuple $(x_1(t), \ldots, x_m(t), a_1, \ldots, a_k)$, then we can use the Euler's equations to get the exact set of possible values of this tuple at the next moment of time.

The reason for exactness is that the values $x_i(t)$ depend only on the previous values $b_j(t - \Delta t)$, $b_j(t - 2\Delta t)$, etc., and not on the current values $b_j(t)$.

To predict the values $x_i(T)$ at a moment T, we need $n = T/\Delta t$ iterations.

To update the values, we need to consider all possible combinations of $m+k+l$ variables $x_1(t), \ldots, x_m(t), a_1, \ldots, a_k, b_1(t), \ldots, b_l(t)$; so, to predict the values at moment $T = n \cdot \Delta t$ in the future for a given accuracy $\varepsilon > 0$, we need the running time $n \cdot C^{m+k+l} \sim n^{k+l+m+1}$. This is is still polynomial in n.

Other possible cases when our approach is efficient. Similar computations can be performed in other cases when we have an iterative process where a fixed finite number of variables is constantly updated.

In such problems, there is an additional factor which speeds up computations. Indeed, in the modern computers, fetching a value from the memory, in general, takes much longer than performing an arithmetic operation. To decrease this time, computers have a hierarchy of memories – from registers from which the access is the fastest, to cash memory (second fastest), etc. Thus, to take full use of the speed of modern processors, we must try our best to keep all the intermediate results in the registers. In the problems in which, at each moment of time, we can only keep (and update) a small current values of the values, we can store all these values in the registers – and thus, get very fast computations (only the input values $x_1, \ldots, x_n$ need to be fetched from slower-to-access memory locations).

Comment. The discrete version of the class of problems when we have an iterative process where a fixed finite number of variables is constantly updated is described in [8], where efficient algorithms are proposed for solving these discrete problems – such as propositional satisfiability. The use of this idea for interval computations was first described in Chapter 12 of [6].

Additional advantage of our technique: possibility to take constraints into account. Traditional formulations of the interval computation problems assume that we can have arbitrary tuples $(x_1, \ldots, x_n)$ as long as $x_i \in \mathbf{x}_i$ for all i. In practice, we may have additional constraints on x_i. For example, we may know that x_i are observations of a smoothly changing signal at consequent moments of time; in this case, we know that $|x_i - x_{i+1}| \le \varepsilon$ for some small known $\varepsilon > 0$. Such constraints are easy to take into account in our approach.

For example, if know that $\mathbf{x}_i = [-1, 1]$ for all i and we want to estimate the value of a high-frequency Fourier coefficient $f = x_1 - x_2 + x_3 - x_4 + \ldots - x_{2n}$, then usual interval computations lead to en enclosure $[-2n, 2n]$, while, for small ε, the actual range for the sum $(x_1 - x_2) + (x_3 - x_4) + \ldots$ where each of n differences is bounded by ε, is much narrower: $[-n \cdot \varepsilon, n \cdot \varepsilon]$ (and for $x_i = i \cdot \varepsilon$, these bounds are actually attained).

Computation of f means computing the values $f_k = x_1 - x_2 + \ldots + (-1)^{k+1} \cdot x_k$ for $k = 1, \ldots$ At each stage, we keep the set $\mathbf{s}_k$ of possible values of (f_k, x_k), and use this set to find

$$\mathbf{s}_{k+1} = \{(f_k + (-1)^k \cdot x_{k+1}, x_{k+1}) \,|\, (f_k, x_k) \in \mathbf{s}_k \,\&\, |x_k - x_{k+1}| \le \varepsilon\}.$$

In this approach, when computing f_{2k}, we take into account that the value x_{2k} must be ε-close to the value x_k and thus, that we only add $\le \varepsilon$. Thus, our approach leads to almost exact bounds – modulo implementation accuracy $1/C$.

In this simplified example, the problem is linear, so we could use linear programming to get the exact range, but set computations work for similar nonlinear problems as well.

4 Possible Extension to p-Boxes and Classes of Probability Distributions

Classes of probability distributions and p-boxes: a reminder. Often, in addition to the interval $\mathbf{x}_i$ of possible values of the inputs x_i, we also have partial information about the probabilities of different values $x_i \in \mathbf{x}_i$. An exact probability distribution can be described, e.g., by its cumulative distribution function $F_i(z) = \text{Prob}(x_i \le z)$. In these terms, a partial information means that instead of a single cdf, we have a *class* $\mathcal{F}$ of possible cdfs.

A practically important particular case of this partial information is when, for each z, instead of the exact value $F(z)$, we know an interval $\mathbf{F}(z) = [\underline{F}(z), \overline{F}(z)]$ of possible values of $F(z)$; such an "interval-valued" cdf is called a *probability box*, or a *p-box*, for short; see, e.g., [2].

Propagating p-box uncertainty via computations: a problem. Once we know the classes $\mathcal{F}_i$ of possible distributions for x_i, and a data processing algorithms $f(x_1, \ldots, x_n)$, we would like to know the class $\mathcal{F}$ of possible resulting distributions for $y = f(x_1, \ldots, x_n)$.

Idea. For problems like systems of ODES, it is sufficient to keep, and update, for all t, the set of possible joint distributions for the tuple $(x_1(t), \ldots, a_1, \ldots)$.

From idea to computer implementation. We would like to estimate the values with some accuracy $\varepsilon \sim 1/C$ and the probabilities with the similar accuracy $1/C$. To describe a distribution with this uncertainty, we divide both the x-range and the probability (p-) range into C granules, and then describe, for each x-granule, which p-granules are covered. Thus, we enclose this set into a finite union of p-boxes which assign, to each of x-granules, a finite union of p-granule intervals.

A general class of distributions can be enclosed in the union of such p-boxes. There are finitely many such assignments, so, for a fixed C, we get a finite number of possible elements in the enclosure.

We know how to propagate uncertainty via simple operations with a finite amount of p-boxes [2], so for ODEs we get a polynomial-time algorithm for computing the resulting p-box for y.

For p-boxes, we need further improvements to make this method practical. Formally, the above method is polynomial-time. However, it is not yet practical beyond very small values of C. Indeed, in the case of interval uncertainty, we needed C^2 or C^3 subboxes. This amount is quite feasible even for $C = 10$.

To describe a p-subbox, we need to attach one of C probability granules to each of C x-granules; these are $\sim C^C$ such attachments, so we need $\sim C^C$ subboxes. For $C = 10$, we already get an unrealistic 10^{10} increase in computation time.

Acknowledgments. Thanks to NSF grants EAR-0225670 and DMS-0532645 and Texas Dept. of Transportation grant No. 0-5453. Many thanks to anonymous referees and to Sergey P. Shary for valuable suggestions.

References

1. M. Ceberio et al., "How To Take Into Account Dependence Between the Inputs: From Interval Computations to Constraint-Related Set Computations", *Proc. 2nd Int'l Workshop on Reliable Engineering Computing*, Savannah, Georgia, February 22–24, 2006, pp. 127–154; final version in *Journal of Uncertain Systems*, 2007, Vol. 1, No. 1 (to appear).
2. S. Ferson. *RAMAS Risk Calc 4.0.* CRC Press, Boca Raton, Florida, 2002.
3. S. Ferson *et al.*, "Computing Variance for Interval Data is NP-Hard", *ACM SIGACT News*, 2002, Vol. 33, No. 2, pp. 108–118.
4. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*, Springer, London, 2001.
5. G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic: theory and applications.* Prentice Hall, Upper Saddle River, New Jersey, 1995.
6. V. Kreinovich et al., *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997.
7. S. P. Shary, "Solving tied interval linear systems", *Siberian Journal of Numerical Mathematics*, 2004, Vol. 7, No. 4, pp. 363–376 (in Russian).
8. P. Yu. Suvorov, "On the recognition of the tautological nature of propositional formulas", *J. Sov. Math.*, 1980, Vol. 14, 1556–1562.