Economics of Engineering Design under Interval (and Fuzzy) Uncertainty: Case Study of Building Design

Carlos Ferregut[1], Jan Beck[2], Araceli Sanchez[1], and Vladik Kreinovich[1]

[1] University of Texas, El Paso, TX 79968, USA, vladik@utep.edu
[2] X-L Synergy, 2000 Wyoming Ave., El Paso, TX 79903, USA, janbeck@gmail.com

Abstract. One of the main objectives of engineering design is to find a design that is the cheapest among all designs that satisfy given constraints. Most of the constraints must be satisfied under all possible values within certain ranges. Checking all possible combinations of values is often very time-consuming. In this paper, we propose a faster algorithm for checking such constraints.

1 Formulation of the Problem

General problem of engineering design. In engineering design, e.g., when designing a building, the problem is usually to find the cheapest design among all designs that satisfy given constraints. This is a problem behind the usual bidding process when the constraints (requirements) are announced beforehand, and the contract goes to the lowest bidder among those whose design satisfies the constraints.

Checking constraints is often difficult. Most design constraints require that some condition is satisfied for all possible values of parameters within given ranges. For example, a levee must withstand all Category 4 hurricanes, i.e., all the winds within a certain range. A building must remain stable under all possible distribution of loads in different rooms on different floors.

There are many possible distributions, so checking all of them is not practically possible. It is therefore desirable to come up with efficient algorithms for checking such constraints.

General mathematical description of the problem. Let us denote the number of parameters by n, the i-th parameter by x_i, and the range of possible values of the i-th parameter by $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$. For example, for the building stability, x_i is the load in the i-th room.

The desired constraint can be described as a restriction on some characteristics depending on these parameters x_i: e.g., that at every point, the strain does not exceed the critical value. Each constraint of this type can be described as $f(x_1, \ldots, x_n) \le f_0$, where the (known) function $f(x_1, \ldots, x_n)$ describes the dependence of the critical characteristic (like strain) on the values x_i, and f_0 is the threshold value of this characteristic.

The problem is: given the intervals $\mathbf{x}_i$ and the function $f(x_1, \ldots, x_n)$, we want to check whether $f(_1, \ldots, x_n) \le f_0$ for all combinations of possible values $x_i \in \mathbf{x}_i$ of the corresponding parameters.

What we do. In this paper, we describe a new faster algorithm for checking this constraint.

2 Existing Techniques for Solving The Problem

Relation to interval computations. A natural way to check the desired condition is to compute the range $\mathbf{y} = [\underline{y}, \overline{y}]$ of the function f over the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$:

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

and then to check whether the upper endpoint $\overline{y}$ of this range exceeds the threshold f_0.

The process of computing this interval range based on the input intervals $\mathbf{x}_i$ is called *interval computations*; see, e.g., [3, 4].

Nominal values. It is reasonable to take the midpoint $\widetilde{x}_i = \dfrac{\underline{x}_i + \overline{x}_i}{2}$ of the range as the nominal value, and treat the difference $\Delta x_i \stackrel{\text{def}}{=} x_i - \widetilde{x}_i$ as the deviations from the nominal value. The value of f for the nominal values will be denoted by $\widetilde{y} \stackrel{\text{def}}{=} f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

Linearized case. In this paper, we will only consider situations in which the ranges are narrow enough. If the ranges are narrow, then the terms which are quadratic (or of higher order) in Δx_i can be safely neglected. In such situations, the dependence of the desired value

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

on Δx_i can be safely assumed to be linear.

Comment. There are practical situations when the ranges are not narrow enough, and hence, quadratic terms cannot be safely neglected. In this case, the problem of error estimation for indirect measurements becomes computationally difficult (NP-hard) even when the function $f(x_1, \ldots, x_n)$ is quadratic [7, 10].

When the differences are small, we can simplify the expression for $\Delta y \stackrel{\text{def}}{=} y - \widetilde{y} = f(x_1, \ldots, x_n) - f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ if we expand the function f in Taylor series around the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and restrict ourselves only to linear terms in this expansion. As a result, we get the expression

$$\Delta y = c_1 \cdot \Delta x_1 + \ldots + c_n \cdot \Delta x_n, \qquad (1)$$

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

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

The sum (1) attains its largest possible value if each term $c_i \cdot \Delta x_i$ in this sum attains the largest possible value:

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

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

$$\Delta = |c_1| \cdot \Delta_1 + \ldots + |c_n| \cdot \Delta_n. \tag{3}$$

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

Hence, the interval of possible values of Δy is $[-\Delta, \Delta]$, with Δ defined by the formula (3), and the range of y is equal to $[\widetilde{y} - \Delta, \widetilde{y} + \Delta]$.

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

Let us describe how this problem is solved now.

Textbook case: the function f is given by its analytical expression. If the function f is given by its analytical expression, then we can simply explicitly differentiate it, and get an explicit expression for (3).

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

When this algorithm is presented in one of the standard programming languages such as Fortran or C, we can let the compute perform an explicit differentiation; for that, we can use one of the existing automatic differentiation tools (see, e.g., [1, 2]). These tools analyze the code of the program for computing $f(x_1, \ldots, x_n)$ and, as they perform their analysis, they produce the "differentiation code", i.e., a program that computes the partial derivatives c_i.

Once we know an algorithm that computes f in time T, automatic differentiation (AD) enables us to compute all partial derivatives in time $\leq 3T$, hence we can compute Δ in time $O(T + n)$.

Case of proprietary software. The software that computes the value y of the desired characteristic is often proprietary: the owners of the program f do not want to disclose its code; instead, they only allow to use f as a black box.

If we do not know the code of f, then we cannot apply AD to compute all n partial derivatives $c_i = \frac{\partial f}{\partial x_i}$.

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

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

This algorithm is called *numerical differentiation.*

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

Problem: sometimes, numerical differentiation takes too long. Very often, the program f requires a reasonable time to compute. In this case, applying the function f is the most time-consuming part of this algorithm. So, the total time that it takes us to compute Δ is (approximately) equal to the running time T for the program f multiplied by the number of times N_f that we call the program f.

For numerical differentiation, $N_f = n$ (we call f n times to compute n partial derivatives). Hence, if the program f takes a long time to compute, and n is huge, then the resulting time $T \cdot n$ (which is $\gg T + n$) may be too long. For example, for different loads in a multi-story building, we may get n in the hundreds, and T in minutes. In this case, $T \cdot n$ may take several days. This may be OK for a single design, but too long if we want to compare different designs in order to select the optimal one – i.e., the cheapest of all the designs that satisfy all the constraints.

3 A New Method Based on Cauchy Distribution

Can we use Monte-Carlo simulations in the interval setting? It is well known that for probabilistic uncertainty, Monte-Carlo simulation speeds up computations. It is desirable to use a similar technique in interval setting as well.

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

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

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

$$\rho(z) = \frac{\Delta}{\pi \cdot (z^2 + \Delta^2)}; \tag{5}$$

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

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

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

$$c = f(\widetilde{x}_1 + \delta_1, \ldots, \widetilde{x}_n + \delta_n) - f(\widetilde{x}_1, \ldots, \widetilde{x}_n) = c_1 \cdot \delta_1 + \ldots + c_n \cdot \delta_n \tag{6}$$

is Cauchy distributed with the desired parameter (3). So, repeating this experiment N times, we get N values $c^{(1)}, \ldots, c^{(N)}$ which are Cauchy distributed with the unknown parameter, and from them we can estimate Δ.

The bigger N, the better estimates we get.

There are two questions to be solved:

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

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

$$\delta_i = \Delta_i \cdot \tan(\pi \cdot (r_i - 0.5)), \tag{7}$$

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

In order to estimate Δ, we can apply the Maximum Likelihood Method

$$\rho(d^1) \cdot \rho(d^2) \cdot \ldots \cdot \rho(d^n) \to \max,$$

where $\rho(z)$ is a Cauchy distribution density with the unknown Δ. When we substitute the above-given formula for $\rho(z)$ and equate the derivative of the product with respect to Δ to 0 (since it is a maximum), we get an equation

$$\frac{1}{1 + \left(\frac{c^{(1)}}{\Delta}\right)^2} + \ldots + \frac{1}{1 + \left(\frac{c^{(N)}}{\Delta}\right)^2} = \frac{N}{2}. \tag{8}$$

The left-hand side of (8) is an increasing function that is equal to $0(< N/2)$ for $\mathit{\Delta} = 0$ and $> N/2$ for $\mathit{\Delta} = \max \left|c^{(k)}\right|$; therefore the solution to the equation (8) can be found by applying a bisection method to the interval $\left[0, \max \left|c^{(k)}\right|\right]$.

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

$$[\widetilde{x}_1 - \mathit{\Delta}_1, \widetilde{x}_1 + \mathit{\Delta}_1] \times \ldots \times [\widetilde{x}_n - \mathit{\Delta}_n, \widetilde{x}_n + \mathit{\Delta}_n]. \tag{9}$$

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

As a result, we arrive at the following algorithm (described, for a somewhat different problem, in [5, 6, 8, 9]):

Algorithm.

```
– Apply $f$ to the results of direct measurements: $\widetilde{y} := f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$;
– For $k = 1, 2, \ldots, N$, repeat the following:
    • use the standard random number generator to compute $n$ numbers $r_i^{(k)}$, $i = 1, 2, \ldots, n$, that are uniformly distributed on the interval $[0, 1]$;
    • compute Cauchy distributed values $c_i^{(k)} := \tan(\pi \cdot (r_i^{(k)} - 0.5))$;
    • compute the largest value of $|c_i^{(k)}|$ so that we will be able to normalize the simulated measurement errors and apply $f$ to the values that are within the box of possible values: $K := \max_i |c_i^{(k)}|$;
    • compute the simulated measurement errors $\delta_i^{(k)} := \mathit{\Delta}_i \cdot c_i^{(k)}/K$;
    • compute the simulated measurement results $x_i^{(k)} := \widetilde{x}_i + \delta_i^{(k)}$;
    • apply the program $f$ to the simulated measurement results and compute the simulated error of the indirect measurement:
          $c^{(k)} := K \cdot \left(f\left(x_1^{(k)}, \ldots, x_n^{(k)}\right) - \widetilde{y}\right)$;
– Compute $\mathit{\Delta}$ by applying the bisection method to solve the equation (8).
```

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

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

For Cauchy probability density $\rho(z)$, we have $I = 1/(2\mathit{\Delta}^2)$, so the error of the above randomized algorithm is asymptotically normally distributed, with a standard deviation $\sigma_e \sim \mathit{\Delta} \cdot \sqrt{2/N}$. Thus, if we use a "two sigma" bound, we conclude that with probability 95%, this algorithm leads to an estimate for $\mathit{\Delta}$

which differs from the actual value of Δ by $\le 2\sigma_e = 2\Delta \cdot \sqrt{2/N}$. So, if we want to achieve a 20% accuracy in the error estimation, we must use the smallest N for which $2\sigma_e = 2\Delta \cdot \sqrt{2/N} \le 0.2 \cdot \Delta$, i.e., to select $N_f = N = 200$.

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

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

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

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

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

Remark: the problem of non-linearity. In the above text, we assumed that the intervals $\mathbf{x}_i$ are narrow. In this case, terms quadratic in Δx_i are negligible, and so, we can safely assume that the desired function $f(x_1, \ldots, x_n)$ is linear on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$. In practice, some intervals $\mathbf{x}_i$ may be wide, so even when restricted to the box, the function $f(x_1, \ldots, x_n)$ is non-linear. What can we do in this case?

Usually, only a few ranges are wide. For each of the corresponding variables, we can *bisect* the corresponding interval $[\underline{x}_i, \overline{x}_i]$ into two smaller subintervals – for which the dependence is approximately linear. Then, we estimate the range of the function f separately on each of the resulting subboxes, and take the union of these two ranges as the range over the entire box.

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

This bisection idea has been successfully used in interval computations; see, e.g., [3, 4].

4 Conclusions

In many engineering problems, it is desirable to find the cheapest design among all designs that satisfy given constraints $f(x_1, \ldots, x_n) \le f_0$. The constraints must be satisfied for all possible values of several parameters $x_1, \ldots, x_n$, and usually, we only know ranges of possible values $[\underline{x}_i, \overline{x}_i]$ for these parameters. Thus, to check whether the constraint is satisfied, we must check whether the upper endpoint $\overline{y}$ of the range $[\underline{y}, \overline{y}]$ of the function $f(x_1, \ldots, x_n)$ over $x_i \in\in [\underline{x}_i, \overline{x}_i]$ satisfies the desired inequality $\overline{y} \le f_0$.

When we know the code for f, then we can use automatic differentiation (AD) and compute the range of f in time $O(T+n)$. If the owner of the program f only allows to use it as a black box, then we cannot use AD any more. In principle, we can compute each of n derivatives $\partial f/\partial x_i$ by numerical differentiation, but this would require computation time $T \cdot n \gg T + n$.

We have shown that we can also compute $\varDelta$ in time $O(T)$ by using an artificial Monte-Carlo simulations in which each $\varDelta x_i$ is Cauchy distributed with parameter $\varDelta_i$ – then simulated $\varDelta y$ is Cauchy distributed with the desired parameter $\varDelta = \dfrac{\overline{y} - \underline{y}}{2}$.

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCC5-209, NSF grants EAR-0225670 and DMS-0532645, Star Award from the University of Texas System, and Texas Department of Transportation grant No. 0-5453.

References

1. Berz, M., Bischof, C., Corliss, G., Griewank, A.: *Computational Differentiation: Techniques, Applications, and Tools*, SIAM, Philadelphia, 1996.
2. Griewank, A.: *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Philadelphia, 2000.
3. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: *Applied Interval Analysis*, Springer Verlag, London, 2001.
4. Kearfott, R. B., Kreinovich, V., Eds.: *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.
5. Kreinovich, V., Bernat, A., Villa, E., Mariscal, Y.: Parallel computers estimate errors caused by imprecise data. Interval Computations No. 2 (1991) 21–46.
6. Kreinovich, V., Ferson, S.: A New Cauchy-Based Black-Box Technique for Uncertainty in Risk Analysis. Reliability Engineering and Systems Safety **85** (2004) 267–279.
7. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.
8. Kreinovich, V., Pavlovich, M. I.: Error estimate of the result of indirect measurements by using a calculational experiment. Measurement Techniques **28** (1985) 201–205.
9. Trejo, R., Kreinovich, V.: Error estimations for indirect measurements: randomized vs. deterministic algorithms for 'black-box' programs. In: Rajasekaran, S., Pardalos, P., Reif, J., Rolim, J., Eds.: *Handbook on Randomized Computing*, Kluwer, 2001, pp. 673–729.
10. Vavasis, S. A.: *Nonlinear Optimization: Complexity Issues*, Oxford University Press, New York, 1991.