A simplified version of the tomography problem can help to estimate the errors of indirect measurements

V. Kreinovich

Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968, USA

ABSTRACT

In many real-life situations, it is very difficult or even impossible to directly measure the quantity y in which we are interested: e.g., we cannot directly measure a distance to a distant galaxy or the amount of oil in a given well. Since we cannot measure such quantities *directly*, we can measure them *indirectly*: by first measuring some relating quantities $x_1, \ldots, x_n$, and then by using the known relation between x_i and y to reconstruct the value of the desired quantity y.

In practice, it is often very important to estimate the error of the resulting indirect measurement. In this paper, we show that in a natural statistical setting, the problem of estimating the error of indirect measurement can be formulated as a simplified (finite-dimensional) version of a tomography problem (reconstructing an image from its projections).

In this paper, we use the ideas of invariance (natural in statistical setting) to find the optimal algorithm for solving this simplified tomography problem, and thus, for solving the statistical problem of error estimation for indirect measurements.

Keywords: Tomography, error estimation, indirect measurement

1. ERROR ESTIMATION FOR INDIRECT MEASUREMENTS – A STATISTICAL COMPUTATIONAL PROBLEM: FORMULATION OF THE PROBLEM AND KNOWN METHODS OF SOLVING IT

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

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

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

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

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

Possible information available for estimating the error of indirect measurements. First, we know the function $f(x_1, \ldots, x_n)$. This function may be given as an analytical expression, or, more frequently, as an algorithm.

E-mail vladik@cs.utep.edu

It may be a program written in a high-level programming language (i.e., a *source code*), which can be translated into an executable file ready for computations, or it may be only an executable file, with no source code provided.

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

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

The knowledge of these probabilities is desirable but not always required and not always possible. In many practical cases, we only know the upper bounds Δ_i for the possible measurement errors, i.e., we only know that $|\Delta x_i| \le \Delta_i$. In such cases, after each direct measurement, the only information that have about the actual value x_i of the measured quantity is that this value belongs to the *interval* $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In such situations, the problem of estimating the error of indirect measurement is called the problem of *interval computations*; for details and examples of practical applications, see, e.g., Refs. 3,7.

In this paper, we will only consider the situations in which the probabilities are known.

In this paper, we will only consider the major case when errors are normally distributed and the measurements are reasonably accurate. In this paper, we will only consider the situations in which the measurement error Δx_i of each direct measurement is normally distributed with 0 average and known standard deviation σ_i, and measurement errors of different direct measurements are independent random variables. This restriction can be justified by two related reasons:

- The first reason is more theoretical. Usually, the manufacturers of the measuring instruments have made their best effort to eliminate the major sources of measurement error. The resulting measurement error comes from a variety of small independent error sources, and thus, can be described as a sum of a large number of small independent random variables. According to the central limit theorem, such a sum, under reasonable conditions, converges to normal distribution. Thus, if there are sufficiently many small random components, the resulting error distribution is indeed close to normal.
- The second reason for choosing Gaussian distribution is empirical: for the majority of measuring instrument, the measurement error is indeed normally distributed.

We will also be only considering the situations in which the direct measurements are accurate enough, so that the resulting measurement errors Δx_i are small, and terms which are quadratic (or of higher order) in Δx_i can be safely neglected. This is a reasonable assumption, because, e.g., for an error of 1% its square is a negligible 0.01%.

With these restrictions in place, we can easily deduce the explicit expression for the probability distribution of the error Δy of indirect measurement.

Probability distribution of the indirect measurement error: derivation and the resulting formula. Due to the accuracy requirement, we can simplify the expression for $\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_n) - f(x_1, \ldots, x_n)$ if we expand the function f in Taylor series around the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ and restrict ourselves only to linear terms in this expansion. As a result, we get the expression $\Delta y = c_1 \cdot \Delta x_1 + \ldots + c_n \cdot \Delta x_n$, where by c_i, we denoted the value of the partial derivative $\partial f / \partial x_i$ at the point $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$:

$$c_i = \frac{\partial f}{\partial x_i}_{|(\widetilde{x}_1, \ldots, \widetilde{x}_n)}. \tag{1}$$

So, the desired measurement error Δy is a linear combination of independent Gaussian variables, and hence, Δy is also normally distributed, with 0 average and the standard deviation

$$\sigma = \sqrt{c_1^2 \cdot \sigma_1^2 + \ldots + c_n^2 \cdot \sigma_n^2}. \tag{2}$$

Error estimation for indirect measurement: a precise computational formulation of the problem. As a result of the above analysis, we get the following explicit formulation of the problem: given a function $f(x_1, \ldots, x_n)$, n numbers $\widetilde{x}_1, \ldots, \widetilde{x}_n$, and n positive numbers $\sigma_1, \ldots, \sigma_n$, compute the expression (2).

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 (2). This is the case which is typically analyzed in textbooks on measurement theory (see, e.g., Refs. 2,6).

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

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

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

In this paper, we will analyze such black-box situations, and describe the *optimal* black-box algorithm for computing σ. Before we describe the optimal algorithm, we must do two things: formulate the black-box approach in mathematical terms, and describe known black-box methods.

Error estimation for indirect measurements: towards a mathematical reformulated of the black-box approach. The general idea of a black-box method is that, in addition to computing the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of the indirect measurement, we also apply the function f to other inputs which are different from $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

Our ultimate goal is to analyze how errors in x_i induce errors in the value of the function f. We have already assumed that the errors in x_i are small, so the vector $(x_1, \ldots, x_n)$ of the actual values is close to the measured values $(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ (so close that the squares of measurement errors $\Delta x_i = \widetilde{x}_i - x_i$ can be neglected). From the viewpoint of this goal, we are only interested in the values of the function f in the small vicinity of the measurement point, i.e., for the values $x_i = \widetilde{x}_i + \delta_i$ for some small δ_i (so small that their squares are negligible). Since quadratic terms are negligible, we can expand f in Taylor series and only keep linear terms in this expansion. As a result, when we apply the function f to the values $x_i + \delta_i$, we get the result

$$f(\widetilde{x}_1 + \delta_1, \ldots, \widetilde{x}_n + \delta_n) = \widetilde{y} + c_1 \cdot \delta_1 + \ldots + c_n \cdot \delta_n. \tag{3}$$

So, in effect, we get the *dot* (*scalar*) product between the input vector $\vec{\delta} = (\delta_1, \ldots, \delta_n)$ and the (unknown) vector $\vec{c} = (c_1, \ldots, c_n)$. Let us explicitly repeat the resulting mathematical reformulation.

Error estimation for indirect measurements: towards a mathematical reformulated of the black-box approach. We know the values $\widetilde{y}$, $\sigma_1, \ldots, \sigma_n$; we know that there is a vector $(c_1, \ldots, c_n)$ but we do not know the values of its components. For any given vector $(\delta_1, \ldots, \delta_n)$, we can compute the dot product $c_1 \cdot \delta_1 + \ldots + c_n \cdot \delta_n$. Based on the results of these computations, we must compute the value (2).

A straightforward method of solving this problem: numerical differentiation. The most straightforward method of solving this problem is to compute the derivatives c_i one-by-one, and then use the formula (2) to compute the desired σ. To compute i-th partial derivative, we change i-th input x_i, and leave other inputs unchanged. We want the change to be small (so that quadratic terms can be neglected); we already know that changes of the order σ_i are small. So, it is natural to take $\delta_i = \sigma_i$. In other words, to compute c_i, we use the following values: $\delta_1 = \delta_{i-1} = 0$, $\delta_i = \sigma_i$, $\delta_{i+1} = \ldots = \delta_n = 0$. This method is called *numerical differentiation*.

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

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

If we do not have enough time for numerical differentiation, we may use statistical methods. The only way to save on computation time is to limit the number of times we call f, i.e., to limit N_f.

If we limit this number of calls to $N_f < n$, we cannot get an answer which is always correct: Indeed, if applied $N_f < n$ different vectors $\vec{\delta} = (\delta_1, \ldots, \delta_n)$ and computed the values $\vec{\delta} \cdot \vec{c}$, then we get $\leq n - 1$ equations for n variables $c_1, \ldots, c_n$; therefore, this system does not have a single solution; it has an at least 1-D line of possible solutions $\vec{c}$, and for points of this line which go to ∞, the expression (2) tends to ∞ as well. Thus, we can have arbitrarily large values of σ which are consistent with the results of black-box testing, and we cannot, therefore, deduce the exact value of σ from these testing results.

Since we cannot use *deterministic* methods, i.e., methods which always guarantee an answer, we can try *statistical* methods, which provide us with an answer with a certain probability. In real-life applications, the fact that these methods have a (small) probability of erring is OK, because the measuring instruments themselves have a small probability of failing anyway.

Statistical methods can indeed speed up computations: e.g., a straightforward simulation (Monte-Carlo type) saves time drastically:

Monte-Carlo simulation. In this method, we use a computer-based random number generator to simulate the normally distributed error. A standard normal random number generator usually produces a normal distribution with 0 average and standard deviation 1. So, to simulate a distribution with a standard deviation σ_i, we multiply the result α_i of the standard random number generator by σ_i. In other words, we take $\delta_i = \sigma_i \cdot \alpha_i$.

As a result of N Monte-Carlo simulations, we get N values $c^{(1)} = \vec{c} \cdot \vec{\delta}^{(1)}, \ldots, c^{(N)} = \vec{c} \cdot \vec{\delta}^{(N)}$ which are normally distributed with the desired standard deviation σ. So, we can determine σ by using the standard statistical estimate $\sigma = \sqrt{\sum_{k=1}^{N} (c^{(k)})^2 / (N-1)}$. The relative error of this estimate depends only on N (as $\approx 1/\sqrt{N}$), and not on the number of variables n. Therefore, the number of steps N_f needed to achieve a given accuracy does not depend on the number of variables at all. E.g., if we want 20% error in computing σ, it is sufficient to have $N_f \approx 25$. For $n \approx 10^3$, this is much less than $N_f = n$ required for numerical differentiation.

What we are planning to do. In this paper, we will find the *best* statistical method of estimating σ. (This method is somewhat better than the Monte-Carlo method.) To find this best methods, we will reformulate our problem as a tomography problem, and then use geometric intuition and ideas to solve this tomography problem.

Our main idea and the preliminary (restricted) version of our result were first announced in.[4]

2. ERROR ESTIMATION FOR INDIRECT MEASUREMENT REFORMULATED AS A TOMOGRAPHY PROBLEM

When $\sigma_1 = \ldots = \sigma_n = 1$, the error estimation problem has a natural geometric tomographic interpretation. When $\sigma_1 = \ldots = \sigma_n = 1$, then the formula (2) becomes a formula for the length of a vector $\vec{c} = (c_1, \ldots, c_n)$. Therefore, the above error estimation problem for indirect measurements can be reformulated in the following geometric terms: We have an (unknown) vector $\vec{c}$. We want to know its length $c = \|\vec{c}\|$. In order to compute this length, we can pick an arbitrary vector $\vec{\delta}$ and compute a scalar product $\vec{c} \cdot \vec{\delta}$. The question is: how to estimate the desired length by picking the smallest possible number of vector. Alternatively, if the number N of picked vectors is fixed, the question is: How to select these vectors $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}$ in such a way that from the scalar products $c^{(1)}, \ldots, c^{(N)}$, we will able to get the best possible estimate $\widetilde{c}$ for the desired length c.

From the mathematical viewpoint, knowing the scalar product $\vec{c} \cdot \vec{\delta}$ of the unknown vector $\vec{c}$ with a known vector $\vec{\delta}$ is equivalent to knowing the *projection* $\pi_{\vec{\delta}}(\vec{c}) = (\vec{c} \cdot \vec{\delta}) / \|\vec{\delta}\|$. Therefore, the above problem means that we need to reconstruct the length of a vector from its projections to different directions.

In general, problems in which we want to reconstruct a certain property of an unknown object based on the characteristics of its projections are called *tomography problems*. Therefore, the above problem can be viewed as (simple) case of tomography problems.

The general case can be naturally reduced to the (geometrically interpretable) case when $\sigma_1 = \ldots = \sigma_n = 1$. Let us show that the general case of the error estimation problem for indirect measurements can be naturally reduced to the case of $\sigma_i = 1$. Indeed, in the general case, we must find the expression (2) for unknown values $c_1, \ldots, c_n$. Instead of considering the values c_i as unknowns, we can introduce new unknowns: $c'_i = c_i \cdot \sigma_i$. Then, the desired expression (2) is simply the length $\sigma = \|\vec{c'}\|$ of the new unknown vector $\vec{c'} = (c'_1, \ldots, c'_n)$.

To complete the reduction to the case of $\sigma_i = 1$, we need to reformulate the expression for the observed quantity $\sum \delta_i \cdot c_i$ in terms of the new variables $c'_i = c_i \cdot \sigma_i$. Such a reformulation is straightforward: $\sum \delta_i \cdot c_i = \sum \delta'_i \cdot c'_i$, where $\delta'_i = \delta_i / \sigma_i$. If we know δ'_i, then we can reconstruct δ_i as $\delta_i = \delta'_i \cdot \sigma_i$.

Therefore, if we know how to solve the geometrizable problem (with $\sigma_i = 1$), we can then solve the problem with arbitrary σ_i as follows: first, we generate a vector $\vec{\delta'}$ which is appropriate for the geometrized problem, and then use the values $\delta_i = \delta'_i \cdot \sigma_i$ to solve the given problem.

This reduction is very natural, because in both above methods – numerical differentiation and Monte-Carlo – the values δ_i were actually of the form $\delta_i = \sigma_i \cdot \alpha_i$ for some values α_i which did not depend on σ_i.

Precise formulation of the corresponding geometric problem. In view of the above reduction, it is sufficient to solve the geometrizable particular case of the error estimation problem. Let us formulate this problem in precise mathematical terms.

Definition 1. *Let positive integers n and N be given. By an algorithm which solves the n-variable error estimation problem in N calls (or simply an (n, N)-algorithm, for short), we mean a tuple $\mathcal{U} = (p_1, \ldots, p_N, e)$, where:*

- *for every k from 1 to N, p_k is a function which assigns, to every element from $R^{n \times (k-1)} \times R^{k-1}$, a probability measures on R^n, and*
- *e is a function which assigns to every element of $R^{n \times N} \times R^N$ a probability measure on R.*

Definition 2. *Let $n, N > 0$. By a (n, N)-computational trajectory, we mean a sequence $\vec{\delta}^{(1)}, c^{(1)}, \vec{\delta}^{(2)}, c^{(2)}, \ldots, \vec{\delta}^{(N)}, c^{(N)}, \widetilde{c}$, where $\widetilde{c}$ is a real number, and for each k from 1 to N, $\vec{\delta}^{(k)} \in R^n$ is an n-dimensional vector and $c^{(k)}$ is a real number.*

Definition 3. *Let $\mathcal{U} = (p_1, \ldots, p_N, e)$ be an (n, N)-algorithm. Then, for each n-dimensional vector $\vec{c} \in R^n$, the algorithm $\mathcal{U}$ generates the following probability distribution on the set of all possible (n, N)-computational trajectories:*

- *for $k = 1$, a random vector $\vec{\delta}^{(1)}$ is generated according to the probability distribution p_1;*
- *then, we take $c^{(1)} = \vec{c} \cdot \vec{\delta}^{(1)}$;*
- *...*
- *for $k > 1$, a random vector $\vec{\delta}^{(k)}$ is generated according to the probability distribution $p_k\left(\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(k-1)}, c^{(1)}, \ldots, c^{(k-1)}\right)$;*
- *then, we take $c^{(k)} = \vec{c} \cdot \vec{\delta}^{(k)}$;*
- *...*
- *finally, a random value $\widetilde{c}$ is generated according to the probability measure $e\left(\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}, c^{(1)}, \ldots, c^{(N)}\right)$.*

Let $\Phi(\widetilde{c}, c)$ be a function which, for each c, attains its minimum for $\widetilde{c} = c$ and is convex in c. This function will be called the inaccuracy of approximating c by $\widetilde{c}$. For each vector $\vec{c} \in R^n$ of length $c = \|\vec{c}\| \neq 0$, the inaccuracy $I(\mathcal{U}, \vec{c})$ of an algorithm $\mathcal{U}$ on this vector is defined as $I(\mathcal{U}, \vec{c}) = E(\Phi(\widetilde{c}, c))$. By an inaccuracy $I(\mathcal{U})$ of an algorithm $\mathcal{U}$, we mean the largest possible value of its inaccuracy on any vector $\vec{c} \neq 0$.

Example. For example, we can take

$$\Phi(\widetilde{c}, c) = \left(\frac{\widetilde{c} - c}{c}\right)^2.$$

Definition 4. *Let $n, N > 0$. We say that an (n, N)-algorithm is optimal if has the smallest possible inaccuracy.*

Now, we are ready to formulate and to prove our main result:

3. THE MAIN RESULT: OPTIMAL METHOD OF ERROR ESTIMATION FOR INDIRECT MEASUREMENTS

3.1. The main result (in mathematical form)

In this section, we will explicitly describe, for each n and N and for each inaccuracy criterion, the optimal (n, N)-algorithm.

Theorem 1. *When $N \geq n$, numerical differentiation is the optimal (n, N)-algorithm. When $N < n$, the following (n, N)-algorithm $\mathcal{U} = (p_1, \ldots, p_k, e)$ is optimal:*

- *p_1 is a uniform distribution on a unit sphere in R^n;*
- *for each $k > 1$, $p_k(\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(k-1)}, c^{(1)}, \ldots, c^{(k-1)})$ is a uniform distribution on the set of all unit vectors of R^n which are orthogonal to $(k-1)$ vectors $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(k-1)}$ (this set is an $(n-k)$-dimensional sphere);*
- *finally, the probability distribution $e(\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}, c^{(1)}, \ldots, c^{(N)})$ is concentrated, with probability 1, on a real number*
$$\widetilde{c} = g\left(\sum_{k=1}^{N} (c^{(k)})^2\right), \tag{4}$$
where $g : R \to R$ is a function of one variable which depends on the criterion Φ.

Comments and examples. As we will see from the proof, a similar result holds if instead of estimating the length c of the vector $\vec{c}$, we estimate an arbitrary *function* of this length. For example, if we want to estimate $\sigma^2 = c^2$, then it is natural to choose a criterion
$$\Phi(\widetilde{c}, c) = \left(\frac{(\widetilde{c})^2 - c^2}{c^2}\right)^2.$$
For this criterion, the optimal estimate comes from $g(z) = \sqrt{(n+2)/(N+2)} \cdot z$: that $g(z) = k \cdot z$ follows from the fact that the problem is invariant under scaling $z \to \lambda \cdot z$ for an arbitrary $\lambda > 0$, and the specific optimal choice of k comes from the explicit computations.

A similar theorem holds if instead of *arbitrary* (n, N)-algorithms, we will only consider algorithm which produce *un-biased* estimates for c or for an appropriate function of c. For example, if we are estimating $c^2 = \sigma^2$, and we want to choose an unbiased estimate which is optimal under the above criterion, then we get the estimate $(\widetilde{c})^2 = (n/N) \cdot \sum_{k=1}^{N} \left(c^{(k)}\right)^2$. Here, $g(z) = (n/N) \cdot z$.

3.2. How to actually program this optimal method

The first auxiliary algorithm: Gram-Schmidt orthogonalization. The mathematical description of this optimal method may look somewhat clumsy, but in reality, this method is easy to program. The programming is based on the well-known Gram-Schmidt orthogonalization procedure: If we have N vectors $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}$, then the following formulas produce orthogonal unit vectors $\vec{e}^{(1)}, \vec{e}^{(2)}, \ldots$ which span the same linear space as the vectors $\vec{\delta}^{(k)}$:
$$\vec{e}^{(1)} = \frac{\vec{\delta}^{(1)}}{\left\|\vec{\delta}^{(1)}\right\|}, \tag{5}$$
and then for each $k > 1$,
$$\vec{e}^{(k)} = \frac{\vec{\varphi}^{(k)}}{\left\|\vec{\varphi}^{(k)}\right\|},$$
where
$$\vec{\varphi}^{(k)} = \vec{\delta}^{(k)} - \sum_{i=1}^{k-1} \left(\vec{\delta}^{(k)} \cdot \vec{e}^{(i)}\right) \cdot \vec{e}^{(i)}. \tag{6}$$

The second auxiliary algorithm: Gaussian random number generator. The program also uses a random number generator for generating a normally distributed random variable with 0 average and unit standard deviation, so that consequent calls to this program produce independent and equally distributed random variables (some programming languages have an explicit commands for such a generator, others have known algorithms for it).

The program itself.

- First, we apply this random number generator n times, and get the n components of the first vector $\vec{a}^{(1)}$. Then, we apply this same random number generator n more times, and get n components of the second vector $\vec{a}^{(2)}$, etc.
- After we have generated N such vectors, we apply Gram-Schmidt orthogonalization to generate the vectors $\vec{e}^{(1)}, \ldots, \vec{e}^{(N)}$.
- Then, for each k from 1 to N, we apply the function f to the values $\tilde{\vec{x}} + \vec{e}^{(k)}$, thus computing $c^{(k)} = f\left(\tilde{\vec{x}} + \vec{e}^{(k)}\right) - \tilde{y}$.
- Finally, we compute the estimate $\tilde{\vec{c}}$ by using the formula (4).

3.3. Proof of optimality

The main idea of the proof. The original problem is rotation-invariant. Therefore, if we have an (n, N)-algorithm for solving this problem which is not rotation-invariant, we can design a new algorithm if we first apply a random rotation and then apply this algorithm to the rotated problem. The resulting new algorithm is rotation-invariant, and because the function (inaccuracy) that we want to minimize is assumed to be convex, additional randomization can only decrease its expected value, and thus, can only improve the algorithm. This, when we are looking for an optimal algorithm, we only need to consider rotation-invariant algorithms. We will show, step-by-step, that the only rotation-invariant algorithm is the algorithm described in Theorem 1.

Comment. For a general overview of how the general ideas of symmetry and invariance help in solving optimization problems in computer-related areas, see, e.g., Ref. 5.

When looking for an optimal algorithm, we can always assume that the vectors $\vec{\delta}^{(k)}$ produced by an algorithm form an orthonormal basis. Indeed, let us show that an arbitrary (n, N)-algorithm can be modified, without changing its inaccuracy, into a new algorithm which only deals with the orthonormal bases.

Ultimately, whenever the original algorithm produces a sequence of vectors $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}$, the new algorithm will produce the Gram-Schmidt orthogonalization $\vec{e}^{(1)}, \ldots, \vec{e}^{(N)}$ of this sequence of vectors. To be more precise, a new algorithm follows the original one, with the following modifications:

- On the 1st step, when the original algorithm produced a vector $\vec{\delta}^{(1)}$ and wants to compute the value $c^{(1)} = \vec{c} \cdot \vec{\delta}^{(1)}$, the new algorithm first produces the unit vector $\vec{e}^{(1)}$ by applying the first orthogonalization step (5), then computes a new scalar product $c^{(1)}_{\text{new}} = \vec{c} \cdot \vec{e}^{(1)}$, and then reconstructs $c^{(1)}$ as $c^{(1)} = c^{(1)}_{\text{new}} \cdot \|\vec{\delta}^{(1)}\|$.
- For each $k \geq 2$, on k-th step, when the original algorithm produced a vector $\vec{\delta}^{(k)}$, and wants to compute the value $c^{(k)} = \vec{c} \cdot \vec{\delta}^{(k)}$, the new algorithm first performs the k-th step (6) of the orthogonalization procedure, i.e., produces a vector $\vec{e}^{(k)}$, then computes $c^{(k)}_{\text{new}} = \vec{c} \cdot \vec{e}^{(k)}$, and reconstructs the desired value as $c^{(k)} = \left(\vec{\delta}^{(k)} \cdot \vec{e}^{(1)}\right) \cdot c^{(1)}_{\text{new}} + \ldots + \left(\vec{\delta}^{(k)} \cdot \vec{e}^{(k)}\right) \cdot c^{(k)}_{\text{new}}$.

  The possibility of this reconstruction follows from the fact that the vectors $\vec{e}^{(i)}$, $1 \leq i \leq k$, form an orthonormal basis in a space which includes $\vec{\delta}^{(k)}$ and therefore,

  $$\vec{\delta}^{(k)} = \sum_{i=1}^{k} \left(\vec{\delta}^{(k)} \cdot \vec{e}^{(i)}\right) \cdot \vec{e}^{(i)};$$

  multiplying both sides of this equality by $\vec{c}$ and taking into consideration that $\vec{c} \cdot \vec{e}^{(i)} = c^{(i)}_{\text{new}}$, we get the above formula.

The final result of the new algorithm is exactly the same as for the old one, so the inaccuracy of the new algorithm is the same as for the old one. On each step, we replace each call to f (i.e., each computation of a scalar product) by exactly one call (computation); thus, the total number of calls (N) remains the same. However, in the new algorithm, we only compute the scalar product for a sequence of vectors which forms an orthonormal basis.

In view of this transformation, we can, without losing generality, assume that an (n, N)-algorithm only uses orthonormal bases, i.e., that with probability 1, in each trajectory, the vectors $\vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}$ form an orthonormal basis.

A known result: there is only one rotation-invariant probability measure on the set of all bases. We want to generate random orthonormal bases. Since the problem is rotation-invariant, it is reasonable to look for rotation-invariant probability measures on the set of all such bases.

In our further proof, we will use the fact that there is only one such measure. This result is known, but, for completeness, we will describe the main idea of the proof. To describe the probability measure on the set of all N-element orthonormal bases, we must describe: the probabilities of different values of the first vector $p\left(\vec{e}^{(1)}\right)$, the conditional probabilities of different values of the second vector given the first vector $p\left(\vec{e}^{(2)} \mid \vec{e}^{(1)}\right)$, and, for all $k = 3, \ldots, N$, the conditional probabilities $p\left(\vec{e}^{(k)} \mid \vec{e}^{(1)}, \ldots, \vec{e}^{(k-1)}\right)$.

For $k = 1$, all possible vectors $\vec{e}^{(1)}$ form a unit sphere (since the bases are orthonormal). On the unit sphere, there is a only one rotation-invariant distribution: uniform distribution on this sphere, for which the probability measure is equal to the normalized Lebesgue measure on this sphere. Similarly, when the vectors $\vec{e}^{(1)}, \ldots, \vec{e}^{(k-1)}$ are fixed, then, due to orthonormality, $\vec{e}^{(k)}$ must be a unit vector orthogonal to all the previous vectors $\vec{e}^{(i)}$, $1 \le i \le k-1$. Thus, all such vectors form a unit sphere in a linear subspace of R^n, namely, on a space $\left[Lin\left(\vec{e}^{(1)}, \ldots, \vec{e}^{(k-1)}\right)\right]^{\perp}$ which is formed by all vectors orthogonal to all $k-1$ previous vectors. Again, on a unit sphere, there is only one rotation-invariant probability distribution, for which the probability measure is equal to the normalized Lebesgue measure on the corresponding linear subspace. The uniqueness is proven.

The algorithm from Section 3.2 produces a basis which is random with respect to a rotation-invariant distribution. Indeed, according to the construction from Section 3.2, each component a_i of each vector $\vec{a} = (a_1, \ldots, a_n)$ is an independent Gaussian variable with 0 average and standard deviation 1. Thus, the corresponding probability density for each component a_i is equal to const $\cdot \exp(-a_i^2/2)$, and the resulting probability density for each vector $\vec{a}$ is equal to the product of these densities: const $\cdot \exp(-a_1^2 - \ldots - a_n^2) =$ const $\cdot \exp(-\|\vec{a}\|^2/2)$. This probability density depends only on the length $\|\vec{a}\|$ of the vector $\vec{a}$, and is, therefore, rotation-invariant.

Since the distribution of each of the vectors $\vec{a}^{(k)}$ is rotation-invariant, and these vectors are, by construction, statistically independent, the resulting distribution on the set of all tuples $(\vec{a}^{(1)}, \ldots, \vec{a}^{(N)})$ is also rotation-invariant. Since Gram-Schmidt orthogonalization is also a rotation-invariant procedure, the resulting probability distribution on the set of all the bases is also rotation-invariant. The statement is proven.

When looking for an optimal algorithm, we can always assume that the vectors $\vec{\delta}^{(k)}$ form an orthonormal basis which is random with respect to a rotation-invariant distribution. Indeed, let us show that an arbitrary (n, N)-algorithm can be modified, without worsening its inaccuracy, into a new algorithm in which orthonormal bases are random with respect to the rotation-invariant distribution.

To produce this new algorithm, we will use a *random rotation* T, which can be defined as a linear transformation that turns the coordinate unit vectors $\vec{b}^{(1)} = (1, 0, \ldots, 0), \ldots, \vec{b}^{(n)} = (0, \ldots, 0, 1)$ into elements of an n-element random orthonormal basis $\vec{e}^{(1)}, \ldots, \vec{e}^{(n)}$ (and thus, due to linearity, turns every vector $\vec{x} = (x_1, \ldots, x_n)$ into a vector $T(\vec{x}) = x_1 \cdot \vec{e}^{(1)} + \ldots + x_n \cdot \vec{e}^{(n)}$).

Now that a random rotation is defined, we can describe the new (n, N)-algorithm. First, we pick a random rotation T. Then, we follow the old algorithm step-by-step. Every time the old algorithm generates a vector $\vec{\delta}^{(k)}$ for computing $c^{(k)} = \vec{c} \cdot \vec{\delta}^{(k)}$, in the new algorithm, we call f with the *new* vector $\vec{\delta}^{(k)}_{\text{new}} = T\left(\vec{\delta}^{(k)}\right)$, i.e., compute $c^{(k)}_{\text{new}} = \vec{c} \cdot T\left(\vec{\delta}^{(k)}\right)$.

Let us show that for the new algorithm, the inaccuracy can only become smaller. Indeed, since the scalar product is rotation-invariant, the new value of $c^{(k)}_{\text{new}}$, which we obtained by combining the new (rotated) vector $\vec{\delta}^{(k)}_{\text{new}}$ with the old vector $\vec{c}$, is equal to the result $T^{-1}(\vec{c}) \cdot \vec{\delta}^{(k)}$ of combining the old (unrotated) vector $\vec{\delta}^{(k)}$ with the new vector $\vec{c}_{\text{new}} = T^{-1}(\vec{c})$ (which is obtained from $\vec{c}$ by the reverse rotation).

If we fix T, the trajectory of an old algorithm for a vector $\vec{c}$ is a trajectory of the new algorithm for the rotated vector $\vec{c}_{\text{new}} = T^{-1}(\vec{c})$. Thus, when T is fixed, the new algorithm has the same inaccuracies as the old one, except that the new algorithm has them for different vectors $\vec{c}$. So, if we had fixed T, the maximum over $\vec{c}$ would be exactly the same for the old and for the new algorithms.

In the actual new algorithm, there is a new randomization step: instead of fixing T, we choose T according to the (rotation-invariant) random distribution. As a result, for each vector $\vec{c}$, the actual average for a new algorithm is equal to the inaccuracy averaged over different $\vec{c}$ (to be more precise, averaged over all vectors $T^{-1}(\vec{c})$ for a random rotation $\vec{c}$). The average cannot exceed the maximum, and therefore, for every $\vec{c}$, this average (i.e., the inaccuracy of the new algorithm) cannot exceed the worst-case inaccuracy of the old algorithm. Thus, the inaccuracy of the new algorithm is indeed smaller than or equal to the inaccuracy of the old one.

To complete the proof of this statement, it is sufficient to show that in the new algorithm, the resulting probability distribution of the bases $\vec{\delta}^{(k)}_{\text{new}}$ is indeed rotation-invariant. Indeed, if we apply an additional rotation S to all bases, it is equivalent to changing from T to the composition $S \circ T$ of the rotations, i.e., equivalently, to replacing a random orthonormal basis $\vec{e}^{(1)}, \ldots, \vec{e}^{(n)}$ in the definition of the rotation T by a rotated basis $S(\vec{e}^{(1)}), \ldots, S(\vec{e}^{(n)})$. The distribution on the set of all bases is rotation-invariant; thus, applying a fixed rotation S to all elements of all the bases should not change the probabilities. Therefore, the resulting distribution of the bases $\vec{\delta}^{(k)}_{\text{new}}$ is also rotation-invariant. The statement is proven.

First preliminary conclusion. Thus, when we are looking for an optimal algorithm, we can always assume that the bases are distributed according to the rotation-invariant distribution. The only difference between different (n, N)-algorithms of this type is how they estimate c based on these bases and on the results $c^{(k)}$ of multiplying $\vec{c}$ and the vectors from these bases. Let us therefore analyze which estimations lead to smaller inaccuracy.

When looking for an optimal algorithm, we can always assume that the final estimate e is deterministic. In our general definitions, we assumed that after we get all the values $c^{(k)}$, we may still need to use some (Monte-Carlo) randomization to produce the final estimate $\widetilde{c}$. Let us show that if this is the case, then we can replace this randomized estimate $\widetilde{c}$ by its deterministic mathematical expectation $\widetilde{c}_{\text{new}} = E(\widetilde{c})$ and not increase the algorithm's inaccuracy. (To be more precise, by a mathematical expectation, we mean a *conditional* mathematical expectation, under the condition that the values $c^{(k)}$ and $\vec{\delta}^{(k)}$ are fixed.)

Indeed, since the inaccuracy function $\Phi(\widetilde{c}, c)$ is convex in $\widetilde{c}$, and for convex functions, $\Phi(E(z)) \le E(\Phi(z))$, we have $\Phi(\widetilde{c}_{\text{new}}, c) = \Phi(E(\widetilde{c}_{\text{old}}), c) \le E(\Phi(\widetilde{c}_{\text{old}}, c))$. Therefore, the inaccuracy of the new algorithm (with the deterministic last step) cannot exceed the inaccuracy of the old method (with possibly stochastic last step). The statement is proven.

When looking for an optimal algorithm, we can assume that the final estimation depends only on $c^{(k)}$ and not on the vectors $\vec{\delta}^{(k)}$. Indeed, let us take an arbitrary (n, N)-algorithm which have passed through all our previous modifications. In particular, for this algorithm, the final step is deterministic, i.e., $\widetilde{c} = \widetilde{e}\left(c^{(1)}, \ldots, c^{(N)}, \vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}\right)$ for some function $\widetilde{e}$.

For an arbitrary rotation T, we can consider an alternative function $\widetilde{e}_T\left(c^{(1)}, \ldots, c^{(N)}, \vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}\right) = \widetilde{e}\left(c^{(1)}, \ldots, c^{(N)}, T\left(\vec{\delta}^{(1)}\right), \ldots, T\left(\vec{\delta}^{(N)}\right)\right)$. The use of this function is equivalent to using the new (rotated) basis $T\left(\vec{\delta}^{(k)}\right)$ *and* the new (rotated) unknown vector $\vec{c}_{\text{new}} = T^{-1}(\vec{c})$.

Since we consider an algorithm which has already passed through our improvements, the bases $\vec{\delta}^{(k)}$ are simply randomly distributed in a rotation-invariant way, so adding a rotation T should not change the distribution of these bases. For the same reason, the quality of the new algorithm should not change is we replace $\vec{c}$ by its rotation. Therefore, the new algorithm should have the same inaccuracy as the old one. This is true for all rotations T, i.e., for every T, we can take $\widetilde{c} = \widetilde{e}_T$ and get the same inaccuracy.

Now, we are ready for the final step in the proof of this statement: we can take an *average* of such estimates for all possible rotations T (average over the rotation-invariant measure on all T):

$$\widetilde{c}_{\text{new}} = E_T\left[\widetilde{e}_T\left(c^{(1)}, \ldots, c^{(N)}, \vec{\delta}^{(1)}, \ldots, \vec{\delta}^{(N)}\right)\right] = E_T\left[\widetilde{e}\left(c^{(1)}, \ldots, c^{(N)}, T\left(\vec{\delta}^{(1)}\right), \ldots, T\left(\vec{\delta}^{(N)}\right)\right)\right].$$

Due to convexity, this averaging can only decrease the inaccuracy of the original algorithm. Since the basis is randomly distributed according to a rotation-invariant distribution, averaging over all T is equivalent to averaging over all possible bases. Hence, this averaging leads to an estimate which does not depend on the basis vectors at all. The statement is proven.

Second preliminary conclusion. Thus, when we are looking for optimal algorithms, it is sufficient to consider final estimates of the type $\widetilde{c} = \widetilde{e}\left(c^{(1)}, \ldots, c^{(N)}\right)$. For the following arguments, it is convenient to combine the N values $c^{(1)}, \ldots, c^{(N)}$ into a single N-dimensional vector $\mathbf{c}$. (We will denote N-dimensional vectors differently to avoid confusion with n-dimensional ones.) In these terms, $\widetilde{c} = \widetilde{e}(\mathbf{c})$.

When looking for an optimal algorithm, we can assume that the final estimate only depends on $\sum (c^{(k)})^2$. This statement can be proved in a similar way, with the only difference that instead of arbitrary rotations in a n-dimensional space, we can consider only rotations within the N-dimensional space generated by the vectors from the basis $\vec{\delta}^{(k)}$, $1 \le k \le N$.

Indeed, if we rotate the basis by such a rotation $\mathbf{T}$, we get a new method, which is equivalent to the old one (because additional rotation does not change the rotation-invariant distribution), but for which the new vectors $\vec{\delta}^{(k)}_{\rm new} = \sum_l r_{kl} \cdot \vec{\delta}^l$, where r_{kl} is an $N \times N$ rotation matrix $\mathbf{T}$ in a N-dimensional space. For this new basis, the new values $c^{(k}_{\rm new}) = \vec{c} \cdot \vec{\delta}^{(k)}_{\rm new}$ are obtained by the same rotation from the old value $c^{(k)}$: $c^{(k)}_{\rm new} = \sum_l r_{kl} \cdot c^{(l)}_{\rm old}$. In other words, the new N-dimensional vector $\mathbf{c}_{\rm new}$ is obtained from the old vector by a rotation $\mathbf{T}$: $\mathbf{c}_{\rm new} = \mathbf{T}(\mathbf{c}_{\rm old})$. Thus, instead of the old estimate $\widetilde{c} = \widetilde{e}(\mathbf{c})$, we can, for every $(N \times N)$-rotation matrix $\mathbf{T}$, consider a new estimate $\widetilde{c} = \widetilde{e}_{\mathbf{T}}(\mathbf{c}) = \widetilde{e}(\mathbf{T}(\mathbf{c}))$.

Now, we are ready for the final step in the proof of this statement: we can take an *average* of such estimates for all possible N-dimensional rotations $\mathbf{T}$ (average over the rotation-invariant measure on all T): $\widetilde{c}_{\rm new} = E_{\mathbf{T}}\left[\widetilde{e}_{\mathbf{T}}(\mathbf{c})\right] = E_{\mathbf{T}}\left[\widetilde{e}(\mathbf{T}(\mathbf{c}))\right]$. Due to convexity, this averaging can only decrease the inaccuracy of the original algorithm. Since the rotation $\mathbf{T}$ is randomly distributed according to a rotation-invariant distribution, the vector $\mathbf{T}(\mathbf{c})$ is rotation-invariant distributed; all such vectors have the same length as the original N-dimensional vector $\mathbf{c}$. Therefore, the vector $\mathbf{T}(\mathbf{c})$ is uniformly distributed over a sphere of radius $\|\mathbf{c}\| = \sqrt{\sum (c^{(k)})^2}$. Hence, this averaging leads to an estimate which does not depend on the actual values of the components of the vector $\mathbf{c}$, but only depends on this length.

The statement is proven, and so is the theorem.

ACKNOWLEDGMENTS

This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by the United Space Alliance, grant No. NAS 9-20000 (P.O. 297A001153), and by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518.

REFERENCES

1. M. Berz, C. Bischof, G. Corliss, and A. Griewank, *Computational differentiation: techniques, applications, and tools*, SIAM, Philadelphia, 1996.
2. W. A. Fuller, *Measurement error models*, J. Wiley & Sons, New York, 1987.
3. R. B. Kearfott and V. Kreinovich (eds.), *Applications of interval computations*, Kluwer, Dordrecht, 1996.
4. E. Koltik, V. G. Dmitriev, N. A. Zheludeva, and V. Kreinovich, "An optimal method for estimating a random error component," *Investigations in Error Estimation*, Proceedings of the Mendeleev Metrological Institute, Leningrad, pp. 36–41, 1986 (in Russian).
5. H. T. Nguyen and V. Kreinovich, *Applications of continuous mathematics to computer science*, Kluwer, Dordrecht, 1997.
6. S. Rabinovich, *Measurement errors: theory and practice*, American Institute of Physics, N.Y., 1993.
7. Website on interval computations: http://cs.utep.edu/interval-comp.