For Interval Computations, If Absolute-Accuracy Optimization is NP-Hard, Then So Is Relative-Accuracy Optimization

Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
email vladik@cs.utep.edu

Abstract

One of the basic problems of interval computations is to compute a range of a given function $f(x_1, \ldots, x_n)$ over a given box

$$[x_1^-, x_1^+] \times \ldots \times [x_n^-, x_n^+]$$

(i.e., to compute the maximum and the minimum of the function on the box). For many classes of functions (e.g., for quadratic functions) this problem is NP-hard; it is even NP-hard if instead of computing the minimum and maximum *exactly*, we want to compute them with a given (absolute) *accuracy* $\varepsilon > 0$. In practical situations, it is more realistic to ask for a *relative* accuracy; are the corresponding problems still NP-hard? We show that under some reasonable conditions, NP-hardness of absolute-accuracy optimization implies that relative-accuracy optimization is NP-hard as well.

Data processing: a practical problem which leads to interval computations. In many real-life situations, we are interested in the value of some physical quantity y which is difficult (or even impossible) to measure directly. To estimate y, we measure directly measurable quantities $x_1, \ldots, x_n$ which have a known relationship with y, and then reconstruct y from the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of these measurements by using this known relation: $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, where f is a known algorithm.

Measurements are never 100% accurate; as a result, the actual value x_i of each measured quantity may differ from the measured value $\widetilde{x}_i$. If we know the upper bound Δ_i for the measurement error $|\Delta x_i| = |\widetilde{x}_i - x_i|$, then after we get the measurement result $\widetilde{x}_i$, we can conclude that the actual value x_i of the measured quantity belongs to the *interval* $\mathbf{x}_i = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. A natural

question is: when $x_i \in \mathbf{x}_i$, what is the resulting interval $\mathbf{y} = f(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \{f(x_1, \ldots, x_n) \,|\, x_i \in \mathbf{x}_i\}$ of possible values of y? The problem of estimating this range interval is called the problem of *interval computations* (see, e.g., [3, 4, 5, 6, 8]).

Interval computations as a particular case of global optimization. In optimization terms:

- the lower endpoint y^- of the range interval $\mathbf{y} = [y^-, y^+]$ is the infimum of the function $f(x_1, \ldots, x_n)$ on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, and
- the upper endpoint y^+ of the range interval is the supremum of the function $f(x_1, \ldots, x_n)$ over the same box.

Computational complexity of interval computations: why it is important. Often, an algorithm $f(x_1, \ldots, x_n)$ used in data processing is reasonably complex and requires a lot of computational steps (and hence, a lot of computation time). When we take into consideration the inaccuracy of direct measurements, and consider the corresponding interval computation problem, then we, in essence, move from the problem of computing the value of a function to an optimization problem of computing the maximum (and the minimum) of a function.

An optimization problem is usually much more time-consuming than simply computing the value of an optimized function at a single point; as a result, the resulting interval computations – i.e., data processing which takes measurement uncertainty into consideration – is much more time-consuming than data processing which does not take this uncertainty into consideration. As a consequence, a data processing algorithm which is quite feasible when we do not take measurement uncertainty into consideration may lead to impractically long computations when this uncertainty is considered (i.e., when we consider the corresponding interval computations problem).

It is therefore very important to estimate the computational complexity of different problems of interval computation.

Computational complexity of interval computations: how to define it. In numerical mathematics and theoretical computer science, there exist several different definitions of computational complexity.

Traditionally, in numerical mathematics, computational complexity is defined as a number of elementary operations (such as additions, multiplications, etc.) which form the algorithm. For example, it is usually stated that the traditional algorithm for matrix multiplication of two $n \times n$ matrices requires $O(n^3)$ steps, meaning $O(n^3)$ multiplications and additions of the corresponding elements. The corresponding definitions are described and used, e.g., in [1]. In these definitions of computational complexity, we assume that the input consists of several real numbers, and that each arithmetic operation with real numbers is counted as a single computational step.

This model of real number computations is, of course, an idealized description of the actual data processing: in reality, we do not know the real numbers which are the actual values of the physical quantities, we only deal with the measurement results which are approximations to these actual values. To perform data processing, we need to input these measurement results into a computer, and in most existing computers, we cannot represent arbitrary real numbers, only rational ones. In most computers, a "real number" means a finite binary expressions of the type 0.00110_2. (We can also implement arithmetic of rational numbers by representing each rational number m/n as a pair of integers (m, n).)

When we are interested in the computational complexity of the original data processing algorithm – without taking measurement uncertainty into consideration – it is thus reasonable to ignore the difference between the original real number and its rational approximate value (with which the actual computations are performed). In this case, it makes perfect sense to use traditional numerical-mathematics models of real number computations (e.g., models from [1]).

For interval computations, however, the situation is different. We cannot any longer ignore the difference between the actual and the measured values of the input quantities, because the entire complexity of interval computations comes from this difference.

Let us give one more argument why this difference must be taken into consideration. Our main motivation for studying the computational complexity of interval computation is to estimate the computation time of different algorithms. This computation time depends not only on what operations we perform, but also on how accurate the measurements are. Less accurate measurements generate fewer digits of the measured values and therefore, fewer bits in the computer representation of the input data; the more accurate the measurements, the more extra digits they generate and therefore, the longer it takes to process all these digits.

Even for the simplest arithmetic operations, such as addition or multiplication of two numbers, the actual computation time depends on the number of digits in these numbers: the more bits in these numbers, the more bit operations addition (or multiplication) takes and therefore, the more computation time this arithmetic operation requires. Since we are interested in analyzing, e.g., how the computation time depends on the measurement accuracy, we cannot any longer consider an arithmetic operation such as addition or multiplication as a single step: we must take into consideration the number of bit operations of which each such arithmetic operation consists.

Therefore, for interval computations, we must use a different computation model, in which the input data are rational numbers, and the computational complexity is defined as the total number of bit operations. This model is mainly used in theoretical computer science to describe computational complexity of different discrete-data algorithms; it is often called a *Turing machine* model of computations. In the general computer science context, this model is described, e.g., in [2, 9]; in the context of interval computations, it is described in [7]. In

this paper, we will use this bit-wise (Turing machine) model of computation.

Interval computation is NP-hard for many reasonable classes of problems. We can apply the above analysis to explain the exact meaning of computational complexity of interval computation problems. In accordance with the above analysis, each input interval $\mathbf{x}_i = [x_i^-, x_i^+]$ is given as a pair of two rational numbers x_i^- and x_i^+; all the coefficients in an algorithm $f(x_1, \ldots, x_n)$ are also rational. For example, a *polynomial* algorithm $f(x_1, \ldots, x_n)$ would mean that we consider expressions of the type

$$f(x_1, \ldots, x_n) = a_0 + \sum a_i \cdot x_i + \sum a_{ij} \cdot x_i \cdot x_j + \sum a_{ijk} \cdot x_i \cdot x_j \cdot x_k + \ldots$$

where the coefficients a_0, a_i, a_{ij}, a_{ijk}, $\ldots$, are rational numbers. This expression (or, to be more precise, the binary computer representation of this expression) serves as a (computer) *code* of the corresponding function. For polynomials, one can easily write a feasible (polynomial-time) program which, given a code for a polynomial function $f(x_1, \ldots, x_n)$ and the (rational) values of the variables $x_1, \ldots, x_n$, returns the value of the corresponding polynomial on given inputs.

In general, we have a *class* $\mathcal{F}$ of functions, i.e., a class of codes (program sequences) and a feasible (polynomial-time) algorithm which takes a function code and the rational values of the variables, and returns the rational value of the function. For interval computations, the input consists of a function f (i.e., a code from a given class $\mathcal{F}$), and $2n$ rational numbers x_i^-, x_i^+, $1 \le i \le n$.

This description takes care of the input. An additional subtlety is how to represent the desired output. In some cases (e.g., for linear functions $f(x_1, \ldots, x_n)$), if the endpoints x_i^- and x_i^+ are rational, and all the coefficients of the algorithm $f(x_1, \ldots, x_n)$ are rational, then the infimum y^- and supremum y^+ of the function $f(x_1, \ldots, x_n)$ on the box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ are themselves rational numbers. However, in many other cases, the values y^- and y^+ are not rational; so, in the existing computer languages, we cannot represent these numbers exactly. The only thing we can do is to fix a *representation accuracy*, i.e., a rational number $\varepsilon > 0$, and generate ε-approximations to the desired values y^- and y^+, i.e., numbers $\widetilde{y^-}$ and $\widetilde{y^+}$ for which $\left|\widetilde{y^-} - y^-\right| \le \varepsilon$ and $\left|\widetilde{y^+} - y^+\right| \le \varepsilon$.

In other words, instead of looking for, say, the *precise* maximum y^+ of a function f, we are looking for *absolute-accuracy* optimization, i.e., we fix a rational number $\varepsilon > 0$ and we are looking for a value $\widetilde{y^+}$ which is (absolutely) ε-close to the (unknown) precise maximum, i.e., for which $\left|\widetilde{y^+} - y^+\right| \le \varepsilon$.

It is known that for every $\varepsilon > 0$, the corresponding absolute-accuracy optimization problem is NP-hard for many reasonable classes of functions (see, e.g., [7]): e.g., it is NP-hard:

- for the class of all polynomials;
- for the class of all quadratic functions $f(x_1, \ldots, x_n) = a_0 + \sum a_i \cdot x_i + \sum a_{ij} \cdot x_i \cdot x_j$ (the proof of this particular result is due to Vavasis [11]);

- for several classes of quadratic functions with sparse matrices a_{ij};
- for all feasible piecewise-linear functions, etc.

(For related problems, see, e.g., [10].)

In these results, the required accuracy is fixed, and for this fixed required accuracy ε, the corresponding absolute-accuracy approximation problem is NP-hard. (As a corollary, we can conclude that the following weaker result is also true: if we consider the required accuracy ε part of the data of the problem, then the resulting problem is NP-hard as well.)

Relative-accuracy optimization may be more realistic. From the practical viewpoint, the requirement to have the bounds with an absolute accuracy may be too strong. If the actual maximum is large, it may be more reasonable to look for an estimate which is good within a certain *relative* accuracy, i.e., for which $\left|\widetilde{y^+} - y^+\right| \le \varepsilon \cdot \left|\widetilde{y^+}\right|$ and $\left|\widetilde{y^+} - y^+\right| \le \varepsilon \cdot |y^+|$, where $\varepsilon > 0$ is a given relative accuracy (such as 1% or 0.01%).

For *small* values y^+ thus defined relative accuracy is too strong: e.g., for $y^+ = 0$, the only way to compute this value with relative accuracy ε (in the sense of the above two inequalities) is to compute it precisely, because the only value $\widetilde{y^+}$ which satisfies these two inequalities is $\widetilde{y^+} = 0 = y^+$. For small values y^+, absolute accuracy makes more practical sense. So, a more realistic approach is to fix a value ε and to require that the computed value $\widetilde{y^+}$ and the actual value y^+ are either absolutely ε-close or strongly relatively ε-close (in the sense of the above two inequalities). In short, we arrive at the following definition (similar definitions were used in Ch. 11 of [7]):

Definition 1. *Let $\varepsilon > 0$.*

- *We say that two real numbers $\widetilde{a}$ and a are absolutely ε-close if $|\widetilde{a} - a| \le \varepsilon$.*
- *We say that two real numbers $\widetilde{a}$ and a are strongly relatively ε-close if $|\widetilde{a} - a| \le \varepsilon \cdot |\widetilde{a}|$ and $|\widetilde{a} - a| \le \varepsilon \cdot |a|$.*
- *We say that two real numbers $\widetilde{a}$ and a are relatively ε-close if they are either absolutely ε-close or strongly relatively ε-close.*

Is the corresponding optimization problem still NP-hard? The above notion of relative ε-closeness is weaker than the notion of absolute ε-closeness; so, in general, it may be possible that the absolute-accuracy optimization problem is NP-hard, while the weaker relative-accurate optimization problem is easier to solve.

We will show, however, that for many important classes, the new problem is still NP-hard.

Definition 2. *We say that a set B of (finite) binary sequences is feasible decidable if there exists a feasible (polynomial-time) algorithm which, given a binary sequence b, checks whether this sequence b belongs to the set B or not.*

Definition 3. *By a class $\mathcal{F}$ of computable functions we mean a triple* $(F, \mathrm{ar}, \mathrm{val})$, *where:*

- *F is a feasibly decidable set of binary sequences; elements of the set F will be called function codes, or simply codes;*
- ar *is a feasible mapping from F to the set of non-negative integers; the value* $\mathrm{ar}(f)$ *is called the arity of a function f;*
- val *is a feasible mapping which takes, as input, a tuple* $(f, x_1, \ldots, x_n)$, *where $f \in F$,* $n = \mathrm{ar}(f)$, *and x_i are rational numbers, and returns a new rational number; this new rational number will be called a value of the function f on the values $x_1, \ldots, x_n$ and denoted by $f(x_1, \ldots, x_n)$.*

Definition 4. *We say that a class* $\mathcal{F} = (F, \mathrm{ar}, \mathrm{val})$ *of computable functions is feasibly shift-invariant if there exists a feasible mapping $S : F \times Q \to F$ which maps every pair (f, c), in which f is a function code and c is a rational number, into a new function code $S(f, c)$ such that for every f and for every c, we have:*

- $\mathrm{ar}(S(f, c)) = \mathrm{ar}(f)$, *and*
- *for every tuple $x_1, \ldots, x_n$, where $n = \mathrm{ar}(f)$, we have*

$$S(f, c)(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) + c.$$

This function $S(f, c)$ will be denoted by $f + c$.

Comments.

- Informally, a class $\mathcal{F}$ is shift-invariant if for every function $f \in \mathcal{F}$ and for every rational number c, the class $\mathcal{F}$ also contains the function $f + c$, and there is a feasible way to construct a code for $f + c$ from the codes for f and c.
- One can easily see that all above classes – the class of all polynomials, the class of all quadratic functions, the class of all feasible piecewise-linear functions – are feasibly shift-invariant.

Definition 5. *We say that class $\mathcal{F}$ of computable functions is feasibly boundable if there exists a polynomial-time (feasible) algorithm which, given a function $f \in \mathcal{F}$ and a given box (with rational coordinates), produces two rational numbers: a lower bound B^- and an upper bound B^+ for the range of this function on this box.*

Comments.

- To avoid confusion, we want to emphasize that in this definition, we require that a feasible algorithm computes lower and upper *bounds*, not for the *exact* infima and suprema.

- All above classes – the class of all polynomials, the class of all quadratic functions, etc. – are feasibly boundable. Indeed, to compute the lower and upper bounds for a polynomial on a given box, we can use, e.g., so-called naive interval computations, when we describe the computation of a polynomial as a sequence of elementary arithmetic operations (addition, multiplication, subtraction, etc.), and then replace each operation with real (rational) numbers in this description by the corresponding operation of interval arithmetic:

$$[a^-, a^+] + [b^-, b^+] = [a^- + b^-, a^+ + b^+];$$

$$[a^-, a^+] - [b^-, b^+] = [a^- - b^+, a^+ - b^-];$$

$$[a^-, a^+] \cdot [b^-, b^+] = [\min(a^- \cdot b^-, a^- \cdot b^+, a^+ \cdot b^-, a^+ \cdot b^+),$$
$$\max(a^- \cdot b^-, a^- \cdot b^+, a^+ \cdot b^-, a^+ \cdot b^+)];$$

(for details, see, e.g., [3, 4, 5, 6, 8]). As a result, we get lower and upper bounds which do not necessarily coincide with the infimum and supremum. For example, for $f(x_1) = x_1 - x_1^2$ on the 1D box (interval) $\mathbf{x}_1 = [0, 1]$, the computation of this polynomial consists of first computing $r_1 = x_1 \cdot x_1$ and then computing $f = x_1 - r_1$; the corresponding naive interval computations lead to the intervals $\mathbf{r}_1 = \mathbf{x}_1 \cdot \mathbf{x}_1 = [0, 1]$ and $\mathbf{f} = \mathbf{x}_1 - \mathbf{r}_1 = [0, 1] - [0, 1] = [-1, 1]$. So, we conclude that the lower bound B^- for the function f is $B^- = -1$, and the upper bound is $B^+ = 1$. These bounds differ from the infimum $y^- = 0$ and supremum $y^+ = 0.25$ of the function $f(x_1) = x_1 - x_1^2$ on the interval $[0, 1]$.

Definition 6. *We say that for a class of computable functions $\mathcal{F}$, absolute-accuracy optimization is NP-hard if for every $\varepsilon > 0$, the problem of computing a maximum M of a given function $f \in \mathcal{F}$ on a given box with an absolute accuracy ε (i.e., the problem of computing a value $\widetilde{M}$ which is absolutely ε-close to M) is NP-hard.*

Comments.

- In this definition, we pick an arbitrary rational number $\varepsilon > 0$, and formulate the formulate the following general problem: given a code $f \in F$ and a tuple of rational intervals $[x_1^-, x_1^+], \ldots, [x_n^-, x_n^+]$, generate a rational number $\widetilde{M}$ which is ε-close to the actual supremum M of f on the corresponding box $[x_1^-, x_1^+] \times \ldots \times [x_n^-, x_n^+]$. As a result, we have a family of problems corresponding to different values of ε. We require that all the problems from this family are NP-hard.

- For all above classes – the class of all polynomials, the class of all quadratic functions, the class of all feasible piecewise-linear functions – absolute-accurate optimization is NP-hard (see, e.g., [7, 11]).

Definition 7. *We say that for a class of functions $\mathcal{F}$, relative-accuracy optimization is NP-hard if for every $\varepsilon > 0$, the problem of computing a maximum M of a given function $f \in \mathcal{F}$ on a given box with a relative accuracy ε (i.e., the problem of computing a value $\widetilde{M}$ which is relatively ε-close to M in the sense of Definition 1) is NP-hard.*

Comment. In both definitions 6 and 7, the required accuracy is fixed, and for this fixed required accuracy ε, the corresponding approximation problem is NP-hard. So, each of these definitions does not simply state that a single problem is NP-hard; it actually states that for a whole family of problems (characterized by a parameter ε), every problem from this family is NP-hard.

Proposition. *If for some shift-invariant feasibly boundable class $\mathcal{F}$, absolute-accuracy optimization is NP-hard, then for this class, relative-accuracy optimization is also NP-hard.*

Comments.

- The condition that the class should be feasibly boundable is not really restrictive: indeed, if we are able to solve the relative-accuracy optimization problem in feasible time, then we get feasible bounds as well; therefore, if the class is not feasible boundable, then no feasible algorithm can solve the corresponding relative-accuracy optimization problem.
- As a corollary, we conclude that for quadratic polynomials (and for quadratic functions with a sparse matrix a_{ij}), the relative-accuracy optimization problem is NP-hard.

Proof. We want to prove that for every $\varepsilon > 0$, the corresponding relative-accuracy optimization problem is NP-hard. Let us therefore fix an arbitrary rational number $\varepsilon > 0$ and prove that the corresponding relative-accuracy optimization problem is NP-hard.

To prove this NP-hardness, we will show that for feasibly shift-invariant feasibly boundable classes $\mathcal{F}$, the appropriate absolute-accuracy optimization problem (for some δ depending on ε) can be polynomially Turing-reduced to the relative-accuracy optimization problem with accuracy ε for this same class. In other words, we will show that we can solve the appropriate absolute-accuracy optimization problem in polynomial time if we use the solver of the relative-accuracy optimization problem with accuracy ε as an oracle (for precise definitions, see, e.g., [2, 9]). Since for every $\delta > 0$, the absolute-accuracy optimization problem is known to be NP-hard, we can thus conclude that the relative-accuracy optimization problem with accuracy ε is NP-hard as well.

Let us show the desired reduction. Assume that we want to solve the absolute-accuracy optimization problem with some accuracy $\delta > 0$ (the value δ will be specified later). This means that, given a function $f(x_1, \ldots, x_n)$ (from the class $\mathcal{F}$) and a box, we must find a values which is absolutely δ-close to the

maximum of the given function on a given box. Let us show how we can use the relative-accuracy oracle to solve this problem.

Since the function $f(x_1, \ldots, x_n)$ is feasibly boundable, we can apply the polynomial-time bounding algorithm and find a lower bound B^- and an upper bound B^+ for the range of the function f on a given box. Then, for every $x = (x_1, \ldots, x_n)$ from the given box, the value $f(x)$ belongs to the interval $[B^-, B^+]$. In particular, the actual maximum M of the function is somewhere in this interval. Let us denote the width $B^+ - B^-$ of this interval by Δ.

Let us fix an integer m (its value will also be specified later). We can define, for every integer i from 0 to m, the values $c_i = B^- + i \cdot \Delta/m$. The intervals $[c_i - \Delta/(2m), c_i + \Delta/(2m)] = [B_i + (i-1/2)\cdot(\Delta/m), B_i + (i+1/2)\cdot(\Delta/m)]$, cover the entire interval $[B^-, B^+] = [B^-, B^- + \Delta]$; so, every value v from the interval $[B^-, B^+]$ is covered by at least one of these intervals $[c_i - \Delta/(2m), c_i + \Delta/(2m)]$. For the corresponding i, we have $v \in [c_i - \Delta/(2m), c_i + \Delta/(2m)]$ and hence, $|v - c_i| \le \Delta/(2m)$. In particular, since the actual maximum M of the function $f(x)$ belongs to the interval $[B^-, B^+]$, we can take $v = M$ and conclude that there exists an integer j for which $0 \le j \le m$ and $|M - c_j| \le \Delta/(2m)$.

One of the conditions of the proposition is that the class of functions $\mathcal{F}$ is feasibly shift-invariant. By definition, this means that for every function $f \in \mathcal{F}$ and for every rational number c, the class $\mathcal{F}$ also contains the function $f + c$. In particular, for for every integer i from 0 to m, we can take the rational number $c = -c_i$ and conclude that the function $f_i = f - c_i$ also belongs to the class $\mathcal{F}$. The actual maximum M_i of each function f_i is equal to $M_i = M - c_i$. Let us apply the relative-accuracy oracle to the functions $f_0, \ldots, f_m$, and compute the ε-relative approximations $\widetilde{M}_i$ to these maxima.

We know that there is a value j for which $|M - c_j| \le \Delta/(2m)$. For this value j, we therefore have $|M_j| = |M - c_j| \le \Delta/(2m)$. By our definition of an oracle, $\widetilde{M}_j$ is relatively ε-close to M_j. According to Definition 1, this means the following:

- either $\left|\widetilde{M}_j - M_j\right| \le \varepsilon$, in which case

$$\left|\widetilde{M}_j\right| \le |M_j| + \left|\widetilde{M}_j - M_j\right| \le \frac{\Delta}{2m} + \varepsilon,$$

- or $\left|\widetilde{M}_j - M_j\right| \le \varepsilon \cdot |M_j|$, in which case

$$\left|\widetilde{M}_j\right| \le |M_j| + \left|\widetilde{M}_j - M_j\right| \le |M_j| + \varepsilon \cdot |M_j| = \frac{\Delta}{2m} + \varepsilon \cdot \frac{\Delta}{2m}.$$

In both cases,

$$\left|\widetilde{M}_j\right| \le \frac{\Delta}{2m} + \max\left(\varepsilon, \varepsilon \cdot \frac{\Delta}{2m}\right).$$

Since for every $a, b \geq 0$, we have $\max(a, b) \leq a + b$, we can conclude that $\left|\widetilde{M_j}\right| \leq K$, where by K, we denoted

$$K = \frac{\Delta}{2m} + \varepsilon + \varepsilon \cdot \frac{\Delta}{2m} = \varepsilon + \frac{\Delta}{2m} \cdot (1 + \varepsilon). \tag{1}$$

So, there always exists an index k for which $\left|\widetilde{M_k}\right| \leq K$. Vice versa, if we find such k, then, since M_k is relatively ε-close to $\widetilde{M_k}$, we can conclude that:

- either $\left|M_k - \widetilde{M_k}\right| \leq \varepsilon$, in which case,

$$|M_k| \leq \left|\widetilde{M_k}\right| + \left|M_k - \widetilde{M_k}\right| \leq K + \varepsilon,$$

- or $\left|M_k - \widetilde{M_k}\right| \leq \varepsilon \cdot \left|\widetilde{M_k}\right|$, in which case,

$$|M_k| \leq \left|\widetilde{M_k}\right| + \left|M_k - \widetilde{M_k}\right| \leq \left|\widetilde{M_k}\right| + \varepsilon \cdot \left|\widetilde{M_k}\right| \leq K + \varepsilon \cdot K.$$

In both cases, $|M_k| \leq K + \max(\varepsilon, K \cdot \varepsilon)$, and therefore (due to $\max(a, b) \leq a + b$), $|M_k| \leq B$, where by B, we denoted:

$$B = K + \varepsilon + \varepsilon \cdot K = \varepsilon + K \cdot (1 + \varepsilon). \tag{2}$$

We know that for every k, the maximum M_k of the auxiliary function $f_k = f - c_k$ is related to the maximum M of the original function $f(x)$ by the formula $M_k = M - c_k$. So, from the inequality $|M_k| \leq B$, we can conclude that $|M - c_k| \leq B$, and therefore, that the actual value of M belongs to the interval $[c_k - B, c_k + B]$ of width $\Delta' = 2B$.

Substituting the expression (2) for B into the formula for Δ', and then substituting the formula (1) for K, we can get the following expression for Δ':

$$\Delta' = 4\varepsilon + 2\varepsilon^2 + \frac{\Delta \cdot (1 + \varepsilon)^2}{m}. \tag{3}$$

If we choose m for which $(1 + \varepsilon)^2/m \leq 1/2$, e.g., if we choose

$$m = \lceil 2(1 + \varepsilon)^2 \rceil, \tag{4}$$

then we can conclude that

$$\Delta' \leq 4\varepsilon + 2\varepsilon^2 + \frac{\Delta}{2}. \tag{5}$$

Subtracting $2 \cdot (4\varepsilon + 2\varepsilon^2)$ from both sides of this inequality, we conclude that

$$\Delta' - (8\varepsilon + 4\varepsilon^2) \leq \frac{\Delta - (8\varepsilon + 4\varepsilon^2)}{2}. \tag{6}$$

So, if originally, we had $\Delta > (8\varepsilon + 4\varepsilon^2)$, then the difference between Δ and the ε-expression $8\varepsilon + 4\varepsilon^2$ gets halved.

We can repeat the same construction, starting with the new maximum-containing interval $[c_k - B, c_k + B]$ of width Δ' (instead of the original interval $[B^-, B^+]$ of width Δ), etc., etc., until we reach (in polynomially many iterations, i.e., in $\approx \log_2(\Delta/\varepsilon)$ iterations) a maximum-containing interval for which the difference between its width Δ_{final} and the ε-expression $8\varepsilon + 4\varepsilon^2$ is $\le \varepsilon$. For this interval, $\Delta_{\text{final}} \le 9\varepsilon + 4\varepsilon^2$. So:

- either after the first step we already had a maximum-containing interval of width $\Delta \le 8\varepsilon + 4\varepsilon^2 < 9\varepsilon + 4\varepsilon^2$,
- or, in polynomial number of steps, we get a maximum-containing interval of width $\Delta_{\text{final}} \le 9\varepsilon + 4\varepsilon^2$.

In both cases, if we choose $\delta = 9\varepsilon + 4\varepsilon^2$, we will find an interval of width $\le \delta$ which contains the desired maximum M. Each endpoint of this interval is, therefore, the desired absolute δ-approximation to M.

So, if we can use the solution of the relative-accuracy optimization problem (corresponding to the given ε) as an oracle, then, for this δ, the following algorithm solves the corresponding absolute-accuracy optimization problem.

First, we compute the integer value m by using the formula (4). Now that m is fixed, we can described the desired algorithm.

This algorithm takes, as input, a function f and a box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$. Based on this input, we do the following:

1. First, we apply the (polynomial-time) bounding algorithm to the given function f on a given box, and compute the bounds B^- and B^+.
2. Second, we compute the difference $\Delta = B^+ - B^-$. If $\Delta \le \delta$, then the value B^+ is already the desired absolute δ-approximation to the maximum M, so we stop; otherwise, we continue.
3. If we continue, then we compute the value K by using the formula (1), the value B by using the formula (2), and $m+1$ values $c_i = B^- + i \cdot \Delta/m$, $0 \le i \le m$. For a given ε, the integer m is fixed, so computing all these values requires a time which is bounded by a polynomial of the bit length of the values B^- and B^+ and thus, by a polynomial in terms of the bit length the input data.
4. Then, we apply the shifting algorithm to compute the codes for the functions $f_i = f - c_i = f + (-c_i)$, $0 \le i \le m$. Since the class $\mathcal{F}$ is feasibly shift-invariant, the shifting algorithm requires polynomial time.
5. Next, we apply the oracle (which solves the relative-accuracy approximation problem) to find ε-approximations $\widetilde{M}_0, \widetilde{M}_1, \ldots, \widetilde{M}_k, \ldots$ $(k \le m)$ to

the suprema of the functions $f_0, f_1, \ldots$ After we get each value $\widetilde{M}_k$, we check whether $|\widetilde{M}_k| \le K$; when we find a value k for which this inequality is true, we stop. (We have shown, in the previous part of the proof, that such a value k always exists.) The number of calls to an oracle in this step is bounded by a constant $(m+1)$, and the input to all these calls is feasible bounded (by a polynomial of the length of the input).

6. For the value k from Step 3, we then take $B^- := c_k - B$ and $B^+ := c_k + B$, and go to Step 2.

Each iteration of this algorithm requires polynomial time (with a uniform polynomial bound), and we have shown that the total number of required iterations is bounded by a polynomial of the length of the input. Therefore, the total computation time of this oracle-using algorithm is also bounded by a polynomial. Thus, the above reduction is indeed polynomial-time.

The polynomial-time reduction is proven, and so is the proposition.

Acknowledgments. This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), 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, and by the National Security Agency under Grants No. MDA904-98-1-0561 and MDA904-98-1-0564.

The author is thankful to all the participants of the Conference on Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems (Gainesville, Florida, February 28–March 2, 1999), especially to P. Pardalos and S. Vavasis, for valuable discussions, and to the anonymous referees for valuable suggestions.

References

[1] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and real computation*, Springer Verlag, N.Y., 1997.

[2] M. E. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, San Francisco, 1979.

[3] R. Hammer, M. Hocks, U. Kulisch, D. Ratz, *Numerical toolbox for verified computing. I. Basic numerical problems*, Springer Verlag, Heidelberg, N.Y., 1993.

[4] E. R. Hansen, *Global optimization using interval analysis*, Marcel Dekker, N.Y., 1992.

[5] R. B. Kearfott, *Rigorous global search: continuous problems*, Kluwer, Dordrecht, 1996.

[6] R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[7] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational complexity and feasibility of data processing and interval computations*, Kluwer, Dordrecht, 1997.

[8] R. Moore, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[9] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.

[10] P. M. Pardalos, *Complexity in Numerical Optimization*, World Scientific, Singapore, 1993.

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