Computing Population Variance and Entropy under Interval Uncertainty: Linear-Time Algorithms

Gang Xiang, Martine Ceberio, and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso, El Paso, TX 79968, USA,
gxiang@utep.edu, mceberio@cs.utep.edu, vladik@utep.edu

Abstract

In statistical analysis of measurement results it is often necessary to compute the range $[\underline{V}, \overline{V}]$ of the population variance $V = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - E)^2$ $\left(\text{where } E = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i\right)$ when we only know the intervals $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$ of possible values of the x_i. While $\underline{V}$ can be computed efficiently, the problem of computing $\overline{V}$ is, in general, NP-hard. In our previous paper "Population Variance under Interval Uncertainty: A New Algorithm" (*Reliable Computing*, 2006, Vol. 12, No. 4, pp. 273–280) we showed that in a practically important case we can use constraints techniques to compute $\overline{V}$ in time $O(n \cdot \log(n))$. In this paper we provide new algorithms that compute $\underline{V}$ (in all cases) and $\overline{V}$ (for the above case) in linear time $O(n)$.

Similar linear-time algorithms are described for computing the range of the entropy $S = -\sum_{i=1}^{n} p_i \cdot \log(p_i)$ when we only know the intervals $\mathbf{p}_i = [\underline{p}_i, \overline{p}_i]$ of possible values of probabilities p_i.

In general, a statistical characteristic f can be more complex so that even computing f can take much longer than linear time. For such f, the question is how to compute the range $[\underline{y}, \overline{y}]$ in as few calls to f as possible. We show that for convex symmetric functions f, we can compute $\overline{y}$ in n calls to f.

1 Computing Population Variance under Interval Uncertainty: Formulation of the Problem

Statistical analysis is important. Once we have n measurement results $x_1, \ldots, x_n$, the traditional statistical analysis starts with computing the standard statistics such as population mean $E = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ and population variance $V = M - E^2$, where $M \overset{\text{def}}{=} \frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2$; see, e.g., [17].

These values are useful, e.g. in detecting *outliers:* once we know the mean E and the standard deviation $\sigma \overset{\text{def}}{=} \sqrt{V}$ of the normal values, we can determine outliers as values x_i for which $|x_i - E| \gg \sigma$, i.e., $|x_i - E| \geq k_0 \cdot \sigma$ for some k_0 (usually, $k_0 = 2$, 3, or 6).

Outliers are important in many application areas: in non-destructive testing, outliers indicate possible faults; in geophysics, outliers should be identified as possible locations of minerals; in medicine, outliers indicate possible illnesses, etc.

The more data points we take, the more accurate the resulting estimates for E and V. Thus, in many practical applications we process large amount of data. For instance, in geophysics, we process thousands and millions data points and in processing census data, we process data about millions of people.

Interval uncertainty. In many real-life situations, due to measurement uncertainty, instead of the actual values x_i of the measured quantity, we only have intervals $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ of possible values of x_i [10, 17].

Usually, the interval $\mathbf{x}_i$ has the form $[\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$, where $\widetilde{x}_i$ is the measurement result, and Δ_i is the known upper bound on the absolute value $|\Delta x_i|$ of the (unknown) measurement error $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$: $|\Delta x_i| \le \Delta_i$.

Another source of interval uncertainty is the existence of *detection limits* for different sensors: if a sensor did not detect any ozone, this means that the ozone concentration is below its detection limit DL, i.e., in the interval $[0, DL]$.

One more source of interval uncertainty is *discretization:* to study the effect of a pollutant on the fish, we check on the fish daily; if a fish was alive on Day 5 but dead on Day 6, then the only information about the lifetime of this fish is that it is somewhere within the interval $[5, 6]$; we have no information about the distribution of different values in this interval.

Yet another source of interval uncertainty is *privacy.* In biomedical systems, statistical analysis of the data often leads to improvements in medical recommendations. However, to maintain privacy, we do not want to use the exact values of the patient's parameters. Instead, for each parameter we select fixed values (thresholds). For each patient we only keep the corresponding range. For example, instead of keeping the exact age, we only record whether the age is between 0 and 10, 10 and 20, 20 and 30, etc.

Finally, intervals occur if instead of measurements, we use *expert estimates* for difficult-to-measure quantities. This is due to the fact that experts can rarely describe exact values of the physical quantities. At best, they can provide the bounds on the possible values, i.e., intervals which contain the (unknown) actual value of the quantity of interest.

In statistical analysis it is necessary to take into account interval uncertainty. As we have mentioned, in many practical situations it is desirable to know the mean E and the variance V. In case of interval uncertainty, different values $x_i \in \mathbf{x}_i$ generally lead to different values of E and V. It is therefore desirable to compute the ranges $\mathbf{E} = [\underline{E}, \overline{E}]$ and $\mathbf{V} = [\underline{V}, \overline{V}]$ of possible values of E and V when $x_i \in \mathbf{x}_i$.

Example of practical applications: in brief. Interval ranges for statistical characteristics have been successfully used in geophysics [15, 16], in environmental science, and in many other application areas; see [13, 14] and references therein.

Computing the range of variance under interval uncertainty: what is known. Since the population mean E is a monotonic function of its n variables $x_1, \ldots, x_n$, its range can be easily computed as $\mathbf{E} = \left[\frac{1}{n} \cdot \sum\limits_{i=1}^{n} \underline{x}_i, \frac{1}{n} \cdot \sum\limits_{i=1}^{n} \overline{x}_i\right]$.

The population variance $V(x_1, \ldots, x_n)$ is, in general, not a monotonic function of its variables x_i. As a result, we cannot easily indicate the values at which this function attains its minimum and maximum. It can be easily checked that population variance is a convex function, which allows us to use known efficient (polynomial time) algorithms of convex optimization to find the minimum $\underline{V}$ of this convex function $V(x_1, \ldots, x_n)$ on the convex box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$. For this particular convex function, it is possible to describe algorithms which are faster than in the general convex case. Namely, we can compute the lower bound $\underline{V}$ in time $O(n \cdot \log(n))$; see, e.g., [14].

For the upper bound $\overline{V}$ the situation is more complicated. Specifically, it is known that the maximum of a convex function on a convex set is attained at one of its extreme points. Thus, the maximum $\overline{V}$ is attained at one of the extreme points of the convex box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$, i.e., when for every i, the variable x_i is equal to one of the endpoints: $x_i = \underline{x}_i$ or $x_i = \overline{x}_i$. There are 2^n combinations of n such endpoints. Therefore, we can compute $\overline{V}$ by finding the maximum of 2^n corresponding values. The computation time for this computation grows exponentially with the size n of the problem.

It is known that in general, computing $\overline{V}$ is an NP-hard problem [7, 14]. This NP-hardness result means that (unless P=NP) no general algorithm is possible that would always compute $\overline{V}$ in feasible (polynomial) time. Crudely speaking, in the worst case, the exponential computation time is inevitable.

However, it is possible to compute $\overline{V}$ in polynomial time in some practically reasonable cases.

In general, we can have both high-accuracy data points (e.g., points with narrow uncertainty intervals) and low-accuracy data points (e.g., points with much wider uncertainty intervals). For example, in addition to accurate measurements that provide narrow intervals for the values of the desired quantity, we may have expert estimates. It is well known that in statistics, if we have a large number of high-accuracy data points, then the additional information provided by low-accuracy data points is negligible. As a result, in statistical analysis only high-accuracy data points are usually taken into account. With this practice in mind, it is reasonable to restrict ourselves to the case when all the intervals are approximately of the same width. This happens when all measurements have been done by a single measuring instrument (or by several measuring instruments of the same type).

How can we describe this mathematically? A clear indication that we have two measuring instruments (MI) of different quality is that one interval is a proper subset of the other one: $[\underline{x}_i, \overline{x}_i] \subseteq (\underline{x}_j, \overline{x}_j)$. Thus, the condition that all data points are of the same accuracy can be formalized as a requirement that no interval is a proper subinterval of another one.

This property holds for other sources of interval uncertainty: e.g., for detection limits, we have intervals of the type $[0, DL_i]$ for which $[0, DL_i] \not\subseteq (0, DL_j)$, and in the privacy case, when we have intervals $[b_k, b_{k+1}]$ between two consecutive thresholds, none of which is a proper subset of the other.

In this practically useful case when no interval is a proper subset of another one, there exists an algorithm for computing $\overline{V}$ in time $O(n \cdot \log(n))$ [5]. This algorithm can be extended to a more general case when the above "no-subset" property holds only for the "narrowed" intervals. Specifically, an $O(n \cdot \log(n))$ time algorithm exists for the case when no "narrowed interval" $[x_i^-, x_i^+]$ is a proper subinterval of the interior of another narrowed interval, where $x^- \stackrel{\text{def}}{=} \widetilde{x}_i - \dfrac{\Delta_i}{n}$ and $x_i^+ \stackrel{\text{def}}{=} \widetilde{x}_i + \dfrac{\Delta_i}{n}$. This condition is equivalent to requiring that $|\widetilde{x}_i - \widetilde{x}_j| \geq \dfrac{|\Delta_i - \Delta_j|}{n}$ for all $i \neq j$.

In this paper we describe two new linear-time algorithms:

- a linear-time algorithm that computes $\underline{V}$ for all possible intervals, and
- a linear-time algorithm that computes $\overline{V}$ for intervals that satisfy the above no-subset property.

Comment. Some preliminary results have previously appeared in a Sandia technical report [8].

2 Linear-Time Algorithm for Computing $\overline{V}$ for the Case when Narrowed Intervals Satisfy the No-Subset Property

Our new algorithm is based on the known fact that we can compute the median of a set of n elements in linear time (see, e.g., [4]); our use of median is similar to the one from [3, 9]. Let us first describe the algorithm itself. In the Appendix, we provide a justification for this algorithm.

For simplicity, let us first consider the case when all the intervals are non-degenerate, i.e., when $\Delta_i > 0$ for all i.

The proposed algorithm is iterative. At each iteration of this algorithm we have three sets:

- the set I^- of all the indices i from 1 to n for which we already know that for the optimal vector x, we have $x_i = \underline{x}_i$;
- the set I^+ of all the indices j for which we already know that for the optimal vector x, we have $x_j = \overline{x}_j$;
- the set $I = \{1, \ldots, n\} \setminus (I^- \cup I^+)$ of the indices i for which we are still undecided.

In the beginning, $I^- = I^+ = \emptyset$ and $I = \{1, \ldots, n\}$. At each iteration we also update the values of two auxiliary quantities $E^- \stackrel{\text{def}}{=} \sum\limits_{i \in I^-} \underline{x}_i$ and $E^+ \stackrel{\text{def}}{=} \sum\limits_{j \in I^+} \overline{x}_j$. In principle, we could compute these values by computing these sums. However, to speed up computations on each iteration, we update these two auxiliary values in a way that is faster than re-computing the corresponding two sums. Initially, since $I^- = I^+ = \emptyset$, we take $E^- = E^+ = 0$.

At each iteration we do the following:

- first, we compute the median m of the set I (median in terms of sorting by $\widetilde{x}_i$);

- then, by analyzing the elements of the undecided set I one by one, we divide them into two subsets $P^- = \{i : \widetilde{x}_i \le \widetilde{x}_m\}$ and $P^+ = \{j : \widetilde{x}_j > \widetilde{x}_m\}$;
- we compute $e^- = E^- + \sum\limits_{i \in P^-} \underline{x}_i$ and $e^+ = E^+ + \sum\limits_{j \in P^+} \overline{x}_j$;
- if $n \cdot x_m^- < e^- + e^+$, then we replace I^- with $I^- \cup P^-$, E^- with e^-, and I with P^+;
- if $n \cdot x_m^- > e^- + e^+$, then we replace I^+ with $I^+ \cup P^+$, E^+ with e^+, and I with P^-;
- if $n \cdot x_m^- = e^- + e^+$, then we replace I^- with $I^- \cup P^-$, I^+ with $I^+ \cup P^+$, and I with $\emptyset$.

At each iteration the set of undecided indices is divided in half. Iterations continue until all indices are decided. After this we return, as $\overline{V}$, the value of the population variance for the vector x for which $x_i = \underline{x}_i$ for $i \in I^-$ and $x_j = \overline{x}_j$ for $j \in I^+$.

Comments.

- This same algorithm can be easily applied if one of the intervals consists of a single point only. This value is plugged in and the variable is eliminated.
- For readers' convenience, all the proofs – that the algorithms are correct and that they require linear time – are placed in a special Appendix.
- As with all asymptotic results, two natural questions arise: 1) How practical is the new linear time $O(n)$ algorithm? 2) For which n is it better than the known $O(n \cdot \log(n))$ algorithm for computing $\overline{V}$? In general, the answer to these questions depends on the constants in the corresponding asymptotics. The constant for the known $O(n \cdot \log(n))$ algorithm is ≈ 1. As one can see from the proof, for our new algorithm, the constant is the same as for known linear time algorithm for computing the median, i.e., it is ≈ 20 [4]; thus, the new algorithm is better when $\log_2(n) > 20$, i.e., when $n > 10^6$. We have mentioned that in many practical applications we do need to process millions of data points; in such applications, the new algorithm for computing $\overline{V}$ is indeed faster.

3 Linear-Time Algorithm for Computing $\underline{V}$

The proposed algorithm is iterative. At each iteration of this algorithm we have three sets:

- the set J^- of all the endpoints $\underline{x}_i$ and $\overline{x}_j$ for which we already know that for the optimal vector x we have, correspondingly, $x_i \ne \underline{x}_i$ (for $\underline{x}_i$) or $x_j = \overline{x}_j$ (for $\overline{x}_j$);
- the set J^+ of all the endpoints $\underline{x}_i$ and $\overline{x}_j$ for which we already know that for the optimal vector x we have, correspondingly, $x_i = \underline{x}_i$ (for $\underline{x}_i$) or $x_j \ne \overline{x}_j$ (for $\overline{x}_j$);
- the set J of the endpoints $\underline{x}_i$ and $\overline{x}_j$ for which we have not yet decided whether these endpoints appear in the optimal vector x.

In the beginning, $J^- = J^+ = \emptyset$ and J is the set of all $2n$ endpoints. At each iteration we also update the values $N^- = \#(J^-)$, $N^+ = \#(J^+)$, $E^- = \sum\limits_{\overline{x}_j \in J^-} \overline{x}_j$, and $E^+ = \sum\limits_{\underline{x}_i \in J^+} \underline{x}_i$. Initially, $N^- = N^+ = E^- = E^+ = 0$.

At each iteration we do the following.

- First we compute the median m of the set J.
- Then, by analyzing the elements of the undecided set J one by one, we divide them into two subsets

$$Q^- = \{x \in J : x \le m\}, \quad Q^+ = \{x \in J : x > m\}.$$

  We also compute $m^+ = \min\{x : x \in Q^+\}$.

- We compute $e^- = E^- + \sum\limits_{\overline{x}_j \in Q^-} \overline{x}_j$, $e^+ = E^+ + \sum\limits_{\underline{x}_i \in Q^+} \underline{x}_i$,

$$n^- = N^- + \#\{\overline{x}_j \in Q^-\}, \quad n^+ = N^+ + \#\{\underline{x}_i \in Q^+\},$$

  and $r = \dfrac{e^- + e^+}{n^- + n^+}$.

- If $r < m$, then we replace J^- with $J^- \cup Q^-$, E^- with e^-, J with Q^+, and N^- with n^-.
- If $r > m^+$, then we replace J^+ with $J^+ \cup Q^+$, E^+ with e^+, J with P^-, and N^+ with n^+.
- If $m \le r \le m^+$, then we replace J^- with $J^- \cup Q^-$, J^+ with $J^+ \cup Q^+$, J with $\emptyset$, E^- with e^-, E^+ with e^+, N^- with n^-, and N^+ with n^+.

At each iteration the set of undecided indices is divided in half. Iterations continue until all indices are decided. After this we return, as $\underline{V}$, the value of the population variance for the vector x for which:

- $x_j = \overline{x}_j$ for indices j for which $\overline{x}_j \in J^-$,
- $x_i = \underline{x}_i$ for indices i for which $\underline{x}_i \in J^+$, and
- $x_i = r$ for all other indices i.

4 Computing Entropy under Interval Uncertainty

4.1 Formulation of the Problem

For a probability distribution with probabilities $p_1, \ldots, p_n$ ($\sum p_i = 1$) the amount of uncertainty can be described by Shannon's entropy $S = -\sum\limits_{i=1}^{n} p_i \cdot \log(p_i)$. In practice we sometimes only know the intervals $\mathbf{p}_i = [\underline{p}_i, \overline{p}_i]$ of possible values of p_i. Different values $p_i \in \mathbf{p}_i$ lead to different S. It is therefore desirable to find the range $[\underline{S}, \overline{S}]$ of the entropy S [11].

Since the function S is concave, computation of $\overline{S}$ is feasible [12, 19]. However, computing $\underline{S}$ is NP-hard [21]. For the case when no interval $[\underline{p}_i, \overline{p}_i]$ is a proper subset of the interior of another interval $\mathbf{p}_j$ we have proposed an $O(n \cdot \log(n))$ algorithm for computing $\underline{S}$ [21].

In this paper we describe two new linear-time algorithms:

- a linear-time algorithm that computes $\overline{S}$ for all possible intervals, and
- a linear-time algorithm that computes $\underline{S}$ for intervals that satisfy the above no-subset property.

Comment. The new algorithms are similar to the above algorithms for computing $\underline{V}$ and $\overline{V}$. The main difference between the algorithms for variance and the algorithms for entropy is as follows. The variance $V(x_1, \ldots, x_n)$ is defined for all possible combinations $x_i \in [\underline{x}_i, \overline{x}_i]$. However, the entropy is only defined for values $p_i \in [\underline{p}_i, \overline{p}_i]$ that satisfy the additional constraint $\sum\limits_{i=1}^{n} p_i = 1$. As a result, while the maximum $\overline{V}$ of the variance V is attained when *each* value x_i attains one of its endpoints, the minimum $\underline{S}$ of the entropy S is generally attained when *all but one* values are endpoints. Indeed, it may not be possible to have $\sum\limits_{i=1}^{n} p_i = 1$ if we only use endpoints.

4.2 Linear-Time Algorithm for Computing $\underline{S}$ for Intervals that Satisfy the No-Subset Property

The proposed algorithm for computing $\underline{S}$ is similar to the algorithm for computing $\overline{V}$. The only difference is in the replacement part. Namely, once we computed m, P^-, P^+, e^-, and e^+, we do the following.

- If $e^- + e^+ > 1$, then we replace I^- with $I^- \cup P^-$, E^- with e^-, and I with P^+.
- If $e^- + e^+ + 2\Delta_m < 1$, then we replace I^+ with $I^+ \cup P^+$, E^+ with e^+, and I with P^-.
- Finally, if $e^- + e^+ \le 1 \le e^- + e^+ + 2\Delta_m$, then we replace I^- with $I^- \cup (P^- - \{m\})$, I^+ with $I^+ \cup P^+$, I with $\{m\}$, E^- with $e^- - \underline{p}_m$, and E^+ with e^+.

When computing $\overline{V}$, iterations continued until $I = \emptyset$. For $\underline{S}$ iterations continue until we have only one undecided index $I = \{k\}$. After this we return, as $\underline{S}$, the value of the entropy for the vector p for which $p_i = \underline{p}_i$ for $i \in I^-$, $x_j = \overline{p}_j$ for $j \in I^+$, and $p_k = 1 - e^- - e^+$ for the remaining value k.

Comment. A similar linear-time algorithm can be used to compute the expected value under interval uncertainty. In addition to computing the range for entropy, it is often useful to compute the range $[\underline{a}, \overline{a}]$ of the expected value $a = \sum a_i \cdot p_i$ of a known variable $(a_1, \ldots, a_n)$ under the constraints $p_i \in [\underline{p}_i, \overline{p}_i]$ and $\sum\limits_{i=1}^n p_i = 1$. For this problem, linear-time algorithms are known; see, e.g., [3, 9]. Let us show that this problem can be also solved by a simple modification of the above algorithm.

It is known that the smallest possible value $\underline{a}$ of the linear form $\sum\limits_{i=1}^n a_i \cdot p_i$ under given constraints is equal to $-\overline{b}$, where $\overline{b}$ is the largest possible value of the form $\sum\limits_{i=1}^n b_i \cdot p_i$, with $b_i = -a_i$. Thus it is sufficient to describe how to compute $\overline{a}$.

To compute $\overline{a}$ we follow the above iterative algorithm while it computes I^- and I^+. We continue iterations until we have only one undecided index $I = \{k\}$. After this we return, as $\overline{a}$, the value of the linear function $\sum\limits_{i=1}^n a_i \cdot p_i$ for the vector p for which $p_i = \underline{p}_i$ for $i \in I^-$, $x_j = \overline{p}_j$ for $j \in I^+$, and $p_k = 1 - e^- - e^+$ for the remaining value k.

4.3 Linear-Time Algorithm for Computing $\overline{S}$

The proposed algorithm is similar to the algorithm for computing $\underline{V}$; the only difference is that for computing $\overline{S}$, at each iteration, instead of computing $r = \dfrac{e^- + e^+}{n^- + n^+}$, we compute $r = \dfrac{1 - e^- - e^+}{1 - n^- - n^+}$.

5 Towards Fast Computation of the Range of Convex Symmetric Functions Under Interval Uncertainty

Computing the range of convex symmetric functions under interval uncertainty: formulation of the problem. In general, a statistical characteristic f can be more complex so that even computing f can take much longer than linear time. For such f, the question is how to compute the range $[\underline{y}, \overline{y}]$ in as few calls to f as possible. In this context, we can classify range-computing algorithms by this number of calls: it is reasonable to call an algorithm quadratic-time if it uses $O(n^2)$ calls, linear time if it uses $O(n)$ calls, etc.

In this section, we show that for a practically useful class of convex symmetric functions f, we can compute $\overline{y}$ in n calls to f – i.e., in the context of number of calls, in linear time.

Specifically, we consider continuous convex symmetric functions on convex symmetric sets $S \subseteq R^n$ containing a non-degenerate box $[\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$, with $\underline{x}_i < \overline{x}_i$ for all i.

A set $S \in R^n$ is called *symmetric* if with every point

$$x = (x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n) \in S$$

it also contain its arbitrary permutation; it is sufficient to require that for every i and j, the set S contain the corresponding transposition $\pi_{i,j}(x) \stackrel{\text{def}}{=} (x_1, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_n)$. A set S is called *convex* if for two points $x, x' \in S$ and for every real number $\alpha \in (0,1)$, the set S also contains the convex combination $\alpha \cdot x + (1-\alpha) \cdot x'$.

A function $f : S \to R$ is called *symmetric* if $f(x) = f(\pi_{i,j}(x))$ for every transposition $\pi_{i,j}$, and *convex* if $f(\alpha \cdot x + (a-\alpha) \cdot x') \le \alpha \cdot f(x) + (1-\alpha) \cdot f(x')$ for all $x, x' \in S$ and for all $\alpha \in (0,1)$.

Comment. It is known that each convex function defined on an open convex set is continuous; see, e.g., [2]. So, if, e.g., the set S coincides with entire space R^n, then we do not need to require continuity: any convex function $f : R^n \to R$ is automatically continuous.

Examples. Variance $\sum\limits_{i=1}^{n} (x_i - E)^2$ and entropy $\sum\limits_{i=1}^{n} -p_i \cdot \log(p_i)$ are examples of convex symmetric statistical characteristics. More general examples are higher-order even central moments $\sum\limits_{i=1}^{n} (x_i - E)^{2d}$ $(d = 1, 2, \ldots)$ and *generalized entropy functions*, i.e., functions $\sum\limits_{i=1}^{n} g(p_i)$ with convex $g(p)$.

Many important physical quantities outside statistics are also convex (or concave); see, e.g., [1, 2, 18, 20]. Some of these convex or concave characteristics are also symmetric.

Computational complexity: what is known. It is known that for convex functions, there exists a feasible (polynomial-time) algorithm for computing its minimum $\underline{y}$ (see, e.g., [2, 19]), but computing its maximum $\overline{y}$ is, in general, NP-hard [19]; as we have mentioned, it is even NP-hard for population variance. It is therefore desirable to find feasible algorithms that solve the maximum in practically reasonable situations. For variance and entropy, such algorithms are known for the case when the inputs satisfy the following *no-subset property*: $[\underline{x}_i, \overline{x}_i] \not\subseteq (\underline{x}_j, \overline{x}_j)$ for all $i \ne j$.

Algorithm for computing $\overline{y}$ with linear number of calls to f. The following algorithm computes the maximum $\overline{y}$ of a given continuous symmetric convex function $f(x_1, \ldots, x_n)$ over a given box $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ for all the cases in which the intervals $\mathbf{x}_i$ satisfy the no-subset property:

- First, we sort n intervals $\mathbf{x}_i$ in lexicographic order $\mathbf{x}_1 \le_{\text{lex}} \mathbf{x}_2 \le_{\text{lex}} \ldots \le_{\text{lex}} \mathbf{x}_n$.
- Second, for each k from 0 to n, we compute $f(s^{(k)})$, where $s^{(k)} \stackrel{\text{def}}{=} (\underline{x}_1, \ldots, \underline{x}_k, \overline{x}_{k+1}, \ldots, \overline{x}_n)$.
- Finally, we return the largest of $n+1$ values $f(s^{(k)})$ as $\overline{y}$.

This algorithm takes $O(n \cdot \log(n))$ steps for sorting (see, e.g., [4]), $n+1$ calls to f (to compute $n+1$ values $f(s^{(k)})$), and $O(n)$ steps to find the largest of these $n+1$ values. Thus, in addition to $n+1$ calls to f, this algorithm takes $O(n \cdot \log(n)) + O(n) = O(n \cdot \log(n))$ computational steps.

Comment. If the algorithm for computing the function f is feasible, i.e., takes a polynomial time $t \le P(n)$ for some polynomial $P(n)$, then computing $\overline{y}$ can be done in time $\le P(n) \cdot (n+1) + O(n \cdot \log(n))$ – i.e., also in polynomial time.

Comment. A function f is convex if and only if $-f$ is concave. The minimum of $-f$ is equal to minus the maximum of f. Thus, the above algorithm can also be used to compute the minima of continuous symmetric concave functions over a box $[\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$ whose intervals satisfy the no-subset property.

Possibility of a faster algorithm. The above algorithm for computing $\overline{y}$ is not always optimal. For example, computing the variance V takes linear time $C \cdot n$, so $n+1$ computations of variance means $(n+1) \cdot C \cdot n \approx C \cdot n^2$ time – while the linear-time algorithm for computing $\overline{V}$ (presented in Section 2) is much faster (for large n).

It turns out that a speed-up is possible not only for the variance V, but also for several other symmetric convex functions f.

Main idea behind the speed-up: some functions f are easy to revise. One of the reasons why we can speed up the computation of $\overline{V}$ is that this function is *easy to revise* in the following sense.

When we go from $s^{(k)}$ to $s^{(k+1)}$, we only change a single component s_{k+1} of the point s, from $\overline{x}_{k+1}$ to $\underline{x}_{k+1}$. Thus, if we keep the values $M \stackrel{\text{def}}{=} \frac{1}{n} \cdot \sum\limits_{i=1}^{n} x_i^2$ and $E = \frac{1}{n} \cdot \sum\limits_{i=1}^{n} x_i$, then updating each of these two values means computing the new values $E' = E - \frac{\overline{x}_{k+1} - \underline{x}_{k+1}}{n}$ and $M' = M - \frac{(\overline{x}_{k+1})^2 - (\underline{x}_{k+1})^2}{n}$, and then computing $V' = M' - (E')^2$. All these updates require a constant number (10) of arithmetic operations (independent on n). Thus, overall, we need $C \cdot n$ time to compute $V(s^{(0)})$ and time $10 \cdot n$ to compute n values $V(s^{(1)})$, ..., $V(s^{(n)})$. So, overall, we need time $C \cdot n + 10 \cdot n + O(n \cdot \log(n)) = O(n \cdot \log(n))$ which is, for large n, smaller than $C \cdot n^2$.

This idea can be applied to other "easy-to-revise" functions. To describe this result, let us first introduce two auxiliary notions: of a revised tuple and of a revisable computation scheme.

Revised tuples. For every tuple $x = (x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$, for every integer $i \le n$, and for every real number x_i', by a *revised tuple*, we mean a tuple $r_{i,x_i'}(x) \stackrel{\text{def}}{=} (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n)$, in which the i-th component of x is replaced by x_i'.

Revisable computation schemes. Let $f(x_1, \ldots, x_n)$ be a computable function. By a *revisable computation scheme* for computing $f(x_1, \ldots, x_n)$, we mean a tuple $\langle f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n), A_f, \Delta_f \rangle$, where:

- $f_1(x)$, ..., $f_m(x)$ are computable functions;
- A_f is an algorithm which, given n values $x = (x_1, \ldots, x_n)$, computes $f(x)$, $f_1(x)$, ..., $f_m(x)$;
- Δ_f is an algorithm which, given the values $f(x)$, $f_1(x)$, ..., $f_m(x)$, an integer i, and a real number x_i', computes the values $f(x')$, $f_1(x')$, ..., $f_m(x')$ for $x' = r_{i,x_i'}(x)$.

Comment. Of course, for every computable function f, there is a trivial revisable computation scheme for computing f: e.g., we can take $m = n$ and $f_i(x_1, \ldots, x_n) = x_i$ for all $i = 1, \ldots, m$. In this case, the algorithm A_f simply means computing $f(x)$ and also returning the original values $x_1, \ldots, x_m$, while the algorithm Δ_f ignores the previous values $f(x)$, $f_1(x)$, ..., $f_m(x)$, and simply computes the new value $f(x')$ (and also returns the new values $x_1', \ldots, x_n'$).

Easy-to-revise functions: a description. We are interested only in the computable functions f which are "easy-to-revise", i.e., for which there exists a revisable computation scheme in which

- the algorithm A_f has approximately the same computational complexity as the best known algorithm for simply computing $f(x)$, and
- the algorithm Δ_f is much faster than A_f.

Comment. In contrast to the notions of a revised tuple and a revisable computation scheme, the notion of an easy-to-revise function is somewhat informal: it depends on which algorithms for computing f we know, and on how we define "approximately the same" and "much faster".

Examples. Let us first explain why the variance $f = V$ is indeed easy-to-revise in the sense of the above (somewhat informal) definition. For the variance, $f_1 = M$ and $f_2 = E$. Computing $f_1 = M$, $f_2 = E$, and $f = M - E^2$ takes linear time $C \cdot n$ – approximately the same time as for all known algorithms for computing variance. However, if we change one of the components x_i to a different value x_i', then, as we have mentioned, updating E and M to new values E' and M' and computing the new value $V' = M' - (E')^2$ requires 10 computational steps. So, here, the revising algorithm Δ_f take 10 time steps, which, for large n, is much faster than $C \cdot n$.

Similarly, the mean E is easy-to-revise: we need time $C \cdot n$ to compute E, but only $3 \ll C \cdot n$ steps to update E.

Higher central even moments, entropy, and generalized entropy are other examples of easy-to-revise symmetric convex functions. For example, the 4-th central moment is a linear combination of the moments $M_k \overset{\text{def}}{=} \sum\limits_{i=1}^{n} x_i^k$ of orders $k = 1, 2, 3, 4$, and each of these four functions m_k is easy to revise.

Algorithm for computing $\overline{y}$ for easy-to-revise symmetric convex functions f. For easy-to-revise functions, we can compute $\overline{y}$ as follows:

- first, we apply the algorithm A_f to compute the values of f and of the auxiliary functions $f_1, \ldots, f_m$ at $s^{(0)}$;
- then, for each k from 1 to n, we apply the algorithm Δ_f to revise the values of $f, f_1, \ldots, f_m$ from $x = s^{(k-1)}$ to $x = s^{(k)}$;
- finally, we compute $\overline{y}$ as the largest of $n+1$ values $f(s^{(k)})$.

This algorithm calls the algorithm A_f once (to compute $f(s^{(0)})$, calls the revision algorithm Δ_f n times, and uses $O(n \cdot \log(n))$ computational steps in addition to these calls.

Comment. Since Δ_f is much faster than A_f, and A_f requires approximately the same time as computing f, the new algorithm for computing $\overline{y}$ takes much less time than computing f for $n+1$ tuples $s^{(0)}, \ldots, s^{(n)}$.

It is worth mentioning that this new algorithm is not always optimal: e.g., for the variance, this algorithm takes time $O(n \cdot \log(n)) + O(n) = O(n \cdot \log(n))$, but we know that we can compute $\overline{V}$ even faster: in linear time.

Computing $\underline{y}$ and $\overline{y}$ under the constraint $\sum\limits_{i=1}^{n} x_i = c$: a problem. As we have mentioned earlier, for entropy, we have an additional constraint $\sum\limits_{i=1}^{n} p_i = 1$ on the possible values of the probabilities p_i. The same constraint holds for computing other characteristics of probabilities such as a generalized entropy.

So, we arrive at the following problem: we know a continuous symmetric convex function $f(x_1, \ldots, x_n)$ (given as an algorithm or, equivalently, as a computer program), we know the intervals $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ that satisfy the above no-subset property, and we know the number c for which we should have $\sum\limits_{i=1}^{n} x_i = c$. Our objective is to compute the range

$$[\underline{y}, \overline{y}] = \left\{ f(x_1, \ldots, x_n) : x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n, \sum_{i=1}^{n} x_i = c \right\}.$$

Computing $\underline{y}$ under the constraint $\sum\limits_{i=1}^{n} x_i = c$. The constraints $x_i \in \mathbf{x}_i$ and $\sum\limits_{i=1}^{n} x_i = c$ describe a convex set, so we can compute the minimum $\underline{y}$ of a convex function f over this set in polynomial time.

Algorithm for computing $\overline{y}$ under the constraint $\sum\limits_{i=1}^{n} x_i = c$ when the intervals satisfy the no-subset property. Under the above no-subset property, the following algorithm computes $\overline{y}$ by calling f once and by using $O(n)$ computational steps in addition to this call.

This algorithm is iterative. At each iteration of this algorithm, we have three sets:

- the set I^- of all the indices i from 1 to n for we already know that for the optimal vector x, we have $x_i = \underline{x}_i$;
- the set I^+ of all the indices j for which we already know that for the optimal vector x, we have $x_j = \overline{x}_j$;

- the set $I = \{1, \ldots, n\} - I^- - I^+$ of the indices i for which we are still undecided.

In the beginning, $I^- = I^+ = \emptyset$ and $I = \{1, \ldots, n\}$. At each iteration, we also update the values of two auxiliary quantities $E^- \stackrel{\text{def}}{=} \sum\limits_{i \in I^-} \underline{x}_i$ and $E^+ \stackrel{\text{def}}{=} \sum\limits_{j \in I^+} \overline{x}_j$. In principle, we could compute these values by computing these sums, but to speed up computations, on each iteration, we update these two auxiliary values in a way that is faster than re-computing the corresponding two sums. Initially, since $I^- = I^+ = \emptyset$, we take $E^- = E^+ = 0$.

At each iteration, we do the following:

- first, we compute the median m of the set I $\Big($median in terms of sorting by $\widetilde{x}_i = \dfrac{\underline{x}_i + \overline{x}_i}{2}\Big)$;
- then, by analyzing the elements of the undecided set I one by one, we divide them into two subsets $X^- = \{i : \widetilde{x}_i \le \widetilde{x}_m\}$ and $X^+ = \{j : \widetilde{x}_j > \widetilde{x}_m\}$;
- we compute $e^- = E^- + \sum\limits_{i \in X^-} \underline{x}_i$ and $e^+ = E^+ + \sum\limits_{i \in X^+} \overline{x}_i$;
- if $e^- + e^+ > c$, then we replace I^- with $I^- \cup X^-$, E^- with e^-, and I with X^+;
- if $e^- + e^+ + 2\Delta_m < c$, then we replace I^+ with $I^+ \cup X^+$, E^+ with e^+, and I with X^-;
- finally, if $e^- + e^+ \le c \le e^- + e^+ + 2\Delta_m$, then we replace I^- with $I^- \cup (X^- - \{m\})$, I^+ with $I^+ \cup X^+$, I with $\{m\}$, E^- with $e^- - \underline{p}_m$, and E^+ with e^+.

At each iteration, the set of undecided indices is divided in half. Iterations continue until we have only one undecided index $I = \{k\}$, after which we return, as $\overline{y}$, the value of the function $f(x_1, \ldots, x_n)$ for the vector x for which $x_i = \underline{x}_i$ for $i \in I^-$, $x_j = \overline{x}_j$ for $j \in I^+$, and $x_k = c - E^- - E^+$ for the remaining value k.

6 Open Questions

In the paper, we describe a linear-time algorithm for computing the range of the variance V. Can similar linear-time algorithms be proposed for computing the endpoints of the intervals for the quantities $E - \alpha \cdot \sqrt{V}$ and $E + \alpha \cdot \sqrt{V}$ – which are important in detecting outliers [6, 13]? for computing other statistical characteristics – like moments or covariance?

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCC5-209, NSF grants EAR-0225670, DMS-0532645, and HRD-0734825, Star Award from the University of Texas System, and Texas Department of Transportation grant No. 0-5453.

The authors are greatly thankful to Pim van den Broek for important discussions and references, to Scott Ferson for valuable discussions, to the anonymous referees for important suggestions, and to David Novick and Matthew G. Averill for their encouragement and invaluable editorial help.

References

[1] S. Berberian, *Lectures in functional analysis and operator theory*, Springer, New York-Heidelberg-Berlin, 1974.

[2] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[3] P. van der Broek and J. Noppen, "Fuzzy weighted average: alternative approach", *Proceedings of the 25th International Conference of the North American Fuzzy Information Processing Society NAFIPS'2006*, Montreal, Quebec, Canada, June 3–6, 2006.

[4] Th. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[5] E. Dantsin, V. Kreinovich, A. Wolpert, and G. Xiang, "Population Variance under Interval Uncertainty: A New Algorithm", *Reliable Computing*, 2006, Vol. 12, No. 4, pp. 273–280.

[6] E. Dantsin, A. Wolpert, M. Ceberio, G. Xiang, and V. Kreinovich, "Detecting Outliers under Interval Uncertainty: A New Algorithm Based on Constraint Satisfaction", *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU'06*, Paris, France, July 2–7, 2006, pp. 802–809.

[7] S. Ferson, L. Ginzburg, V. Kreinovich, L. Longpré, and M. Aviles, Exact Bounds on Finite Populations of Interval Data, *Reliable Computing*, 2005, Vol. 11, No. 3, pp. 207–233.

[8] S. Ferson, V. Kreinovich, J. Hajagos, W. Oberkampf, and L. Ginzburg, *Experimental Uncertainty Estimation and Statistics for Data Having Interval Uncertainty*, Sandia National Laboratories, Report SAND2007-0939, May 2007.

[9] P. Hansen, M. V. P. de Aragao, and C. C. Ribeiro, "Hyperbolic 0-1 programming and optimization in information retrieval", *Math. Programming*, 1991, Vol. 52, pp. 255–263.

[10] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics*, Springer Verlag, London, 2001.

[11] G. J. Klir, *Uncertainty and Information: Foundations of Generalized Information Theory*, J. Wiley, Hoboken, New Jersey, 2005.

[12] V. Kreinovich, "Maximum entropy and interval computations", *Reliable Computing*, 1996, Vol. 2, No. 1, pp. 63–79.

[13] V. Kreinovich, L. Longpré, P. Patangay, S. Ferson, and L. Ginzburg, "Outlier Detection Under Interval Uncertainty: Algorithmic Solvability and Computational Complexity", *Reliable Computing*, 2005, Vol. 11, No. 1, pp. 59–76.

[14] V. Kreinovich, G. Xiang, S. A. Starks, L. Longpré, M. Ceberio, R. Araiza, J. Beck, R. Kandathi, A. Nayak, R. Torres, and J. Hajagos, "Towards combining probabilistic and interval uncertainty in engineering calculations: algorithms for computing statistics under interval uncertainty, and their computational complexity", *Reliable Computing*, 2006, Vol. 12, No. 6, pp. 471–501.

[15] P. Nivlet, F. Fournier, and J. Royer, A new methodology to account for uncertainties in 4-D seismic interpretation, *Proc. 71st Annual Int'l Meeting of Soc. of Exploratory Geophysics SEG'2001*, San Antonio, TX, September 9–14, 2001, 1644–1647.

[16] P. Nivlet, F. Fournier, and J. Royer, Propagating interval uncertainties in supervised pattern recognition for reservoir characterization, *Proc. 2001 Society of Petroleum Engineers Annual Conf. SPE'2001*, New Orleans, LA, September 30–October 3, 2001, paper SPE-71327.

[17] S. Rabinovich, *Measurement Errors: Theory and Practice*, American Institute of Physics, New York, 1993.

[18] A. W. Roberts and D. E. Varberg, *Convex functions*, Academic Press, New York and London, 1973.

[19] S. A. Vavasis, *Nonlinear Optimization: Complexity Issues*, Oxford Science, New York, 1991.

[20] R. Webster, *Convexity*, Oxford University Press, Oxford, New York, Tokyo, 1994.

[21] G. Xiang, O. Kosheleva, and G. J. Klir, "Estimating information amount under interval uncertainty: algorithmic solvability and computational complexity", *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU'06*, Paris, France, July 2–7, 2006, pp. 840–847.

Appendix: Proofs

Proof that the new algorithm for computing $\overline{V}$ requires linear time. At each iteration, computing median requires linear time, and all other operations with I require time t linear in the number of elements $|I|$ of I: $t \le C \cdot |I|$ for some C. We start with the set I of size n. On the next iteration, we have a set of size $n/2$, then $n/4$, etc. Thus, the overall computation time is $\le C \cdot (n + n/2 + n/4 + \ldots) \le C \cdot 2n$, i.e. linear in n.

Proof that under the no-subset property for narrowed intervals, the new algorithm always computes $\overline{V}$. Similarly to [14], one can easily show that since no two narrowed intervals are proper subsets of one another, they can be linearly ordered in lexicographic order. In this order we have $x_1^- \le x_2^- \le \ldots \le x_n^-$, $x_1^+ \le x_2^+ \le \ldots \le x_n^+$. Thus the averages $\widetilde{x}_i = (x_i^- + x_i^+)/2$ are also sorted: $\widetilde{x}_1 \le \widetilde{x}_2 \le \ldots \le \widetilde{x}_n$.

In [5] we have shown that in this sorting the value $\overline{V}$ is attained at one of the vectors $x^{(k)} = (\underline{x}_1, \ldots, \underline{x}_k, \overline{x}_{k+1}, \ldots, \overline{x}_n)$, i.e. that $V = V(x^{(k)})$ for some k.

In [5] we also analyzed the change in $V(x^{(k)})$ when we replace $x^{(k)}$ with $x^{(k-1)}$, i.e. when we replace $\underline{x}_k$ with $\overline{x}_k = \underline{x}_k + 2\Delta_k$. We have shown that $V_{k-1} - V_k = \dfrac{4\Delta_k}{n} \cdot (x_k^- - E_k)$, where $E_k \stackrel{\text{def}}{=} E(x^{(k)})$.

Hence $V_{k-1} < V_k$ if and only if $x_k^- < E_k$. Multiplying both sides of this inequality by n we get an equivalent inequality $x_k^- < n \cdot E_k$, where $n \cdot E_k = \sum\limits_{i=1}^{k} \underline{x}_i + \sum\limits_{j=k+1}^{n} \overline{x}_j$. Similarly $V_{k-1} > V_k$ if and only if $x_k^- > E_k$, and $V_{k-1} = V_k$ if and only if $x_k^- = E_k$.

When we go from k to $k+1$, we replace the larger value $\overline{x}_{k+1}$ in the sum $n \cdot E_k$ by a smaller value $\underline{x}_k$. Thus, the sequence $n \cdot E_k$ is strictly decreasing with k, while x_k^- is (maybe non-strictly) increasing with k. Therefore, once we have $n \cdot x_k^- < E_k$, i.e., $V_{k-1} < V_k$, these inequalities will hold for smaller k as well. Similarly, once we have $n \cdot x_k^- > E_k$, i.e., $V_{k-1} > V_k$, these inequalities will hold for larger k as well.

Once we have $n \cdot x_k^- = E_k$, i.e., $V_{k-1} = V_k$, then we will have $V_k > V_{k+1} > \ldots$ and $V_k = V_{k-1} > V_{k-2} > \ldots$, i.e. $V_k = V_{k-1}$ will be the largest value of V.

In other words, the sequence V_k first increases ($V_k > V_{k-1}$ for $k = 1, 2, \ldots$) and then starts decreasing ($V_k < V_{k-1}$ for larger k), with one or two top values.

For each m, if $V_{m-1} < V_m$ (i.e. if $n \cdot x_m^- < E_m$) this means that the value $k_{\max}$ corresponding to the maximum of V is $\le m$. Hence for all the indices $i \le m$ we already know that in the optimal vector x we have $x_i = \underline{x}_i$. Thus these indices can be added to the set I^-.

If $V_m > V_{m-1}$ (i.e. if $n \cdot x_m^- > E_m$) this means that the value $k_{\max}$ corresponding to the maximum of V is $> m$. Hence for all the indices $i > m$ we already know that in the optimal vector x we have $x_i = \overline{x}_i$. Thus these indices can be added to the set I^+.

Finally, if $V_m = V_{m-1}$ (i.e. if $n \cdot x_m^- = E_m$) then this m is where the maximum is attained.

The algorithm has been justified.

Proof that the new algorithm for computing $\underline{V}$ requires linear time. At each iteration, computing median requires linear time. All other operations with J require time t linear in the number of elements $|J|$ of J. We start with the set J of size n. On the next iteration, we have a set of size $n/2$, then $n/4$, etc. Thus, the overall computation time is $\le C \cdot (n + n/2 + n/4 + \ldots) \le C \cdot 2n$, i.e. linear in n.

Proof that the new algorithm always computes $\underline{V}$. In [7] we proved that if we sort all $2n$ endpoints into a sequence $x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)}$, then for some $k = k_{\min}$ the minimum $\underline{V}$ is attained for the vector x for which the following holds:

- For all indices j for which $\overline{x}_j \le x_{(k)}$ we have $x_j = \overline{x}_j$.
- For all indices i for which $\underline{x}_i \ge x_{(k+1)}$ we have $x_i = \underline{x}_i$.

- For all other indices i we have $x_i = r_k \overset{\text{def}}{=} \dfrac{E_k}{N_k}$, where

$$E_k = \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j + \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i; \quad N_k = \#\{j : \overline{x}_j \le x_{(k)}\} + \#\{i : \underline{x}_i \ge x_{(k+1)}\}.$$

It has also been proven that for the optimal k we have $r_k \in [x_{(k)}, x_{(k+1)}]$.

In general, the condition $x_{(k)} \le r_k = \dfrac{E_k}{N_k}$ is equivalent to

$$N_k \cdot x_{(k)} \le E_k = \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j + \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i.$$

Subtracting $x_{(k)}$ from each of N_k terms in the right-hand side (RHS) and moving the sum of the resulting non-positive differences into the left-hand side (LHS), we conclude that

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k)}). \tag{1}$$

When we increase k, we get (in general) more terms in the LHS and fewer in the RHS. Hence LHS (non-strictly) increases while the RHS non-strictly decreases. So if the inequality (1) holds for some k, it holds for all smaller values of k as well. Thus this inequality holds for all k until a certain value k_0.

Similarly, the condition $x_{(k+1)} \ge r_k = \dfrac{E_k}{N_k}$ is equivalent to

$$N_k \cdot r_{k+1} \ge \sum_{j:\overline{x}_j \le x_{(k)}} \overline{x}_j + \sum_{i:\underline{x}_i \ge x_{(k+1)}} \underline{x}_i.$$

Subtracting $x_{(k+1)}$ from each of N_k terms in RHS and moving the sum of the resulting non-positive differences into LHS, we conclude that

$$\sum_{j:\overline{x}_j \le x_{(k)}} (x_{(k+1)} - \overline{x}_j) \ge \sum_{i:\underline{x}_i \ge x_{(k+1)}} (\underline{x}_i - x_{(k+1)}). \tag{2}$$

When we increase k, the LHS (non-strictly) increases, while the RHS non-strictly decreases. So if the inequality (2) holds for some k, it holds for all larger values of k as well. Thus this inequality holds for all k after a certain value l_0.

So both conditions (1) and (2) are satisfied (which is equivalent to the condition $r_k \in [x_{(k)}, x_{(k+1)}]$) either for a single value $k_{\min}$ or for several sequential values $l_0, l_0 + 1, \ldots, k_0$. Let us show that if this condition is satisfied for several sequential values, this simply means that the same minimum $\underline{V}$ is attained for all these values. For that it is sufficient to show that if both conditions (1) and (2) holds for k and for $k + 1$ then the variance V has the same value for both k and $k + 1$. Indeed, since (1) is true for $k + 1$, we have

$$\sum_{j:\overline{x}_j \le x_{(k+1)}} (x_{(k+1)} - \overline{x}_j) \le \sum_{i:\underline{x}_i \ge x_{(k+2)}} (\underline{x}_i - x_{(k+1)}).$$

The LHS of this new inequality is smaller than or equal to the LHS of the inequality (2), and its RHS is larger than or equal to the RHS of the inequality (2). Thus the only way for both inequalities to hold is when both sides are equal, i.e. when replacing $x_{(k)}$ with $x_{(k+1)}$ and replacing $x_{(k+1)}$ with $x_{(k+2)}$ does not change which endpoints are in I^- and which are in I^+ – and thus, does not change the corresponding value of the variance.

So:

- for $k < k_{\min}$, we have $r_k > x_{(k+1)}$,
- for $k > k_{\min}$, we have $r_k < x_{(k)}$, and

- for $k = k_{\min}$ (or, to be more precise, for $l_0 \le k \le k_0$), we have $x_{(k)} \le r_k \le x_{(k+1)}$.

Hence:

- if $r_k < x_{(k)}$, then we cannot have $k < k_{\min}$ and $k = k_{\min}$, hence $k > k_{\min}$;
- if $r_k > x_{(k+1)}$, then we cannot have $k > k_{\min}$ and $k = k_{\min}$, hence $k < k_{\min}$;
- if $x_{(k)} \le r_k \le x_{(k+1)}$, then we cannot have $k < k_{\min}$ and $k > k_{\min}$, hence $k = k_{\min}$.

Thus the above algorithm finds the correct value of $k_{\min}$ and thence, the correct value of $\underline{V}$.

Proof that the new algorithm for computing $\underline{S}$ requires linear time is similar to the previous proofs.

Proof that under the no-subset property, the new algorithm always computes $\underline{S}$. Due to the no-subset property, we can sort the intervals in lexicographic order. In this case their lower endpoints $\underline{p}_i$, their upper endpoints $\overline{p}_i$, and their midpoints $\widetilde{p}_i$ are also sorted: $\underline{p}_i \le \underline{p}_{i+1}$, $\overline{p}_i \le \overline{p}_{i+1}$, and $\widetilde{p}_i \le \widetilde{p}_{i+1}$. Let us thus assume that the intervals are thus sorted.

Let us now show that it is sufficient to consider monotonic optimal tuples $p_1, \ldots, p_n$, for which $p_i \le p_{i+1}$ for all i. Indeed, if $p_i > p_{i+1}$ then, since $p_i \le \overline{p}_i \le \overline{p}_{i+1}$ and $p_i > p_{i+1} \ge \underline{p}_{i+1}$, we have $p_i \in [\underline{p}_{i+1}, \overline{p}_{i+1}]$ and similarly $p_{i+1} \in [\underline{p}_i, \overline{p}_i]$. Thus we can swap the values p_i and p_{i+1} without changing the value of S. We can repeat this swap as many times as necessary until we get a monotonic tuple that has the exact same value $S = \underline{S}$.

Let us now show that in the optimal tuple, at most one p_i can be inside the corresponding interval. Indeed, if we have two values p_j and p_k strictly inside their intervals, then for an arbitrary small Δ, replacing p_j with $p_j - \Delta$ and p_k with $p_k + \Delta = p + \Delta$ should not increase the resulting entropy. This is only possible when the derivative of the resulting expression w.r.t. Δ is 0, i.e. when $p_j = p_k$.

For $p_j - \Delta = p - \Delta$ and $p_k + \Delta = p + \Delta$ the function S should have a minimum at $\Delta = 0$. Thus its second derivative relative to Δ should be non-negative. However, an explicit computation shows that this derivative is negative. Thus our assumption is false, and at most one p_j can be inside the corresponding interval.

Since the values $\underline{p}_i$ are sorted by i and the values $\overline{p}_j$ are sorted by j, we can now conclude that:

- if $p_j = \underline{p}_j$ and $p_m > \underline{p}_m$, then $p_j \le p_m$; and
- if $p_m = \overline{p}_m$ and $p_j < \overline{p}_j$, then $p_m \ge p_j$.

Thus each value $p_j = \underline{p}_j$ precedes all the values $p_m = \overline{p}_m$, and the only value p_i which is strictly inside the corresponding interval lies in between these values. Hence in a monotonic optimal tuple $p_1, \ldots, p_n$ the first elements are equal to $\underline{p}_j$, then we may have one element which is strictly inside its interval, and then we have values $p_m = \overline{p}_m$.

For the resulting vector $p = (\underline{p}_1, \ldots, \underline{p}_{k-1}, p_k, \overline{p}_{k+1}, \ldots, \overline{p}_n)$, with $\underline{p}_k \le p_k \le \overline{p}_k$, the condition $\sum\limits_{i=1}^{n} p_i = 1$ implies that $\Sigma_k \le 1 \le \Sigma_{k-1}$, where $\Sigma_k \overset{\text{def}}{=} \sum\limits_{i=1}^{k} \underline{p}_i + \sum\limits_{j=k+1}^{n} \overline{p}_j$. When we go from Σ_k to Σ_{k+1}, we replace a larger value $\overline{p}_{k+1}$ with a smaller value $\underline{p}_{k+1}$. Hence $\Sigma_k > \Sigma_{k+1}$. Thus there has to be exactly one $k_{\max}$ for which $\Sigma_k \le 1 \le \Sigma_{k-1}$.

So if we have $\Sigma_m > 1$, this means that the value $k_{\max}$ corresponding to the minimum of S is $> m$. Hence for all the indices $i \le m$ we already know that in the optimal vector p we have $p_i = \underline{p}_i$. Thus these indices can be added to the set I^-.

If Σ_{m-1} $(= \Sigma_m + 2\Delta_m) < 1$, this means that the value $k_{\min}$ corresponding to the minimum of S is $< m$. Hence for all the indices $j \ge m$ we already know that in the optimal vector p we have $p_j = \overline{p}_j$. Thus these indices can be added to the set I^+.

Finally, if $\Sigma_m \le 1 \le \Sigma_{m-1}$ then this m is where the minimum of S is attained.

The algorithm has been justified.

Proof that the new algorithm for computing $\overline{S}$ requires linear time is similar to the previous proofs.

Proof that the new algorithm always computes $\overline{S}$. It is known [11, 12, 21] that if we sort all $2n$ endpoints into a sequence $p_{(1)} \le p_{(2)} \le \ldots \le p_{(2n)}$, then for some $k = k_{\max}$ the maximum $\overline{S}$ is attained for the vector p for which the following holds:

- For all indices j for which $\overline{p}_j \le p_{(k)}$, we have $p_j = \overline{p}_j$.
- For all indices i for which $\underline{p}_i \ge x_{(k+1)}$, we have $p_i = \underline{p}_i$.
- For all other indices, we have $p_i = \text{const}$. Since $\sum\limits_{i=1}^{n} p_i = 1$, we conclude that this constant is equal to $r_k \stackrel{\text{def}}{=} \dfrac{1 - E_k}{n - N_k}$, where
$$E_k = \sum_{j:\overline{p}_j \le p_{(k)}} \overline{p}_j + \sum_{i:\underline{p}_i \ge p_{(k+1)}} \underline{p}_i;$$
$$N_k = \#\{j : \overline{p}_j \le p_{(k)}\} + \#\{i : \underline{p}_i \ge p_{(k+1)}\}.$$

It can also be proven that for the optimal k we have $r_k \in [p_{(k)}, p_{(k+1)}]$. These facts can proven by the same analysis (adding Δp to one value p_j and subtracting Δp from another value p_k) as in our above analysis of $\underline{S}$.

Let us first prove that if $r_k = \dfrac{1 - E_k}{n - N_k} \le p_{(k+1)}$ then the similar inequality $r_{k+1} = \dfrac{1 - E_{k+1}}{n - N_{k+1}} \le p_{(k+2)}$ holds for the next value k. Indeed, the given inequality $\dfrac{1 - E_k}{n - N_k} \le p_{(k+1)}$ is equivalent to $1 - E_k \le (n - N_k) \cdot p_{(k+1)}$.

The only difference between the sums $E_k = \sum\limits_{j:\overline{p}_j \le p_{(k)}} \overline{p}_j + \sum\limits_{i:\underline{p}_i \ge p_{(k+1)}} \underline{p}_i$ and $E_{k+1} = \sum\limits_{j:\overline{p}_j \le p_{(k+1)}} \overline{p}_j + \sum\limits_{i:\underline{p}_i \ge p_{(k+2)}} \underline{p}_i$ is that:

- some terms equal to $p^{(k+1)}$ may be added (if there are j for which $\overline{p}_j = p_{(k+1)}$), and
- some other terms equal to to $p^{(k+1)}$ may be subtracted (if there are i for which $\underline{p}_i = p_{(k+1)}$).

In general, $E_{k+1} = E_k + c_k \cdot p_{(k+1)}$ for some integer c_k (positive, negative, or zero), and $N_{k+1} = N_k + c_k$. Subtracting $c_k \cdot p_{(k+1)}$ from both sides of the given inequality $1 - E_k \le (n - N_k) \cdot p_{(k+1)}$, we conclude that $1 - E_{k+1} \le (n - N_{k+1}) \cdot p_{(k+1)}$, i.e. that $r_{k+1} = \dfrac{1 - E_{k+1}}{n - N_{k+1}} \le p_{(k+1)}$. Since the sequence $p_{(k)}$ is sorted, we thus conclude that $p_{(k+1)} \le p_{(k+2)}$ and hence $r_{k+1} \le p_{(k+2)}$.

So if the inequality $r_k \le p_{(k+1)}$ holds for some k, it holds for all larger values of k as well. Thus this inequality holds for all k after a certain value l_0.

Similarly, we can prove that if the inequality $r_k \ge p_{(k)}$ holds for some k, then it holds for $k - 1$ as well – since the only difference between E_k and E_{k-1} consists of adding and/or subtracting some values $p_{(k)}$. So if the inequality $r_k \ge p_{(k)}$ holds for some k, it holds for all smaller values of k as well. Thus, this inequality holds for all k until a certain value k_0.

Similarly to the proof about $\underline{V}$, we can prove that if there are several values $k = l_0, l_0 + 1, \ldots, k_0$ for which both inequalities hold $p_{(k)} \le r_k \le p_{(k+1)}$, then for these k, the entropy has exactly the same value.

So:

- for $k < k_{\max}$, we have $r_k > p_{(k+1)}$,
- for $k > k_{\max}$, we have $r_k < p_{(k)}$, and
- for $k = k_{\max}$ (or, to be more precise, for $l_0 \le k \le k_0$), we have $p_{(k)} \le r_k \le p_{(k+1)}$.

Hence:

- if $r_k < p_{(k)}$, then we cannot have $k < k_{\max}$ and $k = k_{\max}$, hence $k > k_{\max}$;
- if $r_k > p_{(k+1)}$, then we cannot have $k > k_{\max}$ and $k = k_{\max}$, hence $k < k_{\max}$;
- if $p_{(k)} \le r_k \le p_{(k+1)}$, then we cannot have $k < k_{\min}$ and $k > k_{\min}$, hence $k = k_{\max}$.

Thus, the above algorithm finds the correct value of $k_{\max}$ and thence, the correct value of $\overline{S}$.

Proof that the new algorithm for computing the maximum $\overline{y}$ of a symmetric convex function is correct. To prove that the algorithm is correct we must show that the maximum $\overline{y}$ of the function f is attained at one of the points $s^{(k)}$.

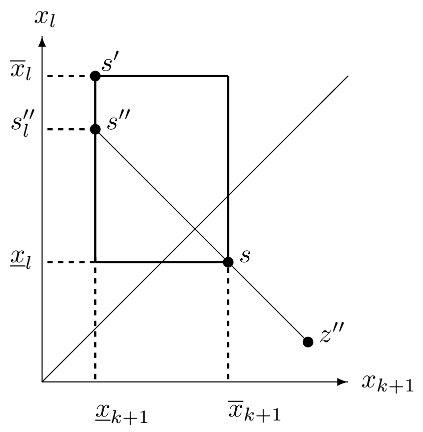
One can easily check that since the intervals $\mathbf{x}_i$ are already sorted in lexicographic order and satisfy the no-subset property, the lower endpoints and the upper endpoints are also sorted, i.e., $\underline{x}_i \le \underline{x}_{i+1}$ and $\overline{x}_i \le \overline{x}_{i+1}$ for all i.

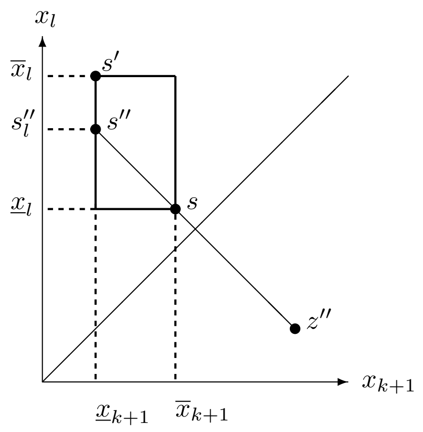
The maximum of a continuous function on a bounded closed polyhedron $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$ is always attained at some point, and since the function f is convex, it is attained at one of the vertices of this set [2, 18, 20], i.e., when for each i, we have $s_i = \underline{x}_i$ or $s_i = \overline{x}_i$.

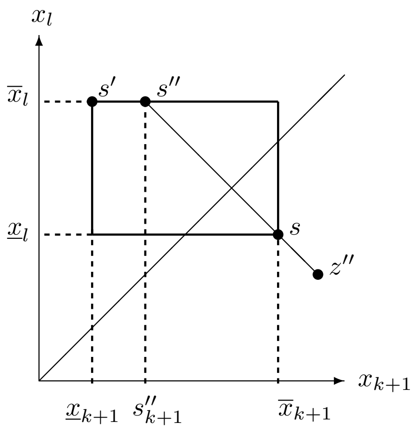
There may be several vertices at which the maximum is attained. Out of all maximizing vertices, we choose a one s with the largest length of the starting sequence of lower bounds. We will denote this length by k; this means that s has the form $s = (\underline{x}_1, \ldots, \underline{x}_k, \overline{x}_{k+1}, \ldots)$, i.e., it starts with k lower bounds and then has an upper bound at the $(k+1)$-st place. We will prove that for this point s, all the components s_l for $l > k+1$ are upper bounds, i.e., that $s = s^{(k)}$.

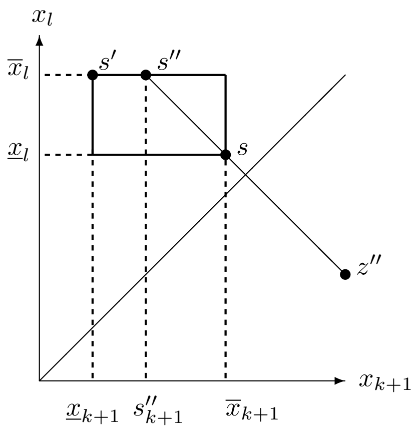
Indeed, let us assume that for some $l > k+1$, the component s_l of the chosen point is a lower bound: $s_l = \underline{x}_l$. We will then construct another point s' at which f also attains its maximum and which has a longer starting sequence of lower bounds – which contradicts to our choice of s.

In this construction, we will only change the $(k+1)$-st and l-th coordinates, so this construction can be naturally illustrated on the corresponding plane. First, we consider the bisecting line $x_{k+1} = x_l$ of the first and third quadrant and find an orthogonal line to it ($x_{k+1} + x_l = \text{const}$) which passes through s. The line has to intersect the interior of the rectangle and to leave it again at some point s'' – which is either the left or the upper face; see the following pictures, in which $s''_l \overset{\text{def}}{=} \underline{x}_l + (\overline{x}_{k+1} - \underline{x}_{k+1})$ and $s''_{k+1} \overset{\text{def}}{=} \overline{x}_{k+1} - (\overline{x}_l - \underline{x}_l)$. Let z'' be a point which is symmetric relative to s''. Since the endpoints are sorted, one can prove that s is in between s'' and z'', i.e., that s is a convex combination of s'' and z''.









The point s at which the maximum is attained is a convex combination of the points s'' and z'', i.e., $s = \alpha \cdot s'' + (1 - \alpha) \cdot z''$. Since f is a convex function, we have $f(s) \le \alpha \cdot f(s'') + (a - \alpha) \cdot f(z'')$. Due to symmetry, we have $f(s'') = f(z'')$ hence $f(s) \le f(s'')$. Since the function f attains its maximum at the point s, it thus attains the maximum at the point s'' as well.

This point s'' is on a straight line segment – namely, on one of the faces of the rectangle. Since the function f is convex, the only way for it to attain the maximum inside the straight line segment is to attain the same maximum on both endpoints of this face, in particular, at a points s' at which $s'_{k+1} = \underline{x}_{k+1}$. For this new point, we have $s'_1 = \underline{x}_1, \ldots, s'_k = \underline{x}_k$, and $s'_{k+1} = \underline{x}_{k+1}$ – which contradicts to our assumption that k is the largest length of the starting lower-endpoint sequence in a maximizing point. Correctness is proven.

Proof that the new algorithm for computing $\overline{y}$ under the constraint $\sum\limits_{i=1}^{n} x_i = c$ is correct. Similarly to the previous proof, we can conclude that the maximum $\overline{y}$ is always attained at one of the vertices of the convex polyhedron $(\mathbf{x}_1 \times \ldots \times \mathbf{x}_n) \cap \left\{x : \sum\limits_{i=1}^{n} x_i = c\right\}$, i.e., at a point s at which for at least $n-1$ values s_i, we have $s_i = \underline{x}_i$ or $s_i = \overline{x}_i$.

Similarly to the previous proof, we can also conclude that the maximum is attained at one of the points $s^{(k)} = (\underline{x}_1, \ldots, \underline{x}_{k-1}, s_k, \overline{x}_{k+1}, \ldots, \overline{x}_n)$. The value s_k can determined by the condition $\sum\limits_{i=1}^{n} s_i = c$. For this value s_k to be between $\underline{x}_k$ and $\overline{x}_k$, we must make sure that $\Sigma_k \le 1 \le \Sigma_{k-1}$, where $\Sigma_k \overset{\text{def}}{=} \sum\limits_{i=1}^{k} \underline{x}_i + \sum\limits_{j=k+1}^{n} \overline{x}_j$. Similar to the case of entropy, we can conclude that since the sums Σ_k decrease with k, there is only one value k for which the above inequality holds, so we can use the linear-time algorithm from the entropy case to find this value k. Once this value is found, we have thus found the maximizing point $s^{(k)}$ and thus, a single call to f finds the desired maximum $f(s^{(k)})$.

Correctness is proven.