# Interval Methods for Presenting Performance of Financial Trading Systems

Guido Deboeck, Karen Villaverde, Vladik Kreinovich

***Abstract*— Even successful financial funds do not always grow monotonically; one in a while small downfalls happen. These downfalls are perceived badly by potential customers. So, it is desirable to present the history of the fund's behavior as the sum of a monotonic trend and of a fluctuating stochastic part. The smaller the stochastic part, the larger the trend. So, in the best presentation corresponds to the smallest stochastic part. In this paper, we describe algorithms that produce the best presentation.**

## I. Formulation of the Problem in Informal Terms

Even successful finanical funds do not always grow monotonically: once in a while small downfalls happen. These downfalls (*underwater periods*) are perceived badly by potential customers. So, it is desirable to present the history of the fund's behavior as the sum of a monotonic trend and of a fluctuating (*stochastic*) part.

The smaller the stochastic part, the larger the trend. So, the best presentation corresponds to the smallest stochastic part. The problem of finding the best presentation was formulated in [1].

In this paper, we describe algorithms that produce the best presentation.

## II. How This Problem is Related to Interval Computations

Our problem is: given a function, to find the close monotonic one. A similar problem has been considered and solved in [4, 3, 5]: Suppose that we have an unknown physical dependency $y(x)$. We measure $y$ for $x = x_1, ..., x_n$, and get measurement results $\tilde{y}_1, ..., \tilde{y}_n$. Measurements are never absolutely accurate. Therefore, from the fact the measured value is $\tilde{y}_i$, we can conclude that the actual value of $y(x_i)$ belongs to the interval $[\tilde{y}_i - \Delta, \tilde{y}_i + \Delta]$, where $\Delta$ is the accuracy of the measuring instrument that is guaranteed by the manufacturer of this instrument. The problem is: given $x_i$, $\tilde{y}_i$, and $\Delta$, to check whether it is possible that the actual dependency is monotonic, and, if the dependency is definitely not monotonic (i.e., if it always has local extrema), to find possible locations of these local extrema.

Our new problem is *different* from this one: we do not know $\Delta$, and we want the resulting monotonic dependency not only to approximate the process, but to be always *below* the actual values. However, we can modify methods from [4, 3, 5] to solve this problem as well.

## III. Formulation of the Problem in Mathematical Terms

**Definition.**

- By a *price sequence*, we mean a sequence $y$ of $n$ real numbers $y_1, ..., y_n$.
- By a *presentation* of a sequence $y$, we mean a sequence $z = (z_1, ..., z_n)$ such that:
  - $z_i \le y_i$ for all $i$, and
  - $z_i$ is non-decreasing, i.e., $z_i \le z_j$ for $i < j$.
- By a *stochastic part* of the presentation, we mean

$$\max_i (y_i - z_i).$$

- The presentation $z$ of a sequence $y$ is called the *best* if its stochastic part is the smallest possible.

**PROBLEM:**
**Given:** a price sequence $y$.
**To find:** the best presentation for $y$ (and to estimate its stochastic part).

## IV. Solution and Algorithms

**PROPOSITION 1.** *For every price sequence, its best presentation is* $z_i = \min(y_i, y_{i+1}, ..., y_n)$.

This formula leads to a natural linear-time algorithm (i.e., an algorithm whose running time is

The authors are with The World Bank, 1818 H Street, N.W., Washington, DC 20433, email gdeboeck@worldbank.org (G. J. Deboeck), with Systems Engineering, Bell Northern Research, P.O. Box 833871 M\S D0-207, Richardson, TX 75083-3871, email villa@bnr.ca (K. Villaverde) and with Computer Science Department, The University of Texas at El Paso, El Paso, TX 79968, email vladik@cs.utep.edu (V. Kreinovich). This work was partially supported by NSF grant No. CDA-9015006 and NASA Research Grant No. 9-757.

bounded by the linear function $C \cdot n$ of the size of the input):

**PROPOSITION 2.** *There exists a linear-time algorithm that computes both the best presentation and its stochastic part.*

If we have several processors that can work in parallel, then we can compute the stochastic part faster:

**PROPOSITION 3.** *There exists an algorithm that given a price sequence $y$, returns the stochastic part of its best presentation. This algorithm uses $n$ processors and require $C \cdot \log(n)$ computation time.*

**PROPOSITION 4.** *There exists an algorithm that, given a price sequence $y$, returns the best presentation $z$ for $y$. This algorithm uses $n^2$ processors and requires $C \cdot \log(n)$ computation time.*

## V. Proofs

### A. Proof of Proposition 1

The sequence $z_i = \min(y_i, y_{i+1}, ..., y_n)$ is a presentation: its monotonicity is evident, and from $z_i = \min(y_i, ...) \le y_i$, it follows that $z_i \le y_i$.

Let us show that $z$ is the best presentation. Indeed, let $t_i$ be any other presentation. Then, $t_i \le y_i$ for all $i$. Since $t_i$ is non-decreasing, we also have $t_i \le t_j$ for all $j \ge i$. But for these $j$, we have $t_j \le y_j$. Therefore, $t_i \le y_j$ for all $j \ge i$, i.e., all the values $y_i, y_{i+1}, ..., y_n$ are bounds for $t_i$. Hence, $t_i$ is bounded by the smallest of these bounds, i.e., $t_i \le \min(y_i, y_{i+1}, ..., y_n) = z_i$. From $t_i \le z_i$, we can conclude that for every $i$, $y_i - z_i \ge y_i - t_i$ and therefore, $\max(y_i - z_i) \le \max(y_i - t_i)$. So, $z$ indeed has the smallest stochastic part and is therefore, the best presentation. Q.E.D.

### B. Proof of Proposition 2

To formulate this algorithm, we must slightly reformulate the result of Proposition 1: namely, we can expres it as $z_{i-1} = \min(z_i, y_{i-1})$. This reformulation enables us to formulate the desired linear-time algorithm:

To get $z_i$, we process the values $y_i$ in the inverse order: $y_n$, $y_{n-1}$, ..., $y_1$. After processing each number $y_i$, we will have $z_i$ and the current estimate $sp$ for the stochastic part.

- We start with $z_n = y_n$, and $sp = 0$.
- If we already have $z_i$ and $sp$, and we read the next value $y_{i-1}$, then we compute
$$z_{i-1} = \min(z_i, y_{i-1}),$$
and update the value of $sp$ as follows:
$$sp := \max(sp, y_{i-1} - z_{i-1}).$$

After we have processed all the numbers, $sp$ will take the desired value. Each processing requires a constant number of steps, so, the total computation time is linear. Q.E.D.

### C. Proof of Proposition 3

To explain this algorithm, we need a different reformulation of the result of Proposition 1. Namely, the stochastic part $sp$ is defined as the largest of the numbers $y_i - z_i = y_i - \min(y_i, y_{i+1}, ..., y_n)$. Inversion changes the order, so,

$$-\min(y_i, y_{i+1}, ..., y_n) = \max(-y_i, -y_{i+1}, ..., -y_n),$$

and hence,

$$y_i - z_i = \max(y_i - y_i, y_i - y_{i+1}, ..., y_i - y_n),$$

i.e., $y_i - z_i$ is the largest of 0 and the numbers $y_i - y_j$ for $i < j$. The desired value $sp$ is the largest of all such maxima, so, it is the largest of 0 and all differences $y_i - y_j$ for all $i < j$:

$$sp = \max[0, \max_{i,j:i<j}(y_i - y_j)].$$

So, in order to compute $sp$, it is sufficient to be able to compute the value

$$\Delta = \max_{i,j:i<j}(y_i - y_j).$$

From $\Delta$, we can compute $sp$ in one computation step: $sp = \max(0, \Delta)$.

This new reformulation enables us to parallelize the computation of $\Delta$. Indeed, suppose that we have divided the time interval $P = \{1, 2, ..., n\}$ into two parts:

- part $P_1$ that contains all the moments from 1 to some $k$, and
- part $P_2$ that contains all the moments from $k+1$ to $n$.

We can now divide the pairs $(i, j)$ with $i < j$ into three groups, depending on what part $i$ and $j$ belong to (the fourth group with $i \in P_2$ and $j \in P_1$, contains no pairs, because every element of $P_1$ is smallesr than every element form $P_2$):

- $i, j \in P_1$;
- $i \in P_1$, and $j \in P_2$;
- $i, j \in P_2$.

Therefore, we can represent $\Delta$ as

$$\Delta = \max(\Delta(P_1), \Delta_{12}, \Delta(P_2)), \qquad (1)$$

where we have denoted:

$$\Delta(P_1) = \max_{i,j \in P_1 : i<j}(y_i - y_j);$$

$$\Delta_{12} = \max_{i \in P_1, j \in P_2} (y_i - y_j);$$

$$\Delta(P_2) = \max_{i,j \in P_2 : i < j} (y_i - y_j).$$

In the formula for $\Delta_{12}$, we omitted the condition $i < j$, because this condition is always true for $i \in P_1$ and $j \in P_2$.

The formula for $\Delta_{12}$ can be reformulated as follows: the largest value of the difference is attained if the number $y_i$ from which we subtract takes the largest value, and the number $y_j$ that is subtracted takes the smallest possible one. Therefore,

$$\Delta_{12} = \max_{i \in P_1} y_i - \min_{j \in P_2} y_j. \tag{2}$$

Substituting (2) into (1), we get the following formula:

$$\Delta(P) = \max(\Delta(P_1), M(P_1) - m(P_2), \Delta(P_2)), \tag{3}$$

where the values

$$M(P) = \max_{i \in P} y_i$$

and

$$m(P) = \min_{i \in P} y_i$$

can be computed as follows:

$$M(P) = \max(M(P_1), M(P_2)); \tag{4}$$

$$m(P) = \max(m(P_1), m(P_2)). \tag{5}$$

Formulas (3–5) enable us to reduce the problem of computing $\Delta$, $M$, and $m$ for an interval of size $n$ to computing similar values for the two halves (of size $n/2$). Computation of both halves can be done in parallel. We can therefore use this bisection to parallelize the computation of $\Delta$ (this *bisection* is one of the standard parallelization tricks, see, e.g., [2]).

The actual parallel algorithm is as follows:

- First, we divide $n$ elements into groups of two: (1,2), (3,4), ..., and for each group $G = (i, i+1)$, we compute the minimum $m(G) = \min(y_1, y_{i+1})$, the maximum $M(G) = \max(y_1, y_{i+1})$, and $\Delta(G) = y_i - y_{i+1}$. These computations are done simultaneously on $n/2$ processors.
- Then, we group these pairs into groups of four elements, and use formulas (3–5) to compute the values of $m$, $M$, and $\Delta$ for each group of four (based on the known values for pairs). These computations are also done in parallel.
- Then, we group these groups of four into groups of $2^3 = 8$, etc.

On $k-$th stage, we get groups of size $2^k$. So, after $\approx \log_2(n)$ stages, we get the desired $\Delta(P)$ for the original price sequence $y$.

Each stage requires a constant time, so, the total time of this algorithm is bounded by a constant times the number of stages, i.e., by $C \cdot \log(n)$. Q.E.D.

*Comment.* If we do not have $n$ processors, and we only have $p$, then, we can start with $p$ groups of size $n/p$, for each of which we compute $\Delta$, $M$, and $m$, using a linear time algorithm from Proposition 1. For this modification, similarly to [5], we get the following result:

**PROPOSITION 5.** *Suppose that we have $p$ processors working in parallel. Then, there exists an algorithm that, given a price sequence $y$ of size $n$, computes the stochastic part of $y$. This algorithm requires $C(n/p) + C\log(p)$ computation time.*

*D. Proof of Proposition 4*

The stochastic part can be computed as in Proposition 3.

As for the best presentation itself, in [5], we have described how use $n^2$ processors to compute $\min(y_i, y_{i+1}, ..., y_j)$ for all $i$ and $j$ in $\le C \cdot \log(n)$ time. In partcular, this algorithm enables us to compute the desired values $z_i = \min(y_i, y_{i+1}, ..., y_n)$. Q.E.D.

## References


[1] G. Deboeck, "Fuzzy logic, newral networks, and genetic algorithms for finanial trading systems", *NAFIPS/IFIS/NASA'94, Proceedings of the First International Joint Conference of The North American Fuzzy Information Processing Society Biannual Conference, The Industrial Fuzzy Control and Intelligent Systems Conference, and The NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic, San Antonio, December 18–21, 1994* (plenary talk).

[2] J. Jájá, **An introduction to parallel algorithms**, Addison-Wesley, Reading, MA, 1992.

[3] K. Villaverde, "How to locate maxima and minima of a function in parallel from approximate measurement results", In: V. Kreinovich, B. Traylor, R. Watson (eds.), *Abstracts of the First UTEP Computer Science Department Students Conference*, El Paso, TX, 1991, pp. 43–44.

[4] K. Villaverde and V. Kreinovich, "A linear-time algorithm that locates local extrema of a function of one variable from interval measurement results," *Interval Computations*, 1993, No. 4, pp. 176–194.

[5] K. Villaverde and V. Kreinovich, "Parallel Algorithms That Locate Local Extrema of a Function of One Variable From Interval Measurement Results", *These Proceedings.*