

# A Quadratic-Time Algorithm For Smoothing Interval Functions

Vladik Kreinovich and Karen Villaverde

## Abstract

In many real-life applications, physical considerations lead to the necessity to consider the smoothest of all signals that is consistent with the measurement results. Usually, the corresponding optimization problem is solved in statistical context. In this paper, we propose a quadratic-time algorithm for smoothing an *interval* function. This algorithm, given  $n + 1$  intervals  $\mathbf{x}_0, \dots, \mathbf{x}_n$  with  $0 \in \mathbf{x}_0$  and  $0 \in \mathbf{x}_n$ , returns the vector  $x_0, \dots, x_n$  for which  $x_0 = x_n = 0$ ,  $x_i \in \mathbf{x}_i$ , and  $\sum (x_{i+1} - x_i)^2 \rightarrow \min$ .

## 1 Motivations

### 1.1 Smoothing is Important

In engineering, astronomy, geophysics, etc, transmitted signals and/or images are “corrupted” by noise; this “noise” includes the unpredictable additions from other sources as well as the changes in the signal that are due to the imperfection of the measuring instrument. As a result of this “corruption”, we cannot uniquely reconstruct the signal from the measurement results: to reconstruct the signal, we have to subtract the (unknown) noise from the measured values; therefore, several possible signals could be responsible for the same measured values. Which of these possible signals should we present to the user?

It turned out that in many real-life situations (voice recognition, astronomical and geophysical imaging, etc.; see, e.g., [9, 3, 2, 4, 5, 8]), physical arguments favor the choice of the *smoothest* possible signal. In the simplest case of a 1-D signal (i.e., a signal  $x(t)$  that depends on only one parameter  $t$ ), the non-smoothness  $S_c(x)$  of a continuous signal  $x(t)$  is usually described by a formula  $S_c(x) = \int (\dot{x}(t))^2 dt$ , where  $\dot{x}$  denotes the derivative of the function  $x$  (see, e.g., [7] for one of the possible justifications of this formula).

In real-life computations, we usually only reconstruct the values  $x_i = x(t_i)$  of the signal on a grid, i.e., for the values  $t_0, t_1 = t_0 + h, \dots, t_k = t_0 + kh, \dots, t_n = t_0 + nh$ . Therefore, we need to reformulate the non-smoothness criterion in

terms of these finitely many values. The natural approximation to a derivative is a ratio  $\Delta x/\Delta t = (x_{i+1} - x_i)/h$ ; therefore, it is natural to take

$$S(\vec{x}) = \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 \tag{1}$$

as a measure of non-smoothness of the reconstructed signal (strictly speaking, we should use  $S/h^2$ , but since we are only interested in choosing the smoothest signal, and not in the estimating its non-smoothness, we can afford to simplify the smoothness expression by multiplying it with a constant  $h^2$ ).

## 1.2 Several Statistical Smoothing Algorithms are Known

There exist many algorithms for selecting the smoothest signal, and there are many successful applications of these algorithms (see, e.g., [9, 3, 2, 4, 5, 8]). However, practically all these algorithms are *statistical*: they are based on the assumption that we know the *probabilities* of different values of error.

## 1.3 Since We Often Do Not Know Probabilities, Interval Smoothing is Necessary

In real life, we often do not know these probabilities, we only know the *intervals* of possible values of the errors. As a result, for each  $i$ , we only know the *interval*  $\mathbf{x}_i = [x_i^-, x_i^+]$  of possible values of the measured quantity  $x_i$ . The problem is: to find the smoothest sequence  $x_0, \dots, x_n$  that is consistent with these measurement results (i.e., for which  $x_i \in \mathbf{x}_i$  for all  $i$ ). In this paper, we will present a new quadratic-time algorithm for solving this problem.

*Comment.* In order to formulate the problem in precise terms, we must make one more remark. Usually, when we start reconstructing the signal, we have the complete record of this signal. In other words, we know that in one or several first moments of time, there was no signal, and that in the (one or several) last moments of time, there will also be no signal at all. In mathematical terms, we assume that  $x_0 = x_n = 0$ . This assumption is consistent only if  $0 \in \mathbf{x}_0$  and  $0 \in \mathbf{x}_n$  (if these inclusions do not occur, this means that either we stopped measuring too early, when the signal has not yet passed, or that we started measuring too late, when the signal was already being sent).

Now, we are ready to formulate the problem and the algorithm.

## 1.4 The Structure of the Paper

The problem itself and the algorithms for solving it will be presented in Section 2. In Section 3, we illustrate our algorithms on a numerical example. The proof that this algorithm actually finds the smoothest signal is given in Section 4. In the last (short) Section 5, we describe the related open problems.

## 2 Definitions and Results

### 2.1 Description of the Problem in Precise Mathematical Terms

**Definition 1.**

- By an *interval measurement result*, we mean a sequence of intervals  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{x}_n$  such that  $0 \in \mathbf{x}_0$  and  $0 \in \mathbf{x}_n$ .
- A sequence  $x_0, \dots, x_n$  is called a *possible signal* if  $x_0 = x_n = 0$  and  $x_i \in \mathbf{x}_i$  for all  $i$ .
- By a *non-smoothness*  $S(\vec{x})$  of a sequence  $\vec{x} = (x_0, x_1, \dots, x_n)$ , we mean a value  $S(\vec{x}) = (x_1 - x_0)^2 + \dots + (x_{k+1} - x_k)^2 + \dots + (x_n - x_{n-1})^2$ .
- By a *smoothing* of a given interval measurement result, we mean a possible signal  $\vec{x}$  with the smallest possible non-smoothness  $S(\vec{x})$ .
- By an *interval smoothing problem*, we mean the following problem: given an interval measurement result, to find its smoothing.

*Comment.* In other words, the *interval smoothing problem* means the following:

**Given:** A sequence of intervals  $\mathbf{x}_i = [x_i^-, x_i^+]$ ,  $0 \leq i \leq n$ , for which  $0 \in \mathbf{x}_0$  and  $0 \in \mathbf{x}_n$ .

**To compute:** The values  $x_0, \dots, x_n$  for which  $x_0 = x_n = 0$ ,  $x_i \in \mathbf{x}_i$ , and  $\sum (x_{i+1} - x_i)^2 \rightarrow \min$ .

### 2.2 The Main Result: Quadratic-Time Smoothing Algorithm

**THEOREM 1.** *There exists an algorithm that solves the interval smoothing problem in time  $O(n^2)$ .*

Let us describe this algorithm.

**Algorithm.** *This is the recursive algorithm: the smoothing for a given  $n$  will be reduced to smoothings for smaller  $n$ .*

*Recursion base.* If  $n = 1$ , then we can directly compute the smoothing as  $x_0 = x_1 = 0$ .

*Recursion step.* If  $n > 1$ , then we do the following:

- 1) First, we analyze the values  $x_1^-, \dots, x_{n-1}^-$  one by one, find the largest of them, and also find the index  $M$  for which this largest value is attained. This can be done by keeping two variables:

- a real value **record** that will contain the largest positive value of  $x_i^-$ ; initially, it is equal to  $x_1^-$ ; for every other  $i$ , it is changed to  $\mathbf{record} := \max(\mathbf{record}, x_i^-)$ ; and
- an integer variable  $M$  that is assigned the new value  $i$  whenever  $x_i^- > \mathbf{record}$ .

If  $x_M^- > 0$ , then we take  $k := M$ ,  $x_k := x_k^-$ , and go to Step 4.

- 2) If  $x_M^- \leq 0$  (i.e., if all values of  $x_i^-$  are non-positive), then we (similarly to Step 1) analyze the values  $x_i^+$  one by one; we compute the smallest of such values, and find the index  $m$  for which this smallest value is attained.
- 3) If  $x_m^+ \geq 0$  (i.e., if all values  $x_i^-$  are non-positive and all values of  $x_i^+$  are non-negative), then we produce  $x_i = 0$  for all  $i$  as the smoothing and stop. If  $x_m^+ < 0$ , then we take  $k := m$  and  $x_k := x_k^+$ .
- 4) On this step, we do the following:
  - Solve the smoothing problem (by using the same algorithm) for  $k$  instead of  $n$ , and for the intervals  $\mathbf{y}_i = \mathbf{x}_i - x_k \cdot (i/k)$ ,  $0 \leq i \leq k$ . As a result, we will get the values  $y_1, \dots, y_{k-1}$ . We compute  $x_i = y_i + x_k \cdot (i/k)$ .
  - Similarly, we apply the same algorithm to the sequence  $\mathbf{y}_i = \mathbf{x}_i - x_k \cdot (n-i)/(n-k)$ ,  $k \leq i \leq n$ . As a result, we will get the values  $y_{k+1}, \dots, y_{n-1}$ , from which we compute  $x_i = y_i + x_k \cdot (n-i)/(n-k)$ .

*Comments.*

- The following modification of this algorithm will also work: if  $x_M^- > 0$ ,  $x_m^+ < 0$ , and  $m \neq M$ , then we can assign  $x_M = x_M^-$  and  $x_m = x_m^+$ , and then apply the same smoothing algorithm to *three* sub-intervals obtained by taking out  $m$  and  $M$ . This modification may save some time.
- If we have several processors working in parallel, then we can further speed up the computations:

### 2.3 A Parallel Smoothing Algorithm

If we have  $n$  processors  $p_1, \dots, p_n$  working in parallel, then we can speed up the computations in three ways:

- First, when we look for the smallest or the largest of the values  $x_i^\pm$ , then, instead of analyzing the values one by one, we can use a known parallel algorithm that finds maxima and minima of  $N$  numbers in time  $O(\log(N))$  (see, e.g., [6]):

- On the first step, we divide  $N$  values into  $N/2$  pairs, and compute the maximum of each pair; thus, we get  $N/2$  results.
  - On the next step, we divide these  $N/2$  results into  $N/4$  pairs, and for each pair, find the largest of the corresponding maxima; this will give us  $N/4$  results, each of them is the maximum of four consequent values.
  - The same “bisection” is to be repeated again and again, so that we are left with  $N/8, N/16, \dots, N/2^s, \dots$  values. After  $s \approx \log(N)$  steps, we get a single value that is equal to the desired maximum.
- Second, when we reduce the smoothing problem for  $x_i, 1 \leq i \leq n-1$ , to solving two different smoothing problems:
    - for  $x_i$  for  $i = 1, \dots, k-1$ , and
    - for  $x_i$  for  $i = k+1, \dots, n$ ,

we can solve these two subproblems *in parallel*, by allocating the processors  $p_1, \dots, p_{k-1}$  to solve the first subproblem, and the processors  $p_{k+1}, \dots, p_n$  to solve the second subproblem.

If one or both of the resulting subproblems are subdivided into two sub-subproblems, we can parallelize these sub-subproblems, etc.

- Finally, we can assign the task of computing each value  $\mathbf{y}_i = \mathbf{x}_i - x_k \cdot (\dots)$  and  $x_i = y_i + (\dots)$  to the corresponding processor  $p_i$ ; this, all these computations will be done in constant time.

As a result of this speed-up, we get the following:

**THEOREM 2.** *There exists a parallel algorithm that solves the interval smoothing problem in time  $O(n \log(n))$  on  $n$  processors.*

## 3 Example

### 3.1 Input

Let us illustrate our algorithm on the following “wavelike” interval function:  $\mathbf{x}_0 = [-1, 1]$ ,  $\mathbf{x}_1 = [-3, -1]$ ,  $\mathbf{x}_2 = [1, 3]$ ,  $\mathbf{x}_3 = [0, 2]$ ,  $\mathbf{x}_4 = [1, 3]$ ,  $\mathbf{x}_5 = [-3, -1]$ , and  $\mathbf{x}_6 = [-1, 1]$ .

### 3.2 Main Algorithm

First, we assign  $x_0 = x_6 = 0$ . Since  $n = 6 > 1$ , we do the following:

First, we compute  $M$  for which  $x_M^- = \max x_i^-$ . For these measurement results,  $M = 2$ , and  $x_2^- = 1 > 0$ . Therefore, according to our algorithm, we assign

$k = 2$ ,  $x_2 = 1$ , and go to Step 4. On Step 4, we assign  $x_2 = 1$ , and consider the following two sub-problems:

- (1) In the first one, we take  $k = 2$ , and take  $\mathbf{y}_1 = \mathbf{x}_1 - (1/2) \cdot x_2 = \mathbf{x}_1 - 0.5 = [-3.5, -1.5]$ . For this new problem,  $y_M^- = \max y_i^- = -3.5 < 0$ , so, we compute  $y_m^+ = \min y_i^+$ . We get  $y_m^+ = -1.5$  and  $m = 1$ . This value  $y_m^+$  is  $< 0$ , so, we take  $y_1 = y_1^+ = -1.5$ .

Hence, we take  $x_1 = y_1 + (1/2) \cdot x_2 = (-1.5) + 0.5 = -1$ .

- (2) In the second sub-problem, we have  $x_k/(n - k) = 1/4 = 0.25$ ; therefore,  $\mathbf{y}_3 = [-0.75, 1.25]$ ,  $\mathbf{y}_4 = [0.5, 2.5]$ , and  $\mathbf{y}_5 = [-3.25, -1.25]$ . Here,  $y_M^- = 0.5 > 0$  for  $M = 4$ , so, we can take  $y_4 = 0.5$ . Hence, in the original problem, we take  $x_4 = y_4 + 0.5 = 1$ . This second sub-problem is now subdivided into two sub-subproblems:

- (2.1) In the first of them, we have  $\mathbf{z}_3 = \mathbf{y}_3 - (1/2) \cdot 0.5 = [-1, 1]$ ; here,  $z_M^- < 0 < z_m^+$ ; therefore, according to Step 3 of our algorithm, as a smoothing, we take  $z_3 = 0$ .

Hence,  $y_3 = 0 + 0.25 = 0.25$ , and  $x_3 = 0.25 + 0.75 = 1$ .

- (2.2) For the second sub-subproblem, we have  $\mathbf{z}_5 = \mathbf{y}_5 - 0.25 = [-2.5, -1.5]$ . Here,  $z_M^- < 0$ , but  $z_m^+ < 0$ ; therefore, we take  $z_5 = z_m^+ = -1.5$ .

Hence,  $y_5 = z_5 + 0.25 = -1.25$ , and  $x_5 = y_5 + 0.25 = -1$ .

### 3.3 The Answer

The result of applying the smoothing algorithm to the given signal is  $x_0 = 0$ ,  $x_1 = -1$ ,  $x_2 = x_3 = x_4 = 1$ ,  $x_5 = -1$ , and  $x_6 = 0$ .

### 3.4 Parallel Algorithm

First, we assign  $x_0 = x_6 = 0$ . Since  $n > 1$ , we do the following:

First, we compute  $M$  for which  $x_M^- = \max x_i^-$ . This is done as follows:

- We compare  $x_1^-$  with  $x_2^-$  on the processor  $p_1$ , and simultaneously, we compare the values  $x_3^-$  with  $x_4^-$  on the processor  $p_3$ . The maxima of these pairs are correspondingly  $x_2^-$  and  $x_4^-$ .
- Next, we compare these maxima  $x_2^-$  and  $x_4^-$ . These values are equal, so, we can take each of them as the largest. Let us take the first of them, i.e.,  $x_2^-$ .
- Finally, we compare  $x_2^-$  with the only remaining value  $x_5^-$ . The largest is  $x_2^-$ , so  $M = 2$ .

For these measurement results,  $M = 2$ , and  $x_M^- = x_2^- = 1 > 0$ . Therefore, according to our algorithm, we assign  $k = 2$ ,  $x_2 = 1$ , and go to Step 4.

On Step 4, we subdivide the original problem into the following two sub-problems that will be performed in parallel:

- (1) In the first sub-problem, we take  $k = 2$ , and take  $\mathbf{y}_1 = \mathbf{x}_1 - (1/2) \cdot x_2 = \mathbf{x}_1 - 0.5 = [-3.5, -1.5]$ . For this new problem,  $y_M^- = \max y_i^- = -3.5 < 0$ , so, we compute  $y_m^+ = \min y_i^+$ . We get  $y_m^+ = -1.5$  and  $m = 1$ . This value  $y_m^+$  is  $< 0$ , so, we take  $y_1 = y_1^+ = -1.5$ . Hence, we take  $x_1 = y_1 + (1/2) \cdot x_2 = (-1.5) + 0.5 = -1$ .
- (2) In the second subproblem, we have  $x_k/(n - k) = 1/4 = 0.25$ ; therefore,  $\mathbf{y}_3 = [-0.75, 1.25]$ ,  $\mathbf{y}_4 = [0.5, 2.5]$ , and  $\mathbf{y}_5 = [-3.25, -1.25]$ . The computations of these intervals are done in parallel by processors  $p_3$ ,  $p_4$ , and  $p_5$  respectively. Here,  $y_M^- = 0.5 > 0$  for  $M = 4$ , so, we can take  $y_4 = 0.5$ . Hence, in the original problem, we take  $x_4 = y_4 + 0.5 = 1$ . This second subproblem is now subdivided into two sub-subproblems:

(2.1) In the first of them, we have  $\mathbf{z}_3 = \mathbf{y}_3 - (1/2) \cdot 0.5 = [-1, 1]$ ; here,  $z_M^- < 0 < z_m^+$ ; therefore, for the smoothing,  $z_3 = 0$ .

Hence,  $y_3 = 0 + 0.25 = 0.25$ , and  $x_3 = 0.25 + 0.75 = 1$ .

(2.2) For the second sub-subproblem, we have  $\mathbf{z}_5 = \mathbf{y}_5 - 0.25 = [-2.5, -1.5]$ . Here,  $z_M^- < 0$ , but  $z_m^+ < 0$ ; therefore, we take  $z_5 = z_m^+ = -1.5$ .

Hence,  $y_5 = z_5 + 0.25 = -1.25$ , and  $x_5 = y_5 + 0.25 = -1$ .

As a result, we get the same smoothing.

## 4 Proofs

### 4.1 Proof That a Smoothing Always Exists

1°. Let us first show that a smoothing always exists.

Indeed, in solving a smoothing problem, we find the minimum of a continuous function  $S(\vec{x})$  on a compact  $\{0\} \times \mathbf{x}_1 \times \dots \times \mathbf{x}_{n-1} \times \{0\}$ ; therefore, this minimum is always attained, i.e., the smoothing always exists.

### 4.2 Proof of the Algorithm's Correctness

2°. Let us show that if all the values  $x_i^-$  are non-positive, and all the values  $x_i^+$  are non-negative, then  $x_i = 0$  is the smoothing.

Indeed, the non-smoothness is always non-negative, and the only way for it to be equal to 0 is when  $x_{i+1} - x_i = 0$  for all  $i$ , i.e., when  $x_0 = x_1 = \dots = x_n = 0$ .

In this case,  $x_i = 0$  is a possible vector, so it has the smallest possible value of  $S$  and is, therefore, a smoothing.

3°. Let us show that if the value  $x_k^- = \max x_i^-$  is positive, then for the smoothing  $x_i$ , we have  $x_k = x_k^-$ .

We will prove this statement by reduction to a contradiction. Indeed, suppose that for a smoothing  $x_i$ , we have  $x_k \neq x_k^-$ . Since  $x_i$  is a smoothing and therefore, a possible signal, we have  $x_k \in [x_k^-, x_k^+]$ ; since  $x_k \neq x_k^-$ , we can conclude that  $x_k > x_k^-$ . Let us now consider a new signal  $y_i = \min(x_i, x_k^-)$ . We will show that this is also a possible signal, and that this new possible signal is smoother than  $\vec{x}$ :  $S(\vec{y}) < S(\vec{x})$ . This conclusion will contradict to the assumption that  $\vec{x}$  is a smoothing, i.e., the smoothest possible signal.

3.1°. Let us prove that  $\vec{y}$  is a possible signal.

We assumed that  $x_i$  is a possible vector, i.e., that  $x_0 = x_n = 0$ , and  $x_i \in \mathbf{x}_i$ . Let us show that the same properties are true for  $y_i$ .

For  $i = 0$ , we have  $x_i = 0$ , and hence,  $y_0 = \min(0, x_k^-) = 0$ . Similarly,  $y_n = 0$ .

If  $x_i \leq x_k^-$ , then  $y_i = \min(x_i, x_k^-) = x_i$ , hence,  $y_i = x_i \in \mathbf{x}_i$ .

If  $x_i > x_k^-$ , then we have  $y_i = x_k^-$ . Since  $x_k^- = \max_j x_j^-$ , we conclude that  $y_i = x_k^- \geq x_i^-$ . On the other hand,  $y_i < x_i \leq x_i^+$ . Hence,  $y_i \in [x_i^-, x_i^+] = \mathbf{x}_i$ .

3.2°. Let us show that  $S(\vec{y}) \leq S(\vec{x})$ .

For each three real numbers  $a$ ,  $b$ , and  $c$ , we have  $(\min(a, c) - \min(b, c))^2 \leq (b - c)^2$  (this inequality can be easily proven if we consider all possible orderings of three numbers  $a$ ,  $b$ , and  $c$ ). As a result, for  $a = x_i$ ,  $b = x_{i+1}$ , and  $c = x_k^-$ , we conclude that  $(y_i - y_{i+1})^2 \leq (x_i - x_{i+1})^2$  for all  $i$ . Adding these inequalities for all  $i$ , we conclude that  $S(\vec{y}) \leq S(\vec{x})$ .

3.3°. To complete our reduction to a contradiction, we must show that  $S(\vec{y}) < S(\vec{x})$ .

Indeed, for  $i = 0$ , we have  $x_0 < x_k^-$ ; for  $i = k$ , we have  $x_k > x_k^-$ . Therefore, there must exist the first value  $i$  for which  $x_i \leq x_k^-$  and  $x_{i+1} > x_k^-$ . For this  $i$ , we have  $x_i < x_{i+1}$ . According to our definition of  $i$ , we have  $y_i = x_i$  and  $y_{i+1} = x_k^- \geq x_i$ . Therefore, subtracting  $x_i$  from both sides of the inequality  $x_k^- < x_{i+1}$ , we conclude that  $0 \leq x_k^- - x_i < x_{i+1} - x_i$ , or, that  $0 \leq y_{i+1} - y_i < x_{i+1} - x_i$ . Hence,  $(y_{i+1} - y_i)^2 < (x_{i+1} - x_i)^2$ . Since for every other  $j$ , we have  $(y_{j+1} - y_j)^2 \leq (x_{j+1} - x_j)^2$ , we conclude that  $S(\vec{y}) < S(\vec{x})$ .

This conclusion, as we have already mentioned contradicts to our assumption that  $\vec{x}$  is a smoothing. This contradiction proves that for a smoothing,  $x_k = x_k^-$ .

4°. We have found the  $k$ -th component of the smoothing. To find all other components, we must minimize the expression that we get by substituting  $x_k$  into  $S(\vec{x})$ . This expression can be represented as the sum of the two parts:  $S(\vec{x}) = S_-(\vec{x}) + S_+(\vec{x})$ , where  $S_-(\vec{x}) = (x_1 - x_0)^2 + \dots + (x_k - x_{k-1})^2$  and  $S_+(\vec{x}) = (x_{k+1} - x_k)^2 + \dots + (x_n - x_{n-1})^2$ . These two components of the objective

function  $S$  depend on disjoint sets of variables:  $S_-$  depends on  $x_1, \dots, x_{k-1}$ , and  $S_+$  depends on  $x_{k+1}, \dots, x_{n-1}$ . Therefore, to minimize non-smoothness  $S$ , we must separately minimize  $S_-$  and  $S_+$ .

5°. The problem  $S_-(x_0, x_1, \dots, x_{k-1}, x_k) \rightarrow \min$  is similar to the smoothing problem, with the only difference that for the smoothing problem, the last value  $x_n$  is known to be 0, while for this problem, the last value  $x_k$  is also known, but its known value is positive.

To reduce this problem to the smoothing one, we will use the new variables described in the algorithm:  $y_i = x_i - x_k \cdot (i/k)$ . For these new variables, we have  $y_i \in \mathbf{y}_i$  and  $y_0 = y_k = 0$ . Let us describe  $S_-$  in terms of  $y_i$ .

In terms of these variables,  $x_i = y_i + x_k \cdot (i/k)$ , and therefore,  $x_{i+1} - x_i = (y_{i+1} - y_i) + x_k/k$ . Hence,

$$(x_{i+1} - x_i)^2 = (y_{i+1} - y_i)^2 + (x_k/k)^2 + 2(x_k/k)(y_{i+1} - y_i),$$

and so, we can represent  $S_-$  as the sum of the three components:

$$S_- = \sum (x_{i+1} - x_i)^2 = \sum (y_{i+1} - y_i)^2 + \sum (x_k/k)^2 + \sum 2(x_k/k)(y_{i+1} - y_i).$$

The last component of  $S_-$  is equal to  $2(x_k/k) \sum (y_{i+1} - y_i) = 2(x_k/k)(y_k - y_0) = 0$ . The second component of  $S_-$  is a constant  $(k \cdot (x_k/k)^2 = (x_k)^2/k)$  that does not depend on  $y_i$  at all. Therefore,  $S_- \rightarrow \min$  iff  $\sum (y_{i+1} - y_i)^2 \rightarrow \min$ .

This equivalence, together with the similarly proved equivalences for  $S_+$  and for the case when we choose  $k$  based on  $x_i^+$ , justifies the recursion described in the algorithm.

So, the above-described algorithm indeed computes the smoothing of a given signal. Q.E.D.

### 4.3 Proof that the Algorithm is Quadratic-Time

6°. To complete the proof of Theorem 1, we must show that our algorithm requires quadratic time.

To estimate the total number of computational steps in this algorithm, let us combine all the operations described in the algorithm before and after the recursive calls into a *first level*. The procedure of the first level calls the same algorithm twice. All the computational steps in these calls will be counted as the *second level*. These algorithms may also call the same algorithm recursively; thus, we get the *third level*, etc.

In the first level, for each  $i$  from 1 to  $n - 1$ , we apply at most eight computational steps:

- one, when we compare the value `record` with  $x_i^-$  when finding the largest of  $x_i^-$ ;
- one (maybe) when we compare `record` with  $x_i^+$  to find the smallest of  $x_i^+$ ;

- at most six to compute the bounds of the interval  $\mathbf{y}_i$ :
  - four, when we compute two bounds of an interval

$$\mathbf{y}_i = \mathbf{x}_i - x_k \cdot (i/k);$$

first, we compute  $x_k \cdot (i/k)$  (one division and one multiplication), and then subtract the result from both bounds (two subtractions);

- six, when we compute two bounds of an interval

$$\mathbf{y}_i = \mathbf{x}_i - x_k \cdot (n - i)/(n - k);$$

first, we compute  $x_k \cdot (n - i)/(n - k)$  (two subtractions, one division and one multiplication), and then subtract the result from both bounds (two subtractions);

- one, when we (after the recursive call) reconstruct  $x_i$  from  $y_i$ : we just add to  $y_i$  the previously computed value  $x_k \cdot (i/k)$  or  $x_k \cdot (n - i)/(n - k)$ .

Totally, we need  $\leq 8(n - 1)$  computational steps for this level. The second level consists of similar computations, but applied to two different problems; so, on this second level, we need  $\leq 8(k - 1)$  computational steps to solve the first problem (with  $i$  from 1 to  $k$ ), and  $\leq 8(n - k - 1)$  computational steps to solve the second problem (with  $i$  from  $k$  to  $n$ ). As a result, we totally need  $\leq 8(n - 2)$  computational steps.

This reduction in the number of computational steps is caused by the fact that we have found one value ( $x_k$ ), and we thus have one less value to consider. On each following level, we have at least one fewer value to consider. Initially, we had  $n - 1$  unknown values, and on each level, the number of unknown values decreases by 1. Therefore, we can have at most  $n - 1$  levels. On the first level, we use  $\leq 8(n - 1)$  steps; on the second level, we use  $\leq 8(n - 2)$ , etc. Totally, we use  $\leq 8[(n - 1) + (n - 2) + \dots + 1] = 8(n - 1)n/2 \leq 4n^2$  computational steps. Theorem 1 is proven. Q.E.D.

#### 4.4 Proof that the Parallel Algorithm Takes $O(n \log(n))$ Time

7°. Let us prove that the parallel algorithm described above takes  $O(n \log(n))$  time.

Indeed, as we have mentioned above, on each level, the main time-consuming operation is computing the maximum of  $N$  elements; all the other operations take constant time (if implemented in parallel). Therefore, on each level, the total computation time is  $O(\log(N))$ , i.e.,  $\leq C \log(N)$  for some constant  $C > 0$ .

- On the first level, we find the maximum of  $n - 1$  elements, so, we need time  $\leq C \log(n - 1)$ .

- On the second level, as we have shown in 6<sup>o</sup>, we process  $\leq n - 2$  elements in each subproblem, and therefore, the required time is  $\leq C \log(n - 2)$ ,
- etc.

Totally, we need time

$$\leq C(n - 1) + C(n - 2) + \dots + C \cdot 1 =$$

$$C[\log(n - 1) + \log(n - 2) + \dots + \log(1)] = C \log[(n - 1)!].$$

It is known that  $\log(n!) = O(n \log(n))$  (see, e.g., [1]), therefore, our parallel algorithm takes time  $O(n \log(n))$ . Theorem 2 is proven. Q.E.D.

## 5 Open Problems

- Is it possible to design a *linear-time* algorithm for smoothing an interval function? a *logarithmic-time parallel* algorithm?
- The measure of non-smoothness (1) described above corresponds to the non-smoothness functional  $S_c(x) = \int (\dot{x}(t))^2 dt$  that uses only the first derivative of the function. In some problems, it is more natural to use other smoothing functionals, that use *second* (and/or higher) order *derivatives* [9, 3, 2, 4, 5, 8], e.g., functionals of the type

$$S_c(x) = a_1 \int (\dot{x}(t))^2 dt + a_2 \int (\ddot{x}(t))^2 dt$$

that correspond to

$$S(\vec{x}) = A_1 \cdot \sum_{i=0}^{n-1} (x_{i+1} - x_i)^2 + A_1 \cdot \sum_{i=1}^{n-1} (x_{i-1} - 2x_i + x_{i+1})^2.$$

It is desirable to construct similar smoothing algorithms for such more general smoothing functionals, especially the functionals that use the second derivative of the signal.

**Acknowledgments.** This work was supported by NSF Grants No. CDA-9015006 and EEC-9322370, and by NASA Grant No. NAG 9-757. The authors are thankful to Slava Nesterov and to the anonymous referees for valuable suggestions.

## References

- [1] Th. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y., 1990.
- [2] V. B. Glasko, *Inverse problems of mathematical physics*, American Institute of Physics, N. Y., 1984.
- [3] *Inverse problems*, SIAM-AMS Proceedings, Vol. 14. American Mathematical Society, Providence, RI, 1983.
- [4] *Inverse problems*, Birkhauser Verlag, Basel, 1986.
- [5] *Inverse problems*, Lecture Notes in Mathematics, Vol. 1225, Springer-Verlag, Berlin-Heidelberg, 1986.
- [6] J. Jájá, *An introduction to parallel algorithms*, Addison-Wesley, Reading, MA, 1992.
- [7] V. Kreinovich, C. Quintana, R. Lea, O. Fuentes, A. Lokshin, S. Kumar, I. Boricheva, and L. Reznik, “What non-linearity to choose? Mathematical foundations of fuzzy control,” *Proceedings of the 1992 International Conference on Fuzzy Systems and Intelligent Control*, Louisville, KY, 1992, pp. 349–412.
- [8] M. M. Lavrentiev, V. G. Romanov, and S. P. Shishatskii, *Ill-posed problems of mathematical physics and analysis*, American Mathematical Society, Providence, RI, 1986.
- [9] A. N. Tikhonov, V. Y. Arsenin, *Solutions of ill-posed problems*, V. H. Winston & Sons, Washington, DC, 1977.

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

Karen Villaverde  
Systems Engineering, BNR  
P.O. Box 833871 M/S D0-207  
Richardson, TX 75083-3871, USA  
villa@bnr.ca