Scale-Based Monotonicity Analysis in Qualitative Modelling with Flat Segments

Martin Brooks[1] and **Yuhong Yan**[1] and **Daniel Lemire**[2]

[1]National Research Council
200 Montreal Road, M-50
Ottawa, ON K1A 0R6
martin.brooks@nrc.gc.ca
yuhong.yan@nrc.gc.ca

[2]University of Quebec
4750 avenue Henri-Julien
Montréal, QC H2T 3E4
lemire@ondelette.com

Abstract

Qualitative models are often more suitable than classical quantitative models in tasks such as Model-based Diagnosis (MBD), explaining system behavior, and designing novel devices from first principles. Monotonicity is an important feature to leverage when constructing qualitative models. Detecting monotonic pieces robustly and efficiently from sensor or simulation data remains an open problem. This paper presents scale-based monotonicity: the notion that monotonicity can be defined relative to a scale. Real-valued functions defined on a finite set of reals e.g. sensor data or simulation results, can be partitioned into quasi-monotonic segments, i.e. segments monotonic with respect to a scale, in linear time. A novel segmentation algorithm is introduced along with a scale-based definition of "flatness".

1 Introduction

Qualitative models are used in applications such as model-based diagnosis [Yan, 2003; Struss, 2002], explaining system behavior [Šuc, 2003; Forbus, 1984; Kuipers, 1986], and designing novel devices from first principles [Williams, 1992]. It is a challenge to build qualitative models for complex real world engineering systems. Current research efforts are on automatic generation of qualitative models from numerical data obtained by numerical simulation or sensors. Many proposed methods, such as in [Struss, 2002] and [Console *et al.*, 2003], work only when the functions are piecewise monotonic. Hence, partitioning data arrays into quasi-monotonic segments is an important problem [Yan *et al.*, 2004b; 2004a].

One could segment a data array in monotonic segments using a naïve algorithm: simply segment wherever there is an extremum. For example, it is easy to segment the array $\{0,\ 1,\ 2,\ 3,\ 2,\ 1,\ 0\}$ in two monotonic segments. However, monotonic segments must be significant for the application at hand and algorithms must be robust because the data will unavoidably be subject to noise or insignificant features. Hence, we might want to consider that the array $\{0,\ 1.2,\ 1.1,\ 3,\ 2,\ 1,\ 0\}$ has only two "significant" monotonic segments since the drop from 1.2 to 1.1 is not large enough to indicate a downward trend. Also algorithms must be fast ($O(n)$) and not require excessive amounts of memory ($O(n)$).

In this paper, we address the "scale-based monotonicity" problem: monotonicity can be defined relative to a scale. We define a metric for monotonic approximation. We solve the following problem: given a number of segments K, find K segments having the smallest monotonic approximation error.

In short, the main novel results of this paper are:

1. A novel optimal segmentation algorithm;
2. A novel definition of scale-based flatness.

2 Monotonicity in Qualitative Modelling

We just list a few examples to show that the monotonic features are important to leverage in qualitative modeling. In QSIM [Kuipers, 1986], the qualitative value $QV(f,t)$, is the pair <qmag, qdir>, where qmag is a landmark value l_i or an interval made up of landmark values (l_i, l_{i+1}), and qdir takes a value from the sign domain (+,0,-) according the $f'(t)$ is increasing, steady, or decreasing respectively. In Model-based Diagnosis (MBD), qualitative models are used to describe system behavior. Two kinds of qualitative models, based respectively on absolute or relative quantities, rely on the monotonicity of the function. The first one is *Finite Relation Qualitative Model* (FRQ) [Struss, 2002; Yan, 2003], where the qualitative relation is represented by tuples of real valued intervals. For a monotonic segment $[x_a, x_b]$ of f, the tuple is determined by the bounding rectangle $\square = [x_a, x_b] \times [\min(f(x_a), f(x_b)), \max(f(x_a), f(x_b))]$, that is the smallest rectangle such that $y = f(x) \bigwedge x_a \leq x \leq x_b \Rightarrow (x,y) \in \square$. The second one is *Qualitative Deviation Model* [Console *et al.*, 2003] where the qualitative relation is represented by the sign of deviation $[\Delta y]$ from a reference point x_{ref} defined as +,-, or 0, according to whether f is increasing, decreasing or flat. For example, if f is monotonic increasing, we have $[\Delta y] = \text{sign}(f(x) - f(x_{ref})) = \text{sign}(x - x_{ref}) = [\Delta x]$.

In this paper, we focus on abstracting qualitative models from scattered data obtained from numerical simulation or sensors. This work is motivated by at least two kinds of applications. First, when applying model-based diagnosis to real world engineering systems, we need to build symbolic qualitative models by a numerical simulation of the engineering models. Second, we need to explain system behaviors from

sensor data. Qualitative model abstraction in this paper is defined as transforming numerical values into qualitative values and functions into qualitative constraints [Šuc and Bratko, 2001]. **Monotonicity Analysis** is defined as partitioning a finite series of real values $x_1, x_2, \ldots, x_n$ over an interval D into "monotonic" segments.

Monotonicity analysis is a crucial step to abstract qualitative models from scattered data but it rises a problem. The difficulty is that when the scattered data contains noise, the monotonicity is not absolute in a neighborhood of any point. The monotonic segments need to be significant in the context of the problem. The small fluctuations caused by noise must be ignored. This requires that noise be removed by computationally efficient methods such that the number of remaining segments is dependent only on the characterization of the noise.

Linear splines is an obvious approach to solve this problem. We can use top-down, bottom-up and sliding window algorithms to approximate the data with a set of straight lines [Keogh *et al.*, 2001]. Then the same-sign slopes can be aggregated as monotonic segments. Various algorithms are derived from classic algorithms [Key *et al.*, 2000] [Hunter and McIntosh, 1999]. The downside of these methods is that there is no link between linear spline approximation error and the actual monotonicity of the data (consider $y = e^x$). Hence, using linear splines, it is difficult to specify either the desired number of monotonic segments or some "monotonicity error" threshold. In addition, linear fitting algorithms are relatively expensive.

Inductive learning is used in [Šuc and Bratko, 2001] to automatically construct qualitative models from quantitative examples. The induced qualitative model is a binary tree, called a qualitative tree, which contains internal nodes (called splits) and qualitatively constrained functions at the leaves. A qualitative constrained function takes the form $M^{s_1,\ldots,s_m} : \mathbb{R}^m \mapsto \mathbb{R}$, $s_i \in \{+,-\}$ and represents a function with m real-valued attributes strictly monotone increasing with respect to the i-th attribute if $s_i = +$, or strictly monotone decreasing if $s_i = -$. For example, $f = M^{+,-}(x,y)$ means f is increasing when x is increasing, and decreasing when y is increasing. A split is a partition of a variable. The unsupervised learning algorithm eq-QUIN determines the landmarks for the splits. The training data set is all possible pairs of data points. eq-QUIN checks the best split against all possible hypotheses. Its complexity is $O(n^2 2^m)$. QUIN is a more efficient algorithm that uses greedy search. Its complexity is $O(n^2 m^2)$. We do not address multidimensional data in this paper.

3 Monotonicity Error

In this section, we define a measure of monotonicity. Suppose we are given a set of n ordered samples noted $F : D = \{x_1, \ldots, x_n\} \subset \mathbb{R} \to \mathbb{R}$ with real values $F(x_1), \ldots, F(x_n)$ and $x_1 < x_2 < \ldots < x_n$. We define, $F|_{[a,b]}$ as the restriction of F over $D \cap [a,b]$. We seek the best monotonic (increasing or decreasing) function $f : \mathbb{R} \to \mathbb{R}$ approximating F. Let $\Omega_\uparrow$ (resp. $\Omega_\downarrow$) be the set of all monotonic increasing (resp. decreasing) functions. The **Optimal Monotonic Approximation Function Error (OMAFE)** of F is given by $\min_{f \in \Omega} \max_{x \in D} |f - F|$ where Ω is either $\Omega_\uparrow$ or $\Omega_\downarrow$. Sign + or - is associated to $\Omega_\uparrow$ or $\Omega_\downarrow$.

The segmentation of a set D is a sequence $S = X_1, \ldots, X_m$ of closed intervals (called "**segments**") in D with $[\min D, \max D] = \bigcup_i X_i$ such that $\max X_i = \min X_{i+1}$ and $X_i \cap X_j = \emptyset$ for $j \neq i, i+1, i-1$. Alternatively, we can define a segmentation from the set of points $X_i \cap X_{i+1} = \{y_i\}$. Given $F : \{x_1, \ldots, x_n\} \to \mathbb{R}$ and a segmentation $\{X_i\}$, the Optimal Piecewise Monotonic Approximation Function Error (OPMAFE) of the segmentation is given by $\max_i \text{OMAFE}(F|_{X_i})$ where the directions of the segments X_i are alternating and such that the direction of the first segment is chosen so as to minimize the OPMAFE.

Solving for a best monotonic function can be done as follows. If we seek the best monotonic increasing function, we first define $\overline{f}_\uparrow(x) = \max\{F(y) : y \leq x\}$ (the maximum of all previous values) and $\underline{f}_\uparrow(x) = \min\{F(y) : y \geq x\}$ (the minimum of all values to come). If we seek the best monotonic decreasing function, we define $\overline{f}_\downarrow(x) = \max\{F(y) : y \geq x\}$ (the maximum of all values to come) and $\underline{f}_\downarrow(x) = \min\{F(y) : y \leq x\}$ (the minimum of all previous values). These functions which can be computed in linear time are all we need to solve for the best approximation function as shown by the next theorem which is a well-known result [Brooks, 1994; Ubhaya, 1974].

Theorem 1. *Given $F : D = \{x_1, \ldots, x_n\} \to \mathbb{R}$, a best monotonic increasing approximation function to F is given by $f_\uparrow = \frac{\overline{f}_\uparrow + \underline{f}_\uparrow}{2}$ and a best monotonic decreasing approximation function is given by $f_\downarrow = \frac{\overline{f}_\downarrow + \underline{f}_\downarrow}{2}$. The corresponding error (OMAFE) is given by $\max_{x \in D} \frac{|\overline{f}_\uparrow(x) - \underline{f}_\uparrow(x)|}{2}$ (monotonic increasing) or $\max_{x \in D} \frac{|\overline{f}_\downarrow(x) - \underline{f}_\downarrow(x)|}{2}$ (monotonic decreasing).*

The implementation of the algorithm suggested by the theorem is straight-forward. Given a segmentation, we can compute the OPMAFE in $O(n)$ time using at most two passes. The functions $f_\uparrow$ and $f_\downarrow$ are sometimes called the standard optimal monotone functions as the solution is not unique in general.

4 Scale-Based Monotonicity

We present the notion of scale-based monotonicity. The intuition is that the fluctuations within a certain scale can be ignored.

Given an ordered set of real values $D = \{x_1, x_2, \ldots, x_n\}$, consider $F : D = \{x_1, x_2, \ldots, x_n\} \mapsto \mathbb{R}$. Given some tolerance value $\delta > 0$, we could say that the data points are *not going down* or are *upward monotone*, if consecutive measures do not go down by more than δ, that is, are such that $F(x_i) - F(x_{i+1}) < \delta$. However, this definition is not very useful because measures can repeatedly go down and thus the end value can be substantially lower than the start value. A more useful definition of *upward monotonicity* would be to require that we cannot find two values x_i and x_j ($x_i < x_j$) such that $F(x_j)$ is lower than $F(x_i)$ by δ (i.e. $F(x_i) - F(x_j) < \delta$). This definition is more useful because in the worst case, the

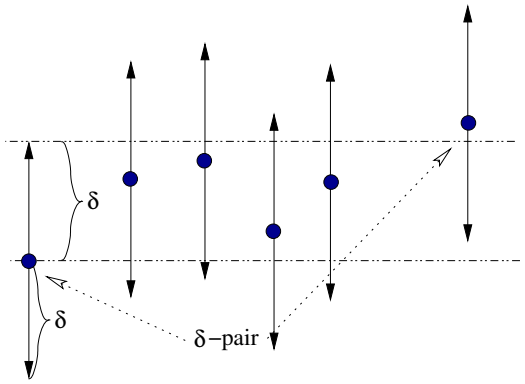

Figure 1: A δ-pair.

last measure will be only δ smaller than the first measure. However, we are still not guaranteed that the data does in fact increase at any point. Hence, we add the constraint that we can find at least two data points $x_k < x_l$ such that $F(x_l)$ is greater than $F(x_k)$ by at least δ ($F(x_l) - F(x_k) \geq \delta$).

To summarize, given some value $\delta > 0$, we say that a sequence of measures is *upward* δ-*monotone* if no two successive measures decrease by as much as δ, and at least one pair of successive measures increases by at least δ. Similarly, we say that a set of measures is *downward* δ-*monotone* if no two successive measures increase by as much as δ, and at least two measures decrease by at least δ.

This generalized definition of monotonicity was introduced in [Brooks, 1994] using δ-pairs (see Fig. 1):

Definition 1.

- *$x < y \in D$ is a δ-**pair** (or a pair of scale δ) for F if $|F(y) - F(x)| \geq \delta$ and for all $z \in D$, $x < z < y$ implies $|F(z) - F(x)| < \delta$ and $|F(y) - F(z)| < \delta$.*
- *A δ-pair's direction is **increasing** or **decreasing** according to whether $F(y) > F(x)$ or $F(y) < F(x)$.*

Notice that pairs of scale δ having opposite directions cannot overlap but they may share an endpoint. Pairs of scale δ of the same direction may overlap, but may not be nested for a certain δ.

We can define δ-monotonicity as follows:

Definition 2. *Let X be an interval, F is δ-**monotone** on X if all δ-pairs in X have the same direction; F is strictly δ-monotonic when there exists at least one such δ-pair. In this case:*

- *F is δ-**increasing** on X if X contains an increasing δ-pair.*
- *F is δ-**decreasing** on X if X contains a decreasing δ-pair.*

We say that a pair is significant at scale δ if it is of scale δ′ for $\delta' \geq \delta$.

In the next section, we discuss how to partition the data set into monotonic segments.

5 A Scale-Based Algorithm for Quasi-Monotonic Segmentation

Suppose that D is a finite set of reals having at least two elements. F is a real-valued function on D, i.e., $F : D \rightarrow R$.

We begin by defining a segmentation at scale δ or a δ-segmentation.

Definition 3. *Let $S = X_1, \ldots, X_n$ be a segmentation of D, and let $\delta > 0$, then S is a δ-**segmentation** of F when all the following conditions hold.*

- *Each X_i is δ-monotone.*
- *Each X_i for $i \neq 1, n$ is strictly δ-monotone.*
- *At least one X_i is strictly δ-monotone.*
- *Adjacent strictly δ-monotone segments have opposite directions.*
- *For each strictly δ-monotone X_i, and for all $x \in X_i$, $F(x)$ lies in the closed interval bounded by $F(\min X_i)$ and $F(\max X_i)$.*
- *When X_1 is not strictly δ-monotone, then for all $x \in X_1 - \max X_1$, $F(x)$ lies in the open interval bounded by $F(\min X_2)$ and $F(\max X_2)$; and when X_n is not strictly δ-monotone, then for all $x \in X_n - \min X_n$, $F(x)$ lies in the open interval bounded by $F(\min X_{n-1})$ and $F(\max X_{n-1})$.*

Each X_i is called a δ-segment of S. When strictly δ-monotone, X_i is a proper δ-segment; when not strictly δ-monotone, X_1 or X_n is an improper δ-segment. For strictly δ-monotone X_i, $\min X_i$ and $\max X_i$ are δ-extrema.

As the following theorems show, all δ-segmentations are "equivalent" and the monotonic approximation error (OPMAFE) is known precisely.

Theorem 2. *Let $\delta > 0$. Let S_1 and S_2 be δ-segmentations of F; then $|S_1| = |S_2|$. Furthermore, the first δ-segments of S_1 and S_2 are both either proper or improper, and similarly for the last δ-segments.*

Theorem 3. *Let $\delta > 0$. Let S be any δ-segmentation of F; then the monotonic approximation error (OPMAFE) $\leq \delta/2$.*

Now we want to compute a δ-segmentation given F. There are two approaches depending on the application one has in mind. The first one is to choose δ and then solve for the segments [Brooks, 1994], when the magnitude of noise is known. When one doesn't know how to choose δ, the second approach is to set the maximal number of segments K, especially when one knows the shape of the function. We focus on this second approach. We begin by labelling the extrema with a corresponding scale $\Delta(x)$.

Definition 4. *Let $x \in D$ be a local extremum of F. x's **delta-scale** $\Delta(x)$ is the largest $\delta > 0$ such that there exists a δ-segmentation having x as a δ-extremum.*

It is immediate from the definition that if $\delta > \Delta(x)$, then x can't be a δ-extremum, but also that if x is a δ-extremum, then $\Delta(x) \geq \delta$. This is stated in the following proposition.

Proposition 1. *Let $\delta > 0$, and let $S = X_1, \ldots, X_n$ be a δ-segmentation of F. Then $\Delta(x) \geq \delta$ for every δ-extremum, x, of S.*

We observe that there must be a smallest δ such that the cardinality of $X_\delta = \{x|\Delta(x) \geq \delta\}$ is at most K. However the set X_δ might contain repeated maxima or repeated minima and those might be removed before the set X_δ can define a δ-segmentation. This is stated in the next theorem.

Theorem 4. *Let $\delta > 0$ and consider the set of extrema $X_\delta = \{x|\Delta(x) \geq \delta\}$ of F as an ordered set. Each extremum in X_δ is either a maximum or a minimum, and a sequence of two minima or two maxima is possible. Consider a subset $X'_\delta \subset X_\delta$ such that*

- *it has no repeated maxima or minima;*
- *there is no superset X''_δ of X'_δ in X_δ without repeated extrema;*

then there exists a δ-segmentation S of F such that X'_δ is exactly the set of δ-extrema of S.

In order to compute $\Delta(x)$ for all extrema x, it is useful to introduce the following definitions.

Definition 5. *Opposite-sense extrema $x < z \in X \subset D$ are an **extremal pair** for X if for all $y \in X$, $x < y < z$ implies $F(y)$ lies in the closed interval defined by $F(x)$ and $F(z)$. The extremal pair has an extent $[F(x),\ F(z)]$ or $[F(z),\ F(x)]$ and increasing or decreasing direction according to $F(x) < F(z)$ or $F(x) > F(z)$. An extremal pair is **maximal** for X when no other extremal pair in X has larger extent. Similarly, an extremal pair is a maximal increasing (decreasing) when no increasing (decreasing) extremal pair has larger extent.*

Intuitively, a maximal extremal pair forms the largest decreasing (increasing) segment inside a larger increasing (decreasing) segment X.

Definition 6. *We recursively define **ordinary** and **special** intervals: D is an ordinary interval. When I is an ordinary interval, then an interval $J \subset I$ is special when J's endpoints constitute a maximal extremal pair in I; in this case J has direction inherited directly from its endpoints. Recursively, if J is special and interval $J' \subsetneq J$ has endpoints constituting a maximal extremal pair in J of direction opposite to that of J, then J' is special. Let $J_i, i = 1 \ldots n$ be all special intervals in an ordinary or special interval X. Choose any nonempty subset of the collection $\{J_i\}$, and let S be the segmentation of X defined by the endpoints of the special intervals in the subset. Then any segment $I \in S$ that does not contain any of the J_i is an ordinary interval.*

One can see that the special intervals are nested. The endpoints x, z of the special intervals have $\Delta(x) = \Delta(z) = |F(x) - F(z)|$. We can call them "twins" due to the same $\Delta(x)$ value. The ordinary intervals are not nested. The endpoints of the ordinary intervals have different $\Delta(x)$. We call each of them "singleton".

If there are several extrema having an equal value, the way to choose the endpoints to constitute a maximal extremal pair is not unique. Therefore, there are different sets of special intervals. They are equivalent. For simplicity, we do not consider the case of equal valued extrema in Algorithm 1 and Algorithm 2. Instead, we explain how to deal with it verbally after introducing the algorithms.

Algorithm 1 computes $\Delta(x)$ for every extremum $x \in D$. The input to Algorithm 1 is a list of extrema - i.e. the extrema data points in the data array.

Algorithm 1 This algorithm labels the extrema in linear time (as in Definition 4).

INPUT: data - a list of the successive extremal values of F, alternating between maximum and minimum. Each element is an "extremum record", having three fields: value, sense, and index. Index identifies the data point, value gives F's value at that data point, and sense indicates whether the extremum is a maximum or minimum.
OUTPUT: a list of "scale records". Each scale record has two fields: scale and extrema_list. Extrema_list comprises either one or two extremum records. The semantics of a scale record is that the indicated extrema have the indicated scale as delta-scale.
Notes about the algorithm: 1) Lists are accessed by element numbers; e.g. data(2) is the second element of data. 2) Lists are manipulated with functions Push and Pop. Push(thing, list) adds thing to list, resulting in thing being the first element of list. Pop(list) removes the first element of list, and returns this first element as the value of the function call. 3) ***MakeList***(item$^{1..*}$) pushes the items into a list generated, starting from the first item. 4) ***MakeScaleRecord*** creates a scale record.

```
LET extrema = empty_list
LET scales = empty_list
Push( Pop(data),extrema)
Push( Pop(data),extrema)
for next_extremum IN data do
   while Length(extrema) > 1 AND {{sense.next_extremum =
   "maximum" AND value.next_extremum > value.extrema(2)}
   OR {sense.next_extremum = "minimum" AND
   value.next_extremum < value.extrema(2)}} do
      Push(MakeScaleRecord( | value.extrema(1) -
      value.extrema(2) |, MakeList( extrema(2), extrema(1) )),
      scales)
      Pop(extrema)
      Pop(extrema)
   if Length( extrema ) = 0 then
      Push( Pop( extrema_list.scales(1) ), extrema )
   Push( next_extremum, extrema )
while Length(extrema) > 1 do
   Push( MakeScaleRecord (| value.extrema(1) -
   value.extrema(2) |, MakeList( extrema(1) )), scales)
   Pop(extrema)
Push ( extrema(1), extrema_list.scales(1) )
RETURN scales
```

The principle of Algorithm 1 is as follows: the *while* loop inside the *for* loop detects the special interval and labels the both endpoints with the difference of their values. But if the extrema list is empty after this labelling, which means that the top item in the list needs to be checked against the coming data, this top item is popped back to extrema in the *if* following the above *while*. Then the next extremum is pushed into the extrema list for the next *for* loops. The *while* outside the *for* loop determines the δ for the ordinary intervals. The last *push* states that the last extremum in the extrema list has the same scale as the one just popped into scale. Actually the endpoints of the biggest ordinary interval are always the last two pushed into scale record list and their scales are identical.

If one desires no more than K segments, it is a simple matter of picking δ as small as possible so that you have no more than $K+1$ significant extrema ($\Delta(x) \geq \delta$) together with the first and last extrema (indexes 0 and $n-1$). Algorithm 2 shows how once the labelling is complete, one can compute the segmentation in time $O(nK\log K)$ which we consider to be linear time when K is small compared to n. It also uses a fixed amount of memory ($O(K)$). The principle of Algorithm 2 is to select the $K+1$ data points to be the endpoints of K segments. Since the endpoints of D are by default, the remaining endpoints come from the largest δ-extrema. The *for* is to select the largest δ-extrema. As in Algorithm 1, we know that the labels are either "twins" or "singleton". Thus in the *for* loop, a maximum of $K+2$ data points are chosen. Then, after the first *if* outside *for*, the smallest δ-extrema are removed to reduce the total endpoints to at most $K+1$. The rest of the code checks whether the first and the last endpoints of D are included. If not, the smallest δ-extrema will be removed in favor of the first and the last endpoints. The number of segments can be less than K since several extrema can take the same δ value and can be removed together.

Algorithm 2 Given a labelling algorithm (see Algorithm 1), this algorithm returns an optimal segmentation using at most K segments. It is assumed that there are at least $K+1$ extrema to begin with.

INPUT: an array d containing the values indexed from 0 to $n-1$
INPUT: K a bound on the number of segments desired
OUTPUT: segmentation points as per Theorem 4 and Proposition 1
$L \leftarrow$ empty array (capacity $K+3$)
for e is index of an extremum in d having scale δ, e are visited in increasing order **do**
  insert (e,δ) in L so that L is sorted by scale in decreasing order (sort on δ) using binary search
  if length(L) = $K+3$ **then**
    pop last(L)
if length(L) > K+1 **then**
  remove all elements of L having the scale of last(L).
if indexes $\{0, n-1\} \not\subset L$ **then**
  if (index $0 \in L$ OR index $n-1 \in L$) AND length(L) = K+1 **then**
    remove all elements of L having the scale of last(L).
  if (index $0 \notin L$ AND index $n-1 \notin L$) AND length(L) $\geq K$ **then**
    remove all elements of L having the scale of last(L).
RETURN: the indexes in L adding 0 and/or $n-1$ when not already present

Algorithms 1 and 2 assume that no two extrema can have the same value. In case of equal valued extrema, we can modify these algorithms without increasing their complexity. In Algorithm 1, we can generalize ScaleRecord so that entries in the extrema_list field can also be lists of extrema of the same value. Then in the *while* loop, we can put the extrema of the same value into the corresponding list. For Algorithm 2, we just need to remember that whenever removing a middle point between two equal valued extrema, the two equal valued extrema have to be treated as one extrema to avoid having two adjacent minima or maxima.

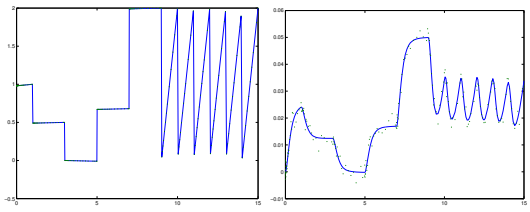

Figure 2: Input flow of tank A (left) and level of tank B (right) with added white noise.

6 Scale-Based Flatness

For applications, it is important to be able to find "flat" segments robustly in a data set. We propose the following definition:

Definition 7. *Given $\delta > 0$, consider a δ-monotonic segment in a δ-segmentation, an interval I in this segment is δ-flat if the standard optimal monotonic approximating function (as defined in Theorem 1) is constant over I.*

Because we can compute the standard optimal monotonic approximating function in $O(n)$ time, we can find flat segments in $O(n)$ time.

The next proposition addresses the flat segments in different δ-segmentations. The first point is derived from the fact that when the δ increases, the new segments are always the result of mergers of the previous segments. Hence, the segments at a small δ are always subsets of the segments at a larger δ. The second point says that the old flat segments will still be flat in the merged segment.

Proposition 2. *Let S be a δ-segmentation and let S' be a δ'-segmentation for $\delta' > \delta$, then 1) any δ-segment $X \in S$ is a subset of some δ'-segment $X' \in S'$, 2) for any δ-flat interval I in X, I is also δ'-flat in X'.*

7 Experimental Results

7.1 Cascade Tank Sample Data

As a source of synthetic data, we consider a system which consists of two cascade tanks A and B. Each tank has an input pipe (incoming water) and an output pipe (outgoing water). Tank A's output pipe is the input pipe of tank B. In the equations below, let A and B be the level of water for the two tanks respectively. The change of water level is proportional to the difference of the input flow and the output flow. Assume *in* is the input flow of tank A. $f(A)$ is the out flow of tank A. $g(B)$ is the output flow of tank B [Kuipers, 1994]. Thus, we have

$$\begin{aligned} A' &= in - f(A) \\ B' &= f(A) - g(B) \end{aligned}$$

In principle, $f(A)$ and $g(B)$ are increasing functions. We assume $f(A) = k_1 A$ and $g(B) = k_2 B$. If we variate the input flow of tank A, *in*, we can control the level of tank B. In this way, we generated some sample data and added noise to it. See Fig. 2 for the the input flow, *in*, of tank A at the left and the level of tank B at the right.

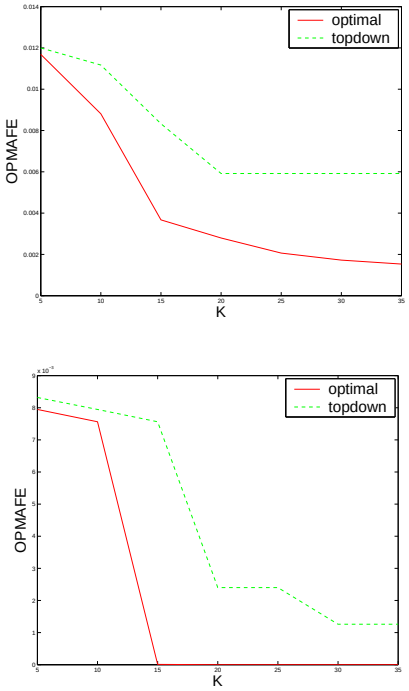

Figure 3: OPMAFE versus number of segments K for tank B with (above) and without white noise (below).

7.2 Optimal Algorithm vs. Top-down Algorithm

We choose the top-down linear spline approximation algorithm to compare the performance with the optimal algorithm developed in this paper. The top-down algorithm is a refined process choosing best split points that minimizes the linear regression error. The top-down algorithm we implemented has complexity of $O(nK^2)$ which is the best complexity on record. The optimal algorithm in this paper is $O(nK\log K)$. The optimal algorithm is faster for K sufficiently large. We also compare the monotonicity approximation error using OPMAFE. Figure 3 compares the OPMAFE for segmentations computed using either algorithm. The optimal algorithm we presented in this paper performs significantly better than the top-down algorithm. Note that OPMAFE is an absolute error measurement (which grows with the amplitude of the data).

8 Conclusion

Monotonicity is the most important feature to leverage when building the qualitative model from the scattered numerical data. This paper gives a solid mathematical foundation to the problem of defining monotonicity based on scale theory, where the values of the data points, not the distance between the data points, determine the monotonicity. It proves to be suitable in qualitative modeling. We present efficient algorithms to segment the data set into monotonic segments in linear time. We will continue to work on the multidimensional case.

References

[Brooks, 1994] Martin Brooks. Approximation complexity for piecewise monotone functions and real data. *Computers and Mathematics with Applications*, 27(8), 1994.

[Console *et al.*, 2003] L. Console, G. Correndo, and C. Picardi. Deriving qualitative deviations from matlab models. In *Proc. of 14th Int. Workshop on Principles of Diagnosis*, pages 87–92, 2003.

[Forbus, 1984] K. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[Hunter and McIntosh, 1999] Jim Hunter and Neil McIntosh. Knowledge-based event detection in complex time series data. In *Proc. of Artificial Intelligence in Medicine and Medical Decision Making (AIMDM99), LNAI 1620*, pages 271–280, 1999.

[Keogh *et al.*, 2001] Eamonn J. Keogh, Selina Chu, David Hart, and Michael J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[Key *et al.*, 2000] Herbert Key, Bernhard Rinner, and Benjamin Kuipers. Semi-quantitative system identification. *Artificial Intelligence*, 119, 2000.

[Kuipers, 1986] B.J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[Kuipers, 1994] Benjamin Kuipers. *Qualitative Reasoning*. the MIT press, 1994.

[Struss, 2002] P. Struss. Automated abstraction of numerical simulation models - theory and practical experience. In *Proc. of 16th Int. Workshop on Qualitative Reasoning*, pages 161–168, 2002.

[Ubhaya, 1974] V. A. Ubhaya. Isotone optimization I. *Journal of Approximation Theory*, 12:146–159, 1974.

[Šuc and Bratko, 2001] D. Šuc and I. Bratko. Induction of qualitative tree. In *Proc. of European Conference of Machine Learning in 2001 (LNCS 2167)*, pages 442–453. Springer, 2001.

[Šuc, 2003] D. Šuc. *Machine Reconstruction of Human Control Strategies*, volume 99 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, 2003.

[Williams, 1992] B. Williams. Interaction-based invention: designing devices from first principles. In *Recent Advances in Qualitative Physics*, pages 413–433. MIT Press, Cambridge, MA, 1992.

[Yan *et al.*, 2004a] Yuhong Yan, Daniel Lemire, and Martin Brooks. Monotone pieces analysis for qualitative modeling. In *Proceedings of ECAI MONET'04*, 2004.

[Yan *et al.*, 2004b] Yuhong Yan, Daniel Lemire, and Martin Brooks. Monotonicity analysis for constructing qualitative models. In *Proceedings of MBR'04*, 2004.

[Yan, 2003] Y. Yan. Qualitative model abstraction for diagnosis. In *Proc. of 17th Int. Workshop on Qualitative Reasoning*, pages 171–179, 2003.