Quasi-Monotonic Segmentation of State Variable Behavior for Reactive Control

Will Fitzgerald
QSS Group, Inc.
NASA Ames Research Center
Moffett Field, CA, 94035

Daniel Lemire
University of Quebec in Montreal
4750 Henri-Julien
Montréal, QC H2T 3E4

Martin Brooks
National Research Council of Canada
1200 Montreal Road
Ottawa, ON K1A 0R6

Abstract

Real-world agents must react to changing conditions as they execute planned tasks. Conditions are typically monitored through time series representing state variables. While some predicates on these times series only consider one measure at a time, other predicates, sometimes called episodic predicates, consider sets of measures. We consider a special class of episodic predicates based on segmentation of the the measures into quasi-monotonic intervals where each interval is either quasi-increasing, quasi-decreasing, or quasi-flat. While being scale-based, this approach is also computational efficient and results can be computed exactly without need for approximation algorithms. Our approach is compared to linear spline and regression analysis.

Reactive control and monitoring

It has been recognized for some time that real-world agents have to monitor conditions as they execute planned tasks. It is rarely the case that the only changes that can occur in the world result from actions taken by the agent. Actions taken by other agents, uncontrollable events occurring in the physical world, and the presence of noise and uncertainty are the norm.

Given this, an agent must be able to react to changes in the world. Moreover, the agent must be able to notice the changes in the first place. A typical description of this is a three-tiered architecture (Bonasso *et al.* 1997), with sensors and effectors at the "bottom" tier, and a high-level planner at the "top" tier. In between, the task execution system monitors for changing conditions and executes actions as appropriate based on the goals given it by the planner. Reactive task execution systems include EXEC (Pell *et al.* 1997), the RAP System (Firby 1989), PRS (Myers 1996), and Apex (Freed *et al.* 2003).

Of course, the conditions to which an agent might imaginably react can be arbitrarily complex, and, for an agent acting in the real world, a "paralysis of analysis" prevents effective action. Part of an effective architecture for agent design is describing the kinds of useful conditions which are likely to be relevant to the agent making use of the kinds of sensors typically available to the agent. Furthermore, it must be feasible–that is, computational efficient–to recognize the

relevant conditions, especially in concert with a variety of simultaneously executing tasks and monitors.

Sensors nominally create time-stamped data, with readings sometimes done at constant intervals, sometimes not. Each reading provides a time-stamped *measurement* of a *state variable*, that is, an *attribute* of some *object* (often the agent, or a component of the agent), which changes over time. For example, an autonomous aircraft will typically have sensors for its own altitude, longitude and latitude, among other measurements. Tasks such as "raise-gear" might have a precondition that the measured altitude be $\geq$ a certain amount.

Given the time series nature of such sensor data, it is natural to consider monitors being "live" for some *interval* of time. Within such an interval I, we have an ordered set of measurements for a given sensor, $\{x_t\}$, ordered by time t. If P is a boolean predicate on x, and n is the cardinality of $\{x_t\}$, then testing for a value in I upon which P is true is clearly at least $O(n)$. In a typical case, P is of constant complexity (such as the example of $\geq$ given above), and so, testing for value upon which P is true is $O(n)$.

Predicates on individual measurement values are not the only possible boolean predicates of interest. For example, an agent might want to act when the average value of a measurement within an interval reaches a certain point. The presence of a condition based on multiple measurements (of the same or different state variable) is sometimes called an *episode* (Mannila, Toivonen, & Verkamo 1997). Episodic predicates can, of course, be arbitrarily complex, and thus be difficult or impossible for an agent to consider. In this paper, we discuss one particular class of episodic predicates, namely those based on segmenting an ordered set of measurements according to changes in monotonicity. For example, consider Fig. 1, which shows a flight path of an airplane (simulated) as it takes off, circles an airport, and then crashes. A reactive controller may monitor for nominal conditions, such as (in the present case) making the first turn. A controller may also monitor for off-nominal conditions, such as failing to make a turn, or (in the present case) "in danger of crashing." In the nominal case, "making a turn" can be sensed by the appropriate increase in longitude and/or latitude. In the off-nominal example, "in danger of crashing" can be sensed by the an unexpected decrease in altitude.

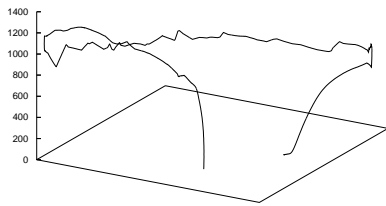If sensor data were typically comprised of long monotone

Figure 1: Flight path of airplane

Figure 2: Mass data at different scales

segments, then such a change in monotonicity would be trivial to detect. Sensors, of course, do not do so, for a variety of reasons. First, sensors are typically sampled at some rate, creating discontinuities. Second, exogenous events cause changes in sensor values which can be non-continuous; indeed, if this were not the case, there would be little need for sensors. Third, sensors inherently produce some amount of error, which can be minimized but never eliminated.

Consider the two plots in Fig. 2, which show real data from a life-support application, showing the measured mass of a distillant in a tank. The top plot exemplifies "noise" due to exogenous events. It is clear that the mass increases (as distillant flows into the tank), then decreases (as it is pumped out of the tank). There are slight "jags" on the increase and decrease (due to some exogenous event such as a cycling pump), but these can be ignored. There is another indication of noise at the beginning of the first increase, probably due to someone picking up the tank and replacing it.

The second plot exemplifies the other kinds of noise; this magnification of the first peak shows the discrete nature of sampling and the inherent error of the scale become evident.

Essentially, we would like to segment the time series of measurements of a state variable into regions of alternating sign, ignoring small violations of monotonicity. An algorithm for doing this has three desiderata: (1) it must provide such a segmentation in a way that is both intuitively and mathematically sound; (2) it must be computationally efficient enough to run in real time, and (3) ideally, it is an *on-line* algorithm that can be updated efficiently as an agent acts and senses in the world.

In this paper, we present an algorithm for segmentation of state variable measurements based on the *quasi-monotonic segmentation* ideas of (Brooks 1994), which provides a mathematically well-motivated notion of monotonic segmentation at a given scale. The algorithm is $O(n)$, where n in the number of measurements, and only requires constant space. It is also an on-line algorithm, whose update runs in constant time and space. We contrast this with algorithms typically used for segmentation, based on linear splining and regression techniques, which provide less intuitive results and are, in fact, $O(n^2)$.
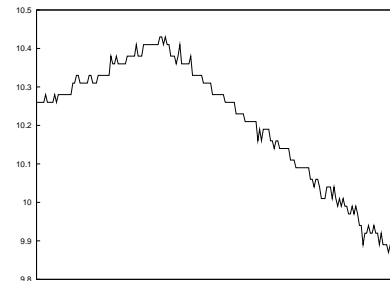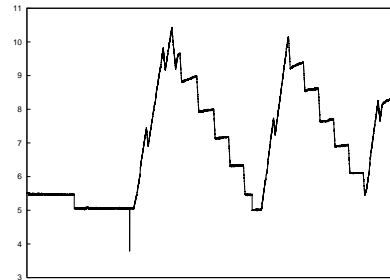
Quasi-monotonic segmentation

For a state variable x, say that an ordered set of measurements of x, $\{x_k\}$ is monotonic if either $x_k \geq x_{k+1}$ or $x_k \leq x_{k+1}$. As we have stated, this does not capture an interesting understanding of "monotonic," due to the sources of noise mentioned above. For practical applications, we want to tolerate some deviation from monotonicity. One possible way to describe allowing such a toleration is to say, given an ordered set of measurements $\{x_k\}$ and some tolerance value $\delta > 0$, 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 $x_i - x_{i+1} > \delta$. However, this definition is not very useful because measures can repeatedly go down and (eventually) the end value can be substantially lower than the start value. A more useful definition of *upward monotonicity* is to require that we cannot find two successive measures x_i and x_j ($j > i$) such that x_j is lower than x_i by δ ($x_i - x_j > \delta$). This definition is more useful because in the worse case, the last measure will be only δ smaller than the first measure. However, we are still not guaranteed that the data does in fact increase. Hence, we ask that we can find at least two successive measures x_k and x_l ($l > k$) such that x_l is greater than x_k by at least δ ($x_l - x_k \geq \delta$).

In order to formalize this concept, we will use the idea of δ-pair introduced in (Brooks 1994) (see Fig. 3). Let F be a function, and D be its domain. The tuple x, y ($x < y \in D$) 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
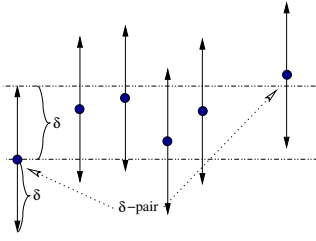
Figure 3: A δ-pair.

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 end point. Pairs of scale δ of the same direction may overlap, but may not be nested. We use the term "pair" to indicate a δ-pair having an unspecified δ.

We say that a sequence of measures $X = \{x_k\}_k$ is δ-monotonic if all δ-pairs have the same sign (all increasing or all decreasing). Given a δ-monotonic data set, we say that it is δ-flat if it contains no δ-pair and it is δ-increasing (resp. decreasing) if all δ-pairs are increasing (resp. decreasing).

Given a sequence of measures $X = \{x_k\}_{k=1,\ldots,N}$, a segmentation $X_1 \ldots X_n$ is given by a set of $n+1$ indices y_i such that $y_{i+1} > y_i$, $y_1 = 1$ and $y_{n+1} = N$ and where $X_i = \{x_k\}_{k=y_i,\ldots,y_{i+1}}$. A segmentation is an extremal segmentation at scale δ if

1. All X_i are δ-monotone;

2. No two adjacent segments have the same monotonicity (flat, increasing, decreasing);

3. For $i = 2, \ldots, n$, $x_{y_i} = \max X_i \cup X_{i-1}$ or $x_{y_i} = \min X_i \cup X_{i-1}$;

4. Unless X_1 is δ-flat, $x_1 = \max X_1$ or $x_1 = \min X_1$; unless X_n is δ-flat, $x_N = \max X_n$ or $x_N = \min X_n$.

The merger of a δ-increasing (resp. decreasing) segment with a δ-flat segment is δ-increasing (resp. decreasing). Hence, depending on whether or not we consider δ-flat segments, we can have various extremal segmentation with varying number of segments. However, if we exclude δ-flat segments, then the number of segments will be the same for all extremal segmentations at the same scale.

Algorithm for segmentation

Algorithm 1 presents an algorithm for quasi-monotone segmentation. A queue is initialized with the index of the first measurement. After determining the initial direction of the measurements, it scans through the measurements, keeping a back-pointer to the last extremum, which is initialized to the first position. When the difference between an element and the measurement at the back-pointer exceeds δ, and the new δ-pair switches signs, a new extremum is added to the queue, the sign is flipped and the back-pointer updated. After completing the scan, the index of the last measurement is added to the queue if it isn't already present.

Basic initialization (lines 4–6) take constant time. The *determineSign* function takes no more than $n - 1$ steps (in practice, it will take many fewer). The algorithm is dominated by the loop in lines 8–19, which is run n times. All of the comparisons and assignments within the loop take constant time; thus the loop runs in $O(n)$. Checking for the presence of the last index in the queue (line 20), and adding it to the queue if it isn't present, takes constant time. Thus, Algorithm 1 is $O(n)$.

All variables (except for the output queue) in Algorithm 1 take constant space.

Note that Algorithm 1 can easily be converted to an on-line algorithm by converting the **for** loop into a stream consumer. As new measurements arrive, they either result in a new extremum, in which case the new extremum is output.

Theorem 1 *Algorithm 1 generates an extremal segmentation at scale δ.*

Proof. Any segment having last index k that was added to Q while $D = 1$ cannot have decreasing δ-pairs. Indeed, suppose that there exist indexes l, j in the segment such that $l < j$ and $x_l - x_j \geq \delta$. When i (the variable of the main for loop) took value j, the maximum of the segment up to this point was x_k with $x_k \geq x_l \Rightarrow x_k - x_j \geq x_l - x_j$ and because j is included in the segment, $x_k - x_j < \delta$ hence $x_l - x_j\delta$.Similarly, we could also show that segments with $D = -1$ cannot have increasing δ-pairs.

Next, notice that the end of all segments, except possible the last two, is set before or at the beginning of a δ-pair with an opposite direction (decreasing if $D = 1$ or *vice versa*). This means that all segments except possibly the first one and the last one are either δ-increasing or δ-decreasing and they alternate. Observe that when either of the first or last segment is neither δ-increasing nor δ-decreasing, then it must be δ-flat. $\square$

As the next theorem shows, our definitions and Algorithm 1 allow for a multiscale analysis: we can relate analyses done at different scales. In short, as we reduce the tolerance, existing segments will be partitioned and new end-points will be added, but existing end-points will remain.

Theorem 2 *Using Algorithm 1, the segmentation points found with $\delta' < \delta$ include those found with δ.*

Proof. At scale δ and as per Algorithm 1, consider an increasing segment ending at index j' followed by a decreasing segment. We have that $x_{j'}$ is the maximum of both segments. A pair at scale δ, must contain a pair at scale δ' for all $\delta' < \delta$. Hence, a segment increasing (decreasing) at scale δ must contain a segment increasing (resp. decreasing) a scale δ'. Hence, there is at least one pair at scale δ' before (and after) j'. Consider the last pair at scale δ' before j', we see that it must be an increasing pair because $x_{j'}$ is a maximum: if the last pair is decreasing, then its starting point would have a value greater than $x_{j'}$ which is impossible. Similarly, the first pair at scale δ' after j' is decreasing. Because j' is the index of the first maximum with value $x_{j'}$ before the first pair, it will be selected by the algorithm at scale δ'. By symmetry, the same result is shown for a segmentation point following a decreasing segment.

Consider now the second segmentation point (at index j) preceded by a flat segment and followed by a decreasing segment at scale δ. We have that x_j is the maximum of both segments. The first pair at scale δ' following j must be decreasing and the last pair before j at scale δ', if it exists, must be increasing. Hence, j must be a segmentation point at scale δ'. The result is shown similarly when j is the index of a minimum or the second last segmentation point at scale δ. $\square$

Algorithm 1 Algorithm for quasi-monotone segmentation.

1: **INPUT:** sequence of values x_i with $i = 1, \ldots, n$
2: **INPUT:** scale δ
3: **OUTPUT:** an ordered sequence of indices which define the segmentation
4: $Q \leftarrow$ empty queue
5: place 1 in Q
6: $k \leftarrow 1$
7: $D \leftarrow \text{determineSign}(x)$
8: **if** $D = 0$ **then**
9:     add n to Q
10:     exit with Q
11: **for** $i \in \{1, \ldots, n-1\}$ **do**
12:     **if** ($D = 1$ and $x_k - x_i \geq \delta$) or ($D = -1$ and $x_i - x_k \geq \delta$) **then**
13:         add k to Q
14:         flip sign of D
15:     **else if** ($D = 1$ and $x_i > x_k$) or ($D = -1$ and $x_i < x_k$) **then**
16:         $k \leftarrow i$
17: **if** last(Q) isn't k and size(Q)>1 **then**
18:     add k to Q
19: **if** last(Q) isn't n **then**
20:     add n to Q
21: exit with Q

1: **FUNCTION** determineSign
2: **INPUT:** sequence of values x_i with $i = 1, \ldots, n$
3: **OUTPUT:** initial direction as -1,1 or 0
4: **for** i in $\{2, n\}$ **do**
5:     **if** $x_i \neq x_0$ **then**
6:         exit with sign of $x_i - x_0$
7: exit with 0

Examples

Figures 4, 5, and 6 show the results of running Algorithm 1 on the aircraft data and mass data described above. The three state variables of the aircraft data (latitude, longitude, and altitude) are plotted separately, and normalized to 1.0[1]. The longitude and latitude data have been split into three segments, with the middle segment beginning and ending where the aircraft began its two turns. The altitude data has been split into two segments: its steep then gradual rise, and its steep decline (and crash).

---

[1]In the simulation, the aircraft did not fly very far–no more than 20 kilometers in either direction–so no conversion from angular measurements have been made here.
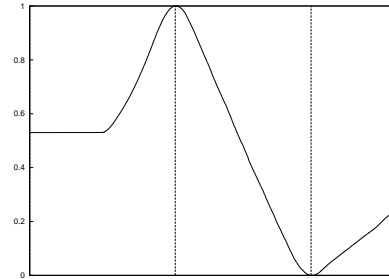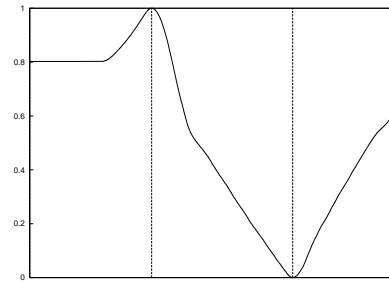

Figure 4: Segmented latitude and longitude data.

The mass of the tank plus distillant plotted in Fig. 6 ranges from about 5 to 11 (except for the anomalous drop to 4). With δ set to 1, the data are placed into six segments: one ending at the anomalous drop, the other five tracking the general rise and fall of the sensed mass.

Comparison to other methods

Segmenting data is a standard problem in time series research; a useful survey can be found in (Keogh *et al.* 1993), who identify three types of approaches: *sliding windows*, in which a segment is grown until an error bound is reached; *top-down*, in which a time-series is recursively partitioned, and *bottom-up*, in which larger and larger segments are merged, starting from smallest segments. Keogh *et al.* also
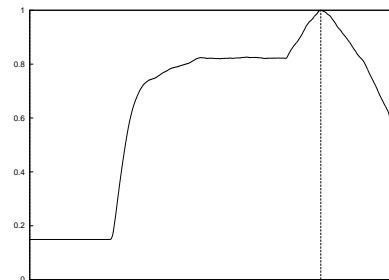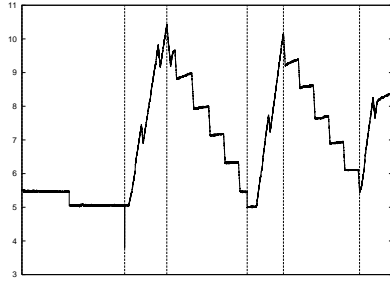

Figure 5: Segmented altitude data.
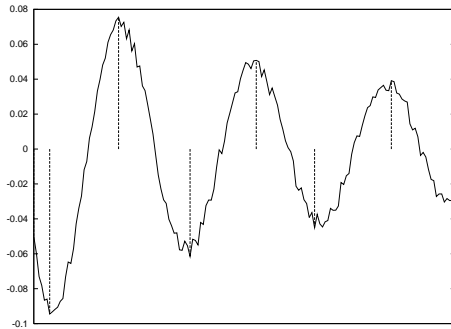
Figure 6: Segmented mass data.

Figure 7: Damped sign wave with noise

Figure 8: Execution time: Alg.1 versus Sliding Window. Note the execution time of Alg. 1 is close to the zero axis.

provide an algorithm, **SWAB**, which combines a sliding window and bottom-up approach (SWAB stands for "sliding window and bottom-up").

Keogh *et al.*'s complexity analysis reveals that the top-down algorithms have a complexity of $O(n^2K)$, where K is the number of segments created, and n is the number of points in the time series. The sliding window, bottom-up, and SWAB algorithms have a complexity of $O(n^2/K)$.

It is interesting to consider what happens as more or fewer segments are created. The maximum number of segments is $n-1$, resulting when each point is paired with its successor to create a segment, up to $n-1$. In this case, the complexity of top-down in $O(n^3)$, and the other algorithms are $O(n)$. The minimum number of segments is 1, resulting from including all of the points in the time series in the one segment. In this case, the complexity of top-down in $O(n^2)$; the other algorithms are also $O(n^2)$. Keogh *et al.* seem to assume that K will be closer to n than it is to 1, and so they claim that SWAB is $O(n)$. Given this assumption; this is correct; however, it is more reasonable to give the complexity analysis in its $O(n^2/K)$. form.

A strong advantage of Algorithm 1 is that its complexity is not sensitive to the number of segments created. As we showed above, Algorithm 1 is $O(n)$, and its complexity is unrelated to K.

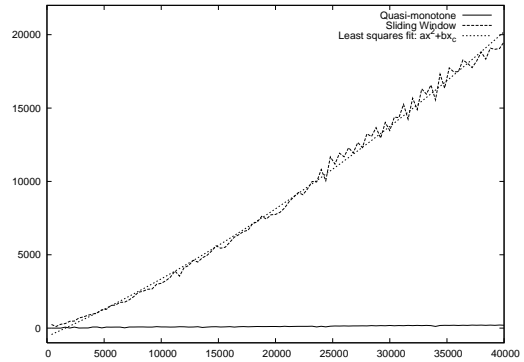It should be clear that monotonicity analysis is performing a related but different task from the algorithms considered by Keogh *et al.*. Consider the damped sin wave, $\sin(x)/x$, as shown (with added noise and with segmentation by Algorithm 1) in Fig. 7. A monotonicity analysis looks just for the locations where the function moves from decreasing to increasing, or vice versa. The linear analysis algorithms may also create intermediate segments between change points. This results in additional work (causing the algorithm to be $O(n^2)$, as described above), and may result in more segments than are useful for monotonicity analysis.

Consider the data in Table 1, which shows the number of segments created for a time series created by a damped sin with noise function, as in Fig. 7, by three algorithms: Algorithm 1, a sliding window algorithm using linear splining (which is provided in the Appendix), and a sliding window algorithm using linear regression. All three algorithms produce a number of segments which is (empirically) linear in the number of points, but the sliding window algorithm always at least 1.5 times as many segments as Algorithm 1– the multiple decreases as the amplitude of the sin wave decreases and the frequency decreases–while the linear regression algorithm consistently creates about 2/3 as many segments as points.

The $O(n^2)$ nature of the sliding window algorithm is evident in Fig. 8.

To restate: the quasi-monotonicity analysis of Algorithm 1 and the techniques described in Keogh *et al.* are really doing different things. The linear techniques are trying to minimize the least squares error (the L_2 norm), while the quasi-monotonicity analysis tries to minimize the maximum error (the L_∞ norm). Additionally, the techniques described in Keogh *et al.* produce additional segments which are not of use for analysing monotonicity, and (as a result) execute more slowly.

Limitations of quasi-monotonic segmentation

Quasi-monotonic segmentation of measurements as produced by Algorithm 1 will only measure changes of direction (up/down) and is oblivious to rates of change (higher derivatives). Further, δ-flat segments are always incorporated into increasing or decreasing segments. When a δ-flat

Points	Alg. 1	Sliding Window	Regression
1600	9	20	1065
3200	16	37	2099
4800	21	49	3191
6400	27	59	4250
8000	34	65	5361
9600	40	77	6361
11200	48	82	7487
12800	54	91	8511
14400	59	97	9522
16000	66	107	10693
17600	72	114	11668
19200	79	120	12729
20800	85	126	13844
22400	91	131	14806
24000	97	137	15952
25600	104	143	16995
27200	110	153	18074
28800	116	156	18998
30400	123	163	20265
32000	130	169	21343
33600	136	175	22478
35200	143	179	23427
36800	149	188	24499
38400	156	190	25697
40000	162	201	26611

Table 1: Number of segments produced by three algorithms as the number of points increase; with $\delta = 0.1$

segment is at the end of an segment meeting a new segment, it is ambiguous which increasing/decreasing segment it is part of. Algorithm 1 uses the L_∞ (maximum) error measure which can be both a positive and a negative: it will be sensitive to even a single measure above δ and doesn't average out the effect. However, short segments can be readily identified as noise-related in some applications or could be interpreted as being significant in other applications.

Conclusion

In reactive control, if segmentation of the state variables into time intervals is to be used as episodic functions, then linear time algorithms must used and soft-realtime processing must be possible. Segmentation in monotone intervals is a natural approach, but using straight-forward algorithms like linear splining is unnecessarily expensive, requires post-processing to aggregate segments having the same sign, assuming no segment has near zero slope, and most likely require approximation algorithm since optimal linear splining is not possible in linear time. On the other hand, the quasi-monotone approach we described is mathematically elegant and computationally convenient.

References

Bonasso, R. P.; Firby, J.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiences with an architec-

ture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence* 9(2/3):237–256.

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

Firby, R. J. 1989. Adaptive execution in complex dynamic worlds. Technical Report YALEU/CSD/RR #672, Computer Science Department, Yale University.

Freed, M.; Matessa, M.; Remington, R.; and Vera, A. 2003. How Apex automates CPM-GOMS. In *Proceedings of the 2003 International Conference on Cognitive Modeling*.

Keogh, E.; Chu, S.; Hart, D.; and Pazzani, M. 1993. Segmenting time series: A survey and novel approach.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. In *Data Mining and Knowledge Discovery*, 259–289. Kluwer Academic Publishers.

Myers, K. L. 1996. A procedural knowledge approach to task-level control. In Drabble, B., ed., *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 158–165. AAAI Press.

Pell, B.; Bernard, D. E.; Chien, S. A.; Gat, E.; Muscettola, N.; Nayak, P. P.; Wagner, M. D.; and Williams, B. C. 1997. An autonomous spacecraft agent prototype. In Johnson, W. L., and Hayes-Roth, B., eds., *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, 253–261. New York: ACM Press.

Sliding window algorithm

Algorithm 2 presents a segmentation algorithm for monotonic segmentation using a sliding window technique of the sort described in (Keogh *et al.* 1993).

Algorithm 2 Sliding window linear spline computation algorithm.

INPUT: two arrays t and m, where t contains time stamps and m contains measure values
INPUT: some error tolerance δ
OUTPUT: a queue Q containing indexes of nodes where we segment
$Q \leftarrow$ empty queue
add 1 to Q
$i \leftarrow 1$
while true **do**
    for j in $\{i+1, \ldots, n\}$ **do**
        Let f be the linear function with $f(t_i) = m_i$ and $f(t_j) = m_j$
        Let $\epsilon = \max\{|f(t_k) - m_k|\}$ for $k \in \{i+1, \ldots, j-1\}$
        if $\epsilon > \delta$ **then**
            $i \leftarrow j - 1$
            add i to Q
            exit for loop
        if $i = n$ **then**
            add i to Q
            exit algorithm with Q