Stream Processing Algorithms that Model Behavior Change

Agostino Capponi*
Computer Science Department
California Institute of Technology
USA
acapponi@cs.caltech.edu

Mani Chandy*
Computer Science Department
California Institute of Technology
USA
mani@cs.caltech.edu

Abstract – ***This paper presents algorithms that fuse information in multiple event streams to update models that represent system behavior. System behaviors vary over time; for example, an information network varies from heavily loaded to lightly loaded conditions; patterns of incidence of disease change at the onset of pandemics; file access patterns change from proper usage to improper use that may signify insider threat. The models that represent behavior must be updated frequently to adapt to changes rapidly; in the limit, models must be updated continuously with each new event. Algorithms that adapt to change in behavior must depend on the appropriate length of history: Algorithms that give too much weight to the distant past will not adapt to changes in behavior rapidly; algorithms that don't consider enough past information may conclude incorrectly, from noisy data, that behavior has changed while the actual behavior remains unchanged. Efficient algorithms are incremental – the computational time required to incorporate each new event should be small and ideally independent of the length of the history.***

Keywords: stream processing, parameter estimation, sense and respond systems, incremental computation, behavioral change.

1 Introduction

1.1 Overview

A sense and respond system (1) estimates the history of global states of the environment from information in streams of events and other data, (2) detects critical conditions – threats or opportunities – by analyzing this history, and (3) then responds in a timely fashion to these conditions. A sense and respond system can be specified by a set of rules where each rule is a pair: a condition and a response. The condition is a predicate on the history of estimated global states and the response is an action. A sense and respond system learns about its environment from information in event streams and other data sources. The environment is represented by a model and algorithms continuously update models as new events arrive on an event stream. The learned model is used to determine best responses to critical conditions.

In some problem areas, critical conditions are signaled by changes in behavior of a system. Information–assurance systems monitor applications, such as email, and usage of information assets, such as files, to get alerts when changes in behavior – that may signal misuse – are detected. Financial applications detect potential non–compliance to regulations by monitoring changes in patterns of income and expenditure. Pharmaceutical companies monitor changes in patterns of problems reported by customers to detect potential problems with products.

These applications develop and continuously update models of system behavior. As system behavior changes, model parameters change too, and significant changes in parameters indicate probable changes in behavior. The systems of interest consist of groups of entities. In the pharmaceutical example, for instance, the system consists of all customers who have bought a product, and the events in the system are activities by customers such as the logging of a complaint or an indication of satisfaction. The system generates a stream of events – the sequence of events generated by all the customers collectively. Successive events may be generated by different entities; for example, a complaint may be registered by one customer followed by complaints by many other customers before an event is generated by the first customer again. Filtering algorithms, such as the Kalman filtering algorithm, assume a model of the evolution of state over time, such as

$$\begin{cases} x_k &= f(x_{k-1}) + v \\ y_k &= g(x_k) + w \end{cases} \quad (1)$$

where x_k is the state of the system at time k, y_k is a signal at time k, and v and w are random variables. In the examples considered in this paper, such relationships between the signals at successive times may not exist because the signals are generated by different

*Caltech Computer Science Technical Report Caltech CSTR:2005.004, February 2005

entities. Therefore, filtering algorithms are less appropriate than other kinds of statistical algorithms.

1.2 Model of Behavior

A signal is represented by a point in a multidimensional space where the dimensions are attributes of behavior. The dimensions in the pharmaceutical example dealing with blood sugar monitors includes the age of the product, the strength of the battery, the type of erroneous reading, length of experience with this type of product and so on. Our algorithm is fed a stream of signals (sometimes called event information) and thus is continuously fed new points in this space. A model is a surface in this space, and a metric of the fitness of the model is the average mean-square distance of points from the surface.

The system may change its behavior and the change may be gradual or abrupt. In the pharmaceutical example, a change may be caused by the introduction of a defective component in some batches of the product. The signals that are generated after the change reflect the changed behavior. The algorithm updates model parameters with each signal it receives with the goal of maintaining an accurate model at all times. Fig. 1 illustrates a changing behavior in 3–dimensional space. The black dots represent signals generated before a change and the circles represent signals after the change where the collection of black dots falls near one hyperplane and the white dots near another.

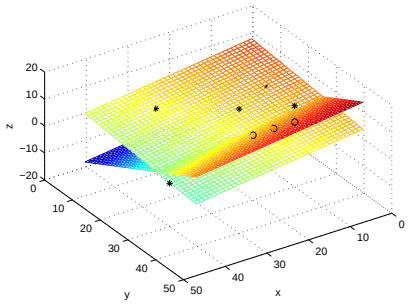

Fig. 1: Two surfaces corresponding to two different models of behaviour. The points marked with o belong to one surface and points marked with * to the other.

1.3 Incremental Algorithms for Stream Processing

A stream processing algorithm takes a sequence of events as its inputs and scans this sequence only once in increasing order of occurrence [**?**, **?**]. The computational complexity measures are the space used by the algorithm and the time required to process each new event. An *incremental* algorithm is one for which the time required to fuse a single event with the history of events is small compared with the length of the history; we seek algorithms in which the time is independent of the length of the history or is a low–degree polylog or polynomial. For example, consider the computation of a moving–point average over a window. When the window moves by one value, the computation of the new moving–point average can be done in time independent of the length of the window: merely add the leading–edge value and subtract the trailing–edge value. We present algorithms for adapting to behavorial change that come close to being incremental.

1.4 Related work

The field of adaptive stream processing has received much attention recently [**?**, **?**, **?**, **?**]. For many applications today, e.g. sensor network applications, data are produced continuously and algorithms must provide accurate answers in real time. Some of the main techniques dealing with continuous stream processing are *summarization* which provides concise representations of a data set using data structures at the expense of accuracy in the answer and *adaptive algorithms* which update the structure as new data arrive in a reasonable amount of time. An adaptive approach proposed in [**?**] processes continuous streams and provide answers within a guaranteed approximation factor. Another interesting approach in [**?**] uses sensor network observations to provide estimations for the temperature of the surrounding within a user defined confidence interval. Correlation between consecutive sensor readings are used to achieve fast computation. The algorithm described in this paper extends earlier work on stream processing to deal with changing behaviors.

1.5 Types of Models

A popular way of estimating a model that fits a set of points is regression. One of the variables of the model is identified to be a *dependent variable* and the other variables are independent variables. A model predicts the value of the dependent variable given the values of all the independent variables. The differences between the values of the dependent variables in the actual set of data points and the values predicted by the model are the errors of the model.

We are not trying to predict one variable given values of others; we are trying to estimate models of behavior as behaviors change. For our purposes, all variables are equivalent. One approach when dealing with n variables is to use n separate regression models, where each regression model singles out one of the variables to be the dependent variable. Changes in any of the n regression models signifies a change in behavior.

An alternate approach is *orthogonal* regression [**?**] in which error is defined as the *minimum* distance of a data point from the surface. As a trivial illustrative example consider a model with two parameters x and y, and where the model is represented by the line $x + y = 1$. Consider a data point $(1, 1)$. Let y be the dependent variable in a regression model. The value of y predicted by the model when $x = 1$ is $y = 0$. Hence,

the error in the standard regression model corresponding to point $(1, 1)$ is the difference (1) between the actual (1) and predicted (0) values. By contrast, the error in the orthogonal regression model is the shortest distance from the point $(1, 1)$ to the line and this error is $\frac{1}{\sqrt{2}}$.

Another model is the degenerate case of a surface where the model is represented by a single point. As in the general case, the error corresponding to a data point is the minimum distance of the data point from the surface which, in this case, degenerates to the distance of the data point from point p that represents the model. The problem simplifies to finding the point p that minimizes total error. As signals arrive on the stream, point p is recomputed with each new signal, and a behavior change corresponds to a significant change in the value of p.

If all the data points are uniformly distributed in a sphere around a point, then a model which is a single point is better than a model which is a surface. Indeed, in this case *every* hyperplane through point p is optimal; thus the hyperplanes carry no more information than the single point p. If, however, the points lie near an extended surface, then a surface model is better than a point model. One solution is to have a general model where the number of dimensions of the surface can vary; for instance in a 3–parameter (and hence 3–dimensional space) model, the modeling surface could change continuously between being a plane, a line and a point. In this paper we restrict ourselves to the case where points fall closer to a surface than to a point.

Regression models are represented by equations in which the independent variables can get arbitrarily large. In many systems, the ranges of variables are limited by physical constraints. For example, in an information assurance system monitoring file accesses there is a physical limit to the rates at which files can be accessed by single process. Limiting ranges of variables changes error estimates. Consider the trivial illustrative example given earlier with variables x and y, where the model is $x + y = 1$. Consider the error due to a data point $(0, 2)$ for two cases: (a) the unlimited variable range $[-\infty, +\infty]$ and (b) the ranges of x and y are limited to $[0, 1]$. The error in the former case is the minimum distance of the point from the line, and this distance is $\frac{1}{\sqrt{2}}$. The error in the latter case is the minimum distance of the point from the line *segment*, and this is the distance (1 unit) from the point $(0, 2)$ to the end $(0, 1)$ of the segment.

A change in behavior may be indicated by a change in the ranges of variables even if there is no change in the surface that represents the model: A change in the length of a line segment may be significant even if the line itself does not change. In this paper we do not consider this issue.

2 Theory

2.1 Exponential smoothing and sliding window

A key issue is that of determining the weight to be given to old information in estimating models: too much weight given to old information results in algorithms that do not update models rapidly; but, the more information that is used, the better the estimates in the presence of noise. Popular algorithms for dealing with different emphases on newer and older data are sliding window and exponential smoothing. A sliding window protocol with window size W estimates a model using only the most recent W data points; it treats all W data points in the window with equal weight, and effectively gives zero weight to points outside the window. An exponential smoothing algorithm with weight α gives a weight of α^k to a data point k units in the past where α is positive and at most 1; thus an algorithm using a small value of α "forgets faster". We refer to α as the *forgetting factor*.

Incremental stream-processing algorithms can be obtained for both sliding window and exponential smoothing. Appendix 2 shows that this is done for the exponential smoothing case. The proof for the sliding window is similar.

An important issue is that of determining the appropriate α to use at each time T. The value of α can range from 1 (in which case all points from 0 to T are weighted equally) to 0 (in which case only the data point that arrived at T is considered). Small values of α adapt to changes rapidly because they give less weight to old data whereas large values of α are better at smoothing out noise.

One approach is to change the relative weights given to old and new data when a change is estimated. For instance, suppose the algorithm estimates at time 103 that with high probability a change occurred at time 100; the algorithm then reduces the weight given to signals received before 100 and increases the weight given to signals received after 100. A disadvantage of this approach is that if the algorithm estimates that a behavioral change has taken place when, in reality, no change has occurred, then the algorithm discards valuable old data needlessly. The same approach can be used with sliding windows.

2.2 Experimental Setup

At any point in time, the behavior of a system is captured by a model which is represented by a bounded surface. Our algorithm attempts to estimate the true model given a sequence of noisy signals. We call the model and the surface estimated from signals the estimated values as opposed to the "true" values. The true model is changed at some point in time during the experiment and we evaluate whether the estimated model follows the true model accurately.

At each point in time, a signal is generated as follows. First a point q is generated on the true bounded

surface randomly, then a scalar error term e is generated randomly using the given error distribution, and finally a data point r is generated where $r = q + e \cdot v$ where v is the unit normal to the true surface at point q.

2.2.1 Algorithm

Input at time T: A sequence of $T-1$ points that arrived in the time interval $[0, T-1]$ and a new point that arrived at time T.
Output at time T: An estimate of the surface – a hyperplane in a linear model – at time T.
Goal: Minimize the deviation between the estimated and true surfaces.

The true model changes over time, and the manner of change is described separately.

2.2.2 Angle between planes

One measure of efficacy of fit of the estimated model to the true model is the angle between the surfaces. The inner product of the unit normals to the hyperplanes representing the estimated and true models is the cosine of the angle between the hyperplanes, and we use this as a measure of goodness. The cosine is 1 if the hyperplanes are parallel and is 0 if they are orthogonal.

2.2.3 Comparison of distances of points from true and estimated surfaces

Another measure of goodness of fit is represented by the differences in distances of data points from the true and estimated surfaces. Let $D_{k,t}$ be the minimum distance of the data point that arrived at time k from the true surface at time t. Recall that $d_{k,t}$ is the minimum distance of the data point that arrived at time k from the surface estimated at time t. Let E be defined as follows:

$$E = \sum_k (D_{k,k} - d_{k,k})^2 \tag{2}$$

Now E is a measure of goodness – the smaller the value of E the better the resulting estimate. We call E the *relative distance error*. Notice that this parameter can be nonzero even if the true and estimated hyperplanes are parallel because this error term is zero if and only if the two hyperplanes are the same.

Appendix 1 shows that the solution that minimizes E can be obtained by solving the convex optimization problem of the minimum eigenvalue or a square system of equations. For now we present results using only the second method.

3 Experiments

We restrict ourselves to linear models; thus, the surface is a hyperplane in a multidimensional space. In each of our experiments we assume that we are given the true model; we generate noisy data from the true model; compute an estimated model from the noisy data; and, compare the true and estimated models. At an arbitrary point in time, we change the true model. Noisy data is now generated using the new true model. (The distribution of noise terms is assumed to remain unchanged even though the true model changes.) Since the estimation algorithm has no specific indication that the true model has changed, the algorithm uses data before the change as well as points after the change. Therefore, the estimated model may not be close to the new true model during and immediately after the change. We would like the estimated model to become close to the true model as the time since the last change increases.

The set of experiments has been restricted to 2–dimensional surfaces. Noise is assumed to be Gaussian. This is not a necessary assumption; in fact the algorithm may be applied with any white noise vector. We consider cases where the noise is low ($\sigma^2 = 1$) or high ($50 \leq \sigma^2 \leq 100$). We study the effect of different values of α on the accuracy of the model. We choose values of α as follows. We pick a positive integer w that we call the window size (not to be confused with the window in the sliding window algorithm) and a positive number γ (which is at most1) that we call the *threshold*. The value of γ was set to 0.5 in the experiments. Given w and γ we pick an α such that the total weight assigned to all the signals w or more time units into the past is exactly (up to a rounding error) γ. For instance, if $w = 4$ and $\gamma = 0.5$ then we know that the first w signals have a total weight of $\frac{1}{2}$, the next w have a total weight of $\frac{1}{4}$, and the next w have a weight of $\frac{1}{8}$ and so forth.

3.1 Experiments with changing behaviors

We ran many experiments. In each experiment a change occurs after a certain number of time points. Each change is a translation followed by a rotation of the (true) plane. Fig. 2 illustrates the 2D case; each line is associated with a behavior change. Here we report on the following experiments

- The true model is changed after 500 time points. The translation is 0.75 and the rotation is 10 degrees.
- The true model is changed after 50 time points. The translation is 0.02 and the rotation is 1 degree.
- The true model is changed after 5 time points. The translation is 0.0018 and the rotation is 0.1 degrees.

Each experiment was run for several thousands of points and thus covered many changes of the true model. For ease of visualization, we only show 1500 points in the figures, however, the same pattern occurs for larger numbers of points.

Fig. 3 shows the cosine and the angle between the true and estimated hyperplanes at different points in

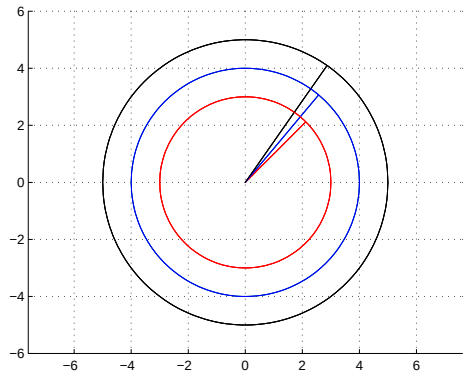

Fig. 2: The model of behavioral change used in the performed experiments. When the model changes points are generated using a different line

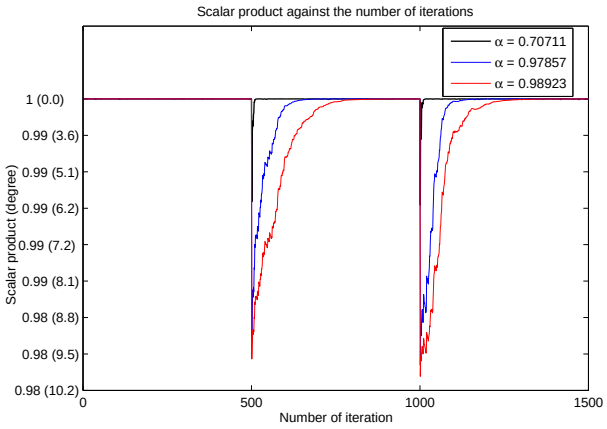

Fig. 3: The scalar product between the vector of estimated and actual coefficients. A change occurs every 500 iterations. Threshold = 0.5, Noise variance = 1

time for the low variance case. Fig. 4 shows the relative distance error as a function of time for the low variance case. The next two figures show results for the high variance case. The angle between the true and estimated planes increases sharply at the point of the change and then decreases. The angle at the instant of change is larger for higher values of α and this is not suprising because higher values of α give greater weight to pre–change data. Also, algorithms with higher values of α take longer, after a change, to reduce the error.

Higher values of α are less susceptible to noise. This is not apparent from the figures in the low–variance case, but is readily apparent in the high–variance case. Indeed, the algorithm with low α cannot distinguish between a change to the true model and noise in the case of high noise variance. This suggests, as expected, that only high values of α should be used in the case of high noise whether the true model is stationary or not. As discussed earlier, an approach is to adapt the relative weights given to old and new data when the

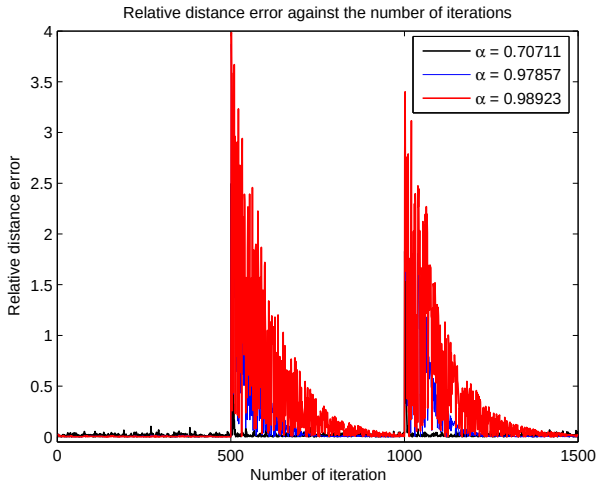

Fig. 4: The relative distance error between the estimated and actual plane. A change occurs every 500 iterations. Threshold = 0.5, Noise variance = 1

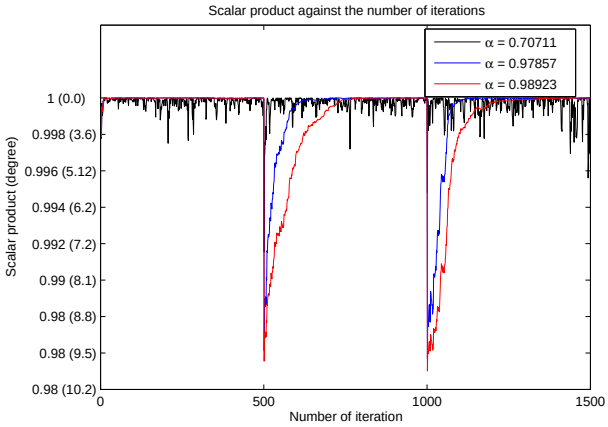

Fig. 5: The scalar product between the vector of estimated and actual coefficients. A change occurs every 500 iterations. Threshold = 0.5, Noise variance = 100

algorithm estimates that a change has occurred.

When changes occur frequently as in Fig. 7-10, then large values of α are not very appropriate since they give large weight to data generated according to different past models. High accuracy is given by large α values, but only when the amount of data generated according to the same model is high; when this does not occur, then smaller α give better performance.

The experiments are explained quite simply by considering the function $\sum_{i=1}^{c} \alpha^{k-i} - \sum_{i=c+1}^{k} \alpha^{k-i}$, where c denotes the time that the true model changes. The first term is the total weight assigned to pre–change signals and the second to post–change signals. Immediately after the change, the pre–change weights are larger because there are fewer post–change signals. Likewise, the higher the value of α the greater the weight given to pre–change signals.

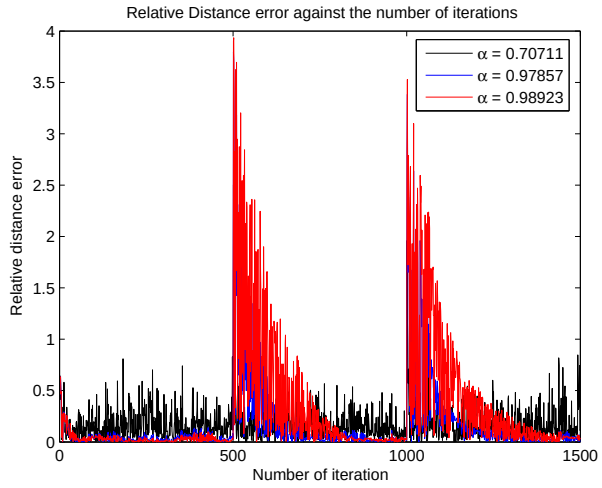

Fig. 6: The relative distance error between the estimated and actual plane. A change occurs every 500 iterations. Threshold = 0.5, Noise variance = 100

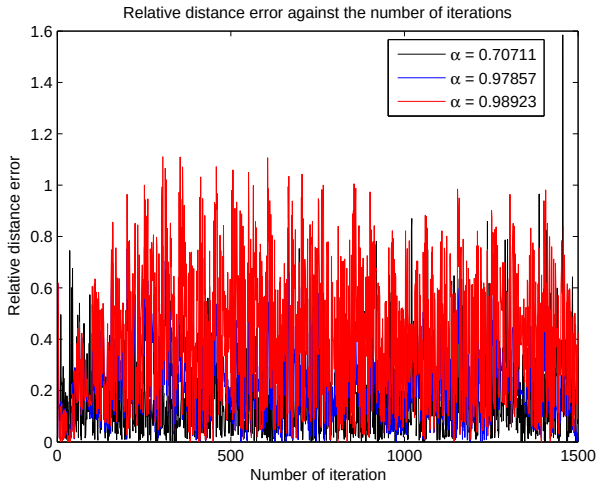

Fig. 8: The relative distance error between the estimated and actual plane. A change occurs every 50 iterations. Threshold = 0.5, Noise variance = 100

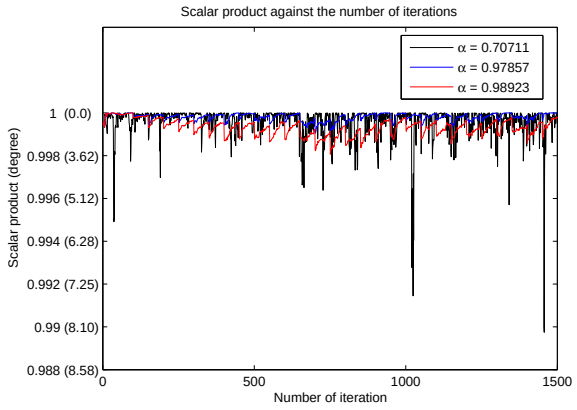

Fig. 7: The scalar product between the vector of estimated and actual coefficients. A change occurs every 50 iterations. Threshold = 0.5, Noise variance = 100

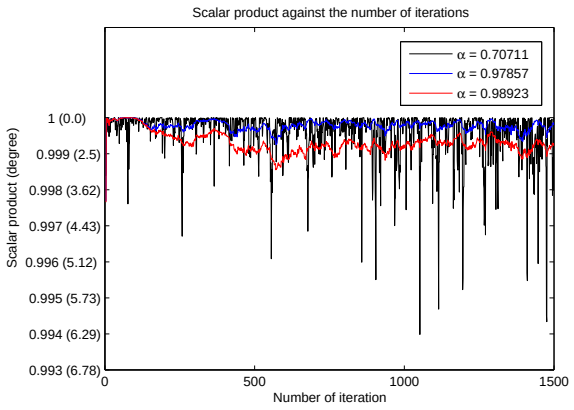

Fig. 9: The scalar product between the vector of estimated and actual coefficients. A change occurs every 5 iterations. Threshold = 0.5, Noise variance = 100

3.2 Adaptive Algorithms

Adaptive algorithms change the relative weights assigned to older and newer data when a change is detected. The figures show that when a change is abrupt, the change can be detected readily and adaptive algorithms work well. If the change is gradual but frequent, then for large α values the algorithm may come close to complete recover, but never recover fully. This clearly appears in Figures 7, 8, 9 and 10.

An alternative strategy is to compare the model at time T with the models at previous times t where t ranges from T to $T-M$ where M is a constant window size. So far, we have only discussed the case where $M = 1$ which is sufficient for significant substantial changes. If the algorithm detects a change between any model at a previous time t and the current time T, then the algorithm adapts the weights, giving greater weight to signals after time t and less to signals before time t. These experiments are ongoing, and we will report on them in the full version of the paper.

3.2.1 Number of steps for convergence for different numbers of parameters and noise

We show in the appendix that the computational complexity for handling each new signal value is a constant independent of the length of the history, except for the case of computing the eigenvalue. In our experiments we compute solutions iteratively using the Matlab routine `fsolve` using the solution obtained at time t as the initial guess for the computation at time $t+1$. Our rationale for choosing such iterative algorithms is that when there is no change in the true model, we expect little or no change in the estimated model, and hence this method of obtaining an initial guess should be very good. We found that about 12 steps were required to converge from a random guess whereas only 4 steps were needed by using the time t value as the initial guess for the time $t+1$ computation.

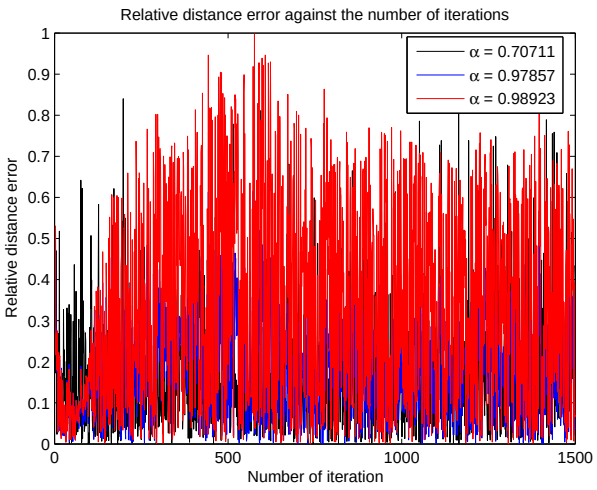

Fig. 10: The relative distance error between the estimated and actual plane. A change occurs every 5 iterations. Threshold = 0.5, Noise variance = 100

4 Conclusions and further Work

We have presented an algorithm for estimating the parameters of a linear model. The algorithm combines the method of orthogonal regression with exponential forgetting to compute the best estimate. We have shown that all the computation can be done incrementally and using a very small amount of memory. For the tested models we have discussed the recovery of parameters as function of the frequency of the behavioral change of the model. In the future we plan to study incremental iterative algorithms for finding the minimum eigenvalue of the matrix S^* in Appendix 1 and consequently obtaining the eigenvector corresponding with the best estimate of parameters. Though the latter problem is convex and therefore easily to solve with standard packages for reasonable matrix dimensions, it may still require an intensive CPU time if all the history is taken into account. We also plan to compare our approach with standard regression techniques.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Invited paper in Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002)*. ACM, 2002.

[2] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. *Conference on Innovative Data systems Research*, 2003.

[3] Sirish Chandrasekaran and Michael J. Franklin. Remembrance of streams past: Overload-sensitive management of archived streams. *VLDB*, 2004.

[4] D.Eberly. *3D Game Engine Design*. Morgan Kaufmann, 2001.

[5] Amol Deshpande, Carlos Guestrin, Samuel R.Madden, Joseph M.Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004.

[6] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004*, pages 180–191. VLDB, 2004.

[7] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: An architectural status report. *IEEE Data Engineering Bulletin*, Vol. 26(1), March 2003.

[8] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. *Proc. of the ACM Intl Conf. on Management of Data*, 2003.

Appendix 1: Method of the minimum eigenvalue

The best fitting hyperplane with respect to a set of points is obtained solving the following minimization problem:

$$\begin{gathered} \min \textstyle\sum_{i=1}^{T} \alpha^i \cdot (\boldsymbol{a}' \cdot \boldsymbol{y}[i] + a_0)^2 \\ \text{subject to } \|\boldsymbol{a}\| = 1 \end{gathered} \tag{3}$$

where T denotes the number of points, $\boldsymbol{a} = (a_1, a_2, \ldots, a_p)$ is the vector of unknowns, a_0 is the offset of the hyperplane from the origin, $\alpha < 1$ is the weighting factor and the $\boldsymbol{y}[i]$ denotes the point received at time i. The larger the weight, the more recent the point. Using Lagrangian multipliers, the problem can be formulated as finding the $p+1$ parameters $a_0, a_1, \ldots, a_p$ which minimize the expression

$$\begin{aligned} E \;=\; & \textstyle\sum_{i=1}^{T} \alpha^{T-i} \cdot [(\boldsymbol{a}' \cdot \boldsymbol{y}[i]) + a_0]^2 + \\ & - \lambda \cdot (\textstyle\sum_{i=1}^{T} a_i^2 - 1) \end{aligned} \tag{4}$$

The gradient vector $\nabla(E) = \left[\frac{\partial E}{\partial a_1}, \frac{\partial E}{\partial a_2}, \ldots, \frac{\partial E}{\partial a_k}\right]'$ is defined by

$$\begin{aligned} \nabla(E) \;=\; & 2 \cdot (\textstyle\sum_{i=1}^{T} \alpha^{T-i} \cdot \boldsymbol{y}[i] \cdot \boldsymbol{y}[i]') \cdot \boldsymbol{a} + \\ & + 2 \cdot (\textstyle\sum_{i=1}^{T} \alpha^{T-i} \cdot \boldsymbol{y}[i]) \cdot a_0 + \\ & - 2 \cdot \lambda \cdot \boldsymbol{a} \end{aligned} \tag{5}$$

while the expression for the derivative with respect to a_0 is

$$\frac{\partial E}{\partial a_0} = \sum_{i=1}^{T} 2 \cdot \alpha^{T-i} \cdot (\boldsymbol{y}[i]' \cdot \boldsymbol{a} + a_0) \quad (6)$$

For simplicity of notation, let $\boldsymbol{S} = \sum_{i=1}^{T} 2 \cdot \alpha^{T-i} \cdot (\boldsymbol{y}[i] \cdot \boldsymbol{y}[i]')$. Furthermore, let $\boldsymbol{s} = \sum_{i=1}^{T} \alpha^{T-i} \cdot \boldsymbol{y}[i]$ and $r = \sum_{i=1}^{T} \alpha^{T-i}$.

Setting eq. (6) to zero we have the constraint that $a_0 = -\boldsymbol{s}' \cdot \frac{\boldsymbol{a}}{r}$. Substituting the expression for a_0 in eq. (5) we obtain the following condition:

$$(\boldsymbol{S} - \frac{1}{r} \cdot \boldsymbol{s} \cdot \boldsymbol{s}') \cdot \boldsymbol{a} - \lambda \cdot \boldsymbol{a} = 0 \quad (7)$$

It is well known that the eigenvector associated with the minimum eigenvalue of $\boldsymbol{S}^* = (\boldsymbol{S} - \frac{1}{r} \cdot \boldsymbol{s} \cdot \boldsymbol{s}')$ corresponds with the best estimate for $\boldsymbol{a}$.

Appendix 2: Derivation of Incremental Computations

We have shown in Appendix 1 the method to find the optimal estimate for $\boldsymbol{a}$ and a_0. Many iterative and efficient methods for finding eigenvalues exist in the literature. In the performed experiments, we use a different approach. We consider the $p+2$ dimensional square system defined by

$$\begin{cases} S \cdot \boldsymbol{a} + \boldsymbol{s} \cdot a_0 - \lambda \cdot \boldsymbol{a} = 0 \\ \boldsymbol{s}' \cdot \boldsymbol{a} + r \cdot b = 0 \\ \boldsymbol{a}' \cdot \boldsymbol{a} = 1 \end{cases} \quad (8)$$

The existence of one solution is guaranteed by the argument in Appendix 1. However, the returned solution will correspond with an eigenvalue of S^*, which is not necessarily the minimum eigenvalue. Consequently, the corresponding eigenvector does not necessarily correspond with the best estimate for the unknown vectors of parameters. In the performed experiments the trust region dogleg method implemented by the standard Matlab routine `fsolve` has been used to solve the system. The quality of the recovered solution has been shown to be very satisfactory (see Section 3). The system of equations to be solved at each step is computed incrementally using the following method.

A data structure which summarizes all points received up to step i and only incorporates the point received at step $i+1$ is used. Doing so, the equations at step $i+1$ can be computed in an amount of time which is constant with respect to the number of received points.

Using eq. (5) and (6) it is easy to see that the gradient $\nabla(E)$ at time $T+1$ can be defined as

$$\begin{aligned} \nabla(E) = & (2 \cdot \alpha \cdot \boldsymbol{S}_T + \boldsymbol{y}[T+1] \cdot \boldsymbol{y}[T+1]') \cdot \boldsymbol{a}' + \\ & + 2 \cdot (\alpha \cdot s_T + \boldsymbol{y}[T+1]) \cdot a_0 + \\ & - 2 \cdot \lambda \cdot \boldsymbol{a} \end{aligned} \quad (9)$$

while the partial derivative with respect to a_0 is

$$\begin{aligned} \frac{\partial E}{\partial a_0} = & \quad 2 \cdot \boldsymbol{a}' \cdot (\alpha \cdot \boldsymbol{s}_T + \boldsymbol{y}[T+1]) + \\ & + 2 \cdot a_0 \cdot (\alpha \cdot r_T + 1) \end{aligned} \quad (10)$$

where $\boldsymbol{S}_T$ denotes the matrix $\boldsymbol{S}$ at time T, s_T is the vector $\boldsymbol{s}$ at time T and r_T is the value of r at time T. Notice that all these parameters have been defined in Appendix 1.

After solving the system of equations at time $T+1$, we can set set $\boldsymbol{S}_{T+1} = \alpha \cdot \boldsymbol{S}_T + \boldsymbol{y}[T+1] \cdot \boldsymbol{y}[T+1]'$, $s_{T+1} = \alpha \cdot s_T + \boldsymbol{y}[T+1]$ and $r_{T+1} = \alpha \cdot r_T + 1$. Doing so, we can incrementally compute the system at step $T+2$.

Appendix 3: Other norms than $L2$

Appendix 2 shows how the calculation of derivatives can be done incrementally for the case when the $L2$ norm is used a distance criteria; in fact the problem can be formulated as

$$\begin{aligned} & \min \sum_{i=1}^{T} \alpha^i \cdot \|(\boldsymbol{a} \cdot \boldsymbol{y}[i] + a_0)\| \\ & \text{subject to } \|\boldsymbol{a}\| = 1 \end{aligned} \quad (11)$$

It can be shown that the systems of equations can be computed incrementally also in the case when the minimization is done using the Ln norm, for $n > 2$, i.e.

$$\begin{aligned} & \min \sum_{i=1}^{T} \alpha^i \cdot (\|(\boldsymbol{a} \cdot \boldsymbol{y}[i] + a_0)\|_n) \\ & \text{subject to } \|\boldsymbol{a}\| = 1 \end{aligned} \quad (12)$$

Due to space limitations we do not discuss the derivation here, but we restrict ourself to the following considerations. The key step for the incremental computation in the $L2$ case is the use of the matrix $\boldsymbol{S}_T$. The k^{th} column of this matrix, $\boldsymbol{S}_T^k$, represents the sum of all vectors $\boldsymbol{y}[i]$, weighted by the k^{th} component of each vector at time T. A generalization of this idea leads us to the use of hypermatrices consisting of n different entries for the case when the Ln norm is used. The dimension of each entry is p, with p denoting the dimension of the vector $\boldsymbol{y}[i]$. Hence, the dimension of an n hypermatrix is p^n, which can be considered constant with respect to the number of points y. The conclusion is that the use of Ln norms, $n > 2$, only introduces a multiplicative factor p^{n-2} in the dimension of the space used by the algorithm to estimate the parameters with respect to the use of the $L2$ norm.