Adaptive Methods for Activity Monitoring of Streaming Data[1]

Vasundhara Puttagunta and Konstantinos Kalpakis
Computer Science & Electrical Engineering Department
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
{vputta1,kalpakis}@csee.umbc.edu

Abstract

Activity monitoring deals with monitoring data (usually streaming data) for interesting events. It has several applications such as building an alarm or an alert system that triggers when outliers or change points are detected.

We discuss desiderata for such a system. Then, assuming that the data can be modeled by linear models, we describe an adaptive incremental method for detecting outliers and change points in data streams. Our algorithm uses (a) intuitive criteria for labeling a data point as an outlier or as a change point, and (b) an adaptive incremental model estimation method. In this paper, we use a forgetting factor–based Recursive Least Squares algorithm for adaptive incremental model estimation. We also present experiment results using both simulated and real data, which show that our algorithms for change and outlier detection could accurately detect these events.

Keywords: streaming data, recursive least squares regression, recursive computations, change detection, outlier detection.

1 Introduction

Activity monitoring is a term that is loosely coupled with monitoring data for interesting (including alarming or strange) events or episodes. Depending on the application, this data could be of different types coming from a variety of sources – for example, financial data sources (credit card transactions, currency exchange rates, market trends), medical and epidemiological data sources (electrocardiograms, disease surveillances), environmental and scientific data sources (pollution and weather sensors, seismic and volcanic activity sensors, solar activity sensors), industry (temperature and pressure monitors, radio-active level monitors, sensors in aircrafts), computer systems and web(click streams, memory and resource usages, network load), telephone calls.

[1]Supported in part by NASA under Contract NAS5–32337 and Cooperative Agreement NCC5–315. Please send all correspondence to Dr. Kalpakis.

To give a flavor, we describe different kinds of activity monitoring applications.

Outlier detection is a challenging problem in datamining. A constant concern for a system administrator is intrusion detection. One approach would be by monitoring the resource usage – for example memory usage, or cpu usage. Pena et al [11] consider the problem of detecting abnormal behavior in aerospace data. Such problems can be formulated as outlier–detection problems. Intuitively an outlier is a data point that differs significantly from what is expected or predicted.

The dynamics of the data sources could be changing over time. These changes could come about gradually or abruptly. For example, in the case of machinery, the wear and tear gradually affects the performance. The changes could also be drastic, i.e. there could be significant changes taking place within a short duration of time. Such changes are not uncommon - external events could trigger a change in the behavior. For e.g. during the World War II, the population estimates [2] of the different states in the US changed dramatically. We call points at which such changes take place change points or events. Detecting change points has a lot of applications. For example, Guralnik and Srivastava [5] consider the problem of detecting events in highway traffic (for e.g. when there is a change from *light* to *heavy* to *congested* traffic). Raghavan et al [12] use individual customer profiles consisting of features such as number of sessions, length of sessions, timing of the sessions, etc to detect which customers are more likely to defect. Another very similar problem in activity monitoring is that of episode detection or time–series segmentation. The problem is to segment the time–series such that each segment represents a particular phase in behavior of the process.

As systems become more complex and as the amount of data that needs to be monitored increases, it becomes increasingly important to automate activity monitoring. Automated activity monitoring systems should have the following characteristics.

1. **Time and space complexity should be low.** Data is usually in the form of streams. In the case of sen-

sors, there could be severe limitations on the resources (CPU, power and memory). Even otherwise, there could be large amounts of data that could be streaming at fast rates (eg surveillance image data). Therefore, these computations should have a very small memory (preferably constant)

2. **Accuracy and timeliness.** As Fawcett and Provost [3] state, "the goal of activity monitoring is to issue alarms *accurately* and in a *timely* fashion". It may be critical to detect interesting events and issue alarms as soon as the interesting events occur. For e.g., one problem that electricity companies face has to do with solar bursts. Solar bursts occur for short durations of time during which switching off the power grids is critical. In addition, the alarms raised should be accurate. By accuracy we mean two things: (a) the number of false negatives as well as false positives is low, and (b) the localization of the outlier or change. In the above example, switching off the power grid is an expensive operation. Therefore, it is not a good idea to have a monitoring system that gives too many false alarms. The above two observations imply that activity monitoring should be done **online** in a near real–time manner.
3. **Adaptive.** In many cases, the nature of the data–source keeps changing over time. For example, the performance of a machine (in a factory) may vary with time due to the wear and tear or set up of the machine. Therefore, it is required that the computations are adaptive to these changes over time.

There is vast statistics literature in model estimation (see [9] and references therein) as well as outlier detection, change detection, and activity monitoring in several different domains (statistics, machine learning, security, automation, user–profiling). Most approaches for event and outlier detection in statistics (e.g. [13]) and data–mining do not consider streaming data. They deal with the case, where the entire data is presented in advance. For example, Knorr and Ng [8] attempt to find distance–based outliers in large datasets. Guthery [6] considers the problem of unconstrained piecewise regression which could be used for episode detection or time–series segmentation and gives a dynamic programming solution. Guralnik and Shrivastava [5] consider the problem of change detection, and they give a batch algorithm using a maximum likelihood criterion for segmentation, and extend it to the incremental case. Keogh et al [7] give an online algorithm for time–series segmentation. Yamanishi et al [14] give an algorithm for outlier detection using online unsupervised learning of a probabilistic model using a finite mixture model.

The approach we take is similar to that of Yi et al [15]. The data are assumed to be coming from a process that can be modeled using a linear model. Linear models of data streams can be estimated using least squares regression. We discuss different ways of using least recursive squares regression to perform change and outlier detection in an online and adaptive manner, and report experimental results of our approach.

The rest of this paper is organized as follows. Preliminaries on least squares are given in section 2. In section 3 we discuss our approach to detect outliers and change points given an online (and preferable adaptive) model estimation algorithm. We describe algorithms for adaptive online model estimation using least squares regression, and using a moving window or a forgetting factor approach. In section 4 we present experimental results. Conclusions are given in section 5.

2 Preliminaries

Linear models and model estimation using least squares regression (LSR), as well as recursive methods for model identification/estimation, have been studied extensively [9, 10]. Our approach is to use extensions of LSR to model the timeseries in an incremental and adaptive way while using little memory and time. In this section, we provide few essential preliminaries of LSR. We first describe the offline computation of LSR, and then we describe the forward recursion that enables one to estimate the regression coefficients in an incremental way.

Suppose the (scalar [2]) output $y(t)$ at time t of a timeseries is a linear combination of p input measurements at time t (or earlier, in case of auto–regression), $\varphi(t) = [\varphi_1(t)\varphi_2(t)\ldots\varphi_p(t)]^T$. Therefore we have

$$y(t) = \theta(t)^T\varphi(t) + v(t), \tag{1}$$

where $\theta(t)$ is the vector of model parameters and $v(t)$ is a random–variable (white noise). Given a sequence of $y(i)$ and $\varphi(i)$ for $i = 1, 2, \ldots, t$ (the offline case where all the data points are available before hand), the LSR method estimates a constant $\theta(t)$, given $\varphi(t)$, so that the mean squared error of the estimated output,

$$V_t(\theta) = \frac{1}{t}\sum_{k=1}^{t}[y(k) - \theta^T\varphi(k)]^2 \tag{2}$$

is minimized. In the offline version of LSR, the estimated model parameters are

$$\hat{\theta}(t) = \bar{R}(t)^{-1}B(t), \tag{3}$$

where $\bar{R}(t) = \sum_{k=1}^{t}\varphi(k)\varphi^T(k)$ is the variance–covariance matrix of the t inputs, and $B(t) = \sum_{k=1}^{t}\varphi(k)y(k)$ is the cross-covariance between the inputs and output.

The LSR estimate $\hat{\theta}(t)$ can also be obtained recursively [9]

$$\hat{\theta}(t) = \hat{\theta}(t-1) + \frac{1}{t}R^{-1}(t)\varphi(t)[y(t) - \hat{\theta}^T(t-1)\varphi(t)] \tag{4}$$

$$R(t) = R(t-1) + \frac{1}{t}[\varphi(t)\varphi^T(t) - R(t-1)] = \frac{1}{t}\bar{R}(t). \tag{5}$$

[2] Extensions to vector output $y(t)$ can be obtained easily.

Computing the inverse $R^{-1}(t)$, which is expensive and sensitive to numerical errors, can be avoided by using the *matrix–inversion* lemma [9]: given matrices A, B, C, and D of compatible dimensions, so that the product BCD and the sum $A + BCD$ exist,

$$[A+BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B+C^{-1}]^{-1}DA^{-1}, \quad (6)$$

provided all the required matrix inverses exist. Using the matrix inversion lemma in equation 5, and letting $P(t) = \bar{R}^{-1}(t) = \frac{1}{t}R^{-1}(t)$, one obtains the *recursive least squares* (RLS) algorithm [9]

$$\hat{\theta}(t) = \hat{\theta}(t-1) + L(t)[y(t) - \hat{\theta}^T(t-1)\varphi(t)] \quad (7)$$

$$L(t) = \frac{P(t-1)\varphi(t)}{1 + \varphi^T(t)P(t-1)\varphi(t)} \quad (8)$$

$$P(t) = P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{1 + \varphi^T(t)P(t-1)\varphi(t)} \quad (9)$$

$P(0)$ can be initialized to a diagonal matrix with large positive diagonal elements, or to the matrix that corresponds to the first few input/output elements of the timeseries. Note that if $P(t) \approx 0$, then the RLS algorithm stops learning/adapting; typically, when this happens, $P(t)$ is reset appropriately.

3 Outlier and Change Point Detection

In this section, we describe our method for detecting outliers and change points using estimated linear models of the timeseries, in an online adaptive manner.

First, we describe the algorithm that we use to detect outliers and change points over a stream when an incremental model estimation algorithm is given. Then, we describe different methods of estimating model parameters starting with the simple RLS algorithm described in the previous section. We discuss certain drawbacks of the RLS algorithm with respect to the adaptivity of the estimated models to changes in the data. We then describe ways of rectifying this.

Intuitively, an outlier is a point that is not *consistent* with points in its *neighborhood*. We say a point p is consistent with a set of points P, if the probability that p is generated by the model that generated P is high. If it is not consistent, then p is an outlier. In the case of timeseries (streams), the neighborhood of a point is its temporal neighborhood. A similar definition of an outlier has been used often in data mining literature (for e.g. see [8, 13]). Outliers have also been defined in other ways as well (eg. [1]). A change point is a point (in time) where the estimated models before and after it differ significantly. Intuitively, the occurrence of many outliers within a short span of time is suggestive of a change in the model of a timeseries. The objective is to detect outliers and change points accurately and timely.

High–level pseudocode for outlier and change detection. Let $\theta(t)$ be the estimated parameters of a linear model for the timeseries at hand at time t.

1. When $p(t) = [y(t), \varphi(t)]$ arrives at time t, we update the model $\theta(t-1)$ to get $\theta(t)$ using one of the incremental update algorithms described later.

2. Using $\theta(t)$ we decide whether or not $p(t)$ is a probable outlier (see below). In this case, the model $\theta(t)$ is reset to $\theta(t-1)$, eg. $p(t)$ is ignored with respect to model estimation.

3. If a certain number of recent points has been declared as outliers, we declare that a change has occurred at the first one of them (see below). Model estimation is started afresh from that point onwards.

Deciding a potential outlier.
We now describe the details of when a point is declared an "outlier" and when a point is declared as a "change point". Table 1 gives the definitions and description of the parameters that are used in our method.

1. Before a point is detected as an outlier/change, we require that at least *minDetectionWindow* points have arrived.

2. Upon the arrival of point $p(t) = [y(t), \varphi(t)]$, we compute the estimated error, $e(t) = |y(t) - \theta(t)^T\varphi(t)|$ for $p(t)$ using the updated model $\theta(t)$.

3. If there was a change point after $t-$*minChangeDistance*, then $p(t)$ is not an outlier. If $e(t)$ is $\leq$ *minErrorThreshold* then $p(t)$ is not an outlier. If $e(t)$ is $>$ *maxErrorThreshold* then $p(t)$ is an outlier, and go to step 5. Otherwise, we continue with the next step.

4. Compute the *errorThreshold* and the *errorOvershoots* at time t. If *errorOvershoots* $\leq$ *maxErrorOvershoots*, then $p(t)$ is declared as an outlier.

5. If $p(t)$ is marked as an outlier, $\theta(t)$ is reset to $\theta(t-1)$.

An point $p(t)$ that is declared an outlier could indeed be a change point. A change point is detected as follows.

1. If there was a change point detected after $t-$*minChangeDistance*, then there is no change.

2. If one of the following conditions below is true, then we declare a change: (a) the number of marked outliers in the interval [t-*changeOutlierWindow*, t] is $\geq$ *changeOutlierCount*, or (b) the *coefficientGain* at t is $\geq$ *gainThreshold*.

3. If a change is declared, then (a) the point p_c at time *t-changeResetWindow* is marked as a change point, (b) all points after it are unmarked (they are not outliers anymore), and (c) the model estimation is started afresh from p_c.

Table 1. Parameters for deciding outliers/changes. Parameters with * are user–provided.

Parameter	Definition
*minDetectionWindow**	at least *minDetectionWindow* points must arrive in the stream before a point is declared as an outlier/change point.
*minErrorThreshold**	the minimum error necessary to trigger outlier testing.
*maxErrorThreshold**	the minimum error threshold above which a point is declared as outlier right away.
*minChangeDistance**	the minimum distance between consecutive change points. This parameter controls the granularity of the changes with respect to time.
*windowSize**	the window size for outlier detection. It controls the number of recent points used in deciding whether a newly arrived point is an outlier.
errorThreshold	is the median of the estimated errors of the points within the interval [*t-windowSize*,*t*] that have not been marked as outliers, and it depends on t.
*outlierSensitivity**	is an outlier sensitivity factor in comparing the error estimate for a point with the errorThreshold. It controls the rate of false positives/negatives.
*overshootSensitivity**	is a sensitivity factor for claiming an error overshoot with respect to the errorThreshold.
errorOvershoots	is the number of points within the interval [*t-windowSize*,*t*] that have estimated error larger than *overshootSensitivity* $\times$ *errorThreshold*.
*maxErrorOvershoots**	the maximum number of error overshoots allowed for an outlier (if there too many outliers in the windowSize, then testing for a change point is done).
*changeOutlierWindow**	is the window size considered in testing for changes.
*changeOutlierCount**	is the minimum number of outliers in the among the last *changeOutlierWindow* points in required for declaring a change point.
coefficientGain	is the maximum percentage change of the parameters of $\theta(t)$ with respect to the parameters of $theta(t-1)$.
*gainThreshold**	a threshold for the coefficentGain that forces change detection.
*changeResetWindow**	the lag time in detecting a change. This controls the localization of the change point.

3.1 The Naive Approach

In this approach the RLS algorithm is used to compute the model parameters over all the data seen so far (unrestricted window).

Initially (for the first few points) this approach may be successful in detecting outliers/change points. As more points come in, RLS estimates the model parameters considering all the points seen so far, and thus it gives equal weight to all the points in the unrestricted window. As a result, the changes in the dynamics of the system may be hidden by the dominance of the older points. Figure 3.1 illustrates this phenomenon. As a result RLS by itself is not adequate. The problem is due to the fact that RLS tries to remember all the data points. Instead, if we can make the system forget older points, we should be able to detect outliers/change points better.

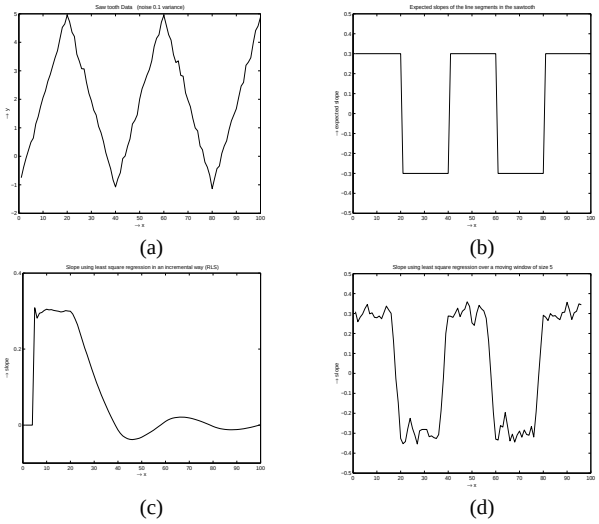

Figure 1. Effect of older points in RLS–based outlier/change point detection. (a) saw–tooth data stream. (b) slopes of the line segments in the saw–tooth. (c) slope estimate using RLS. (d) slope estimate using a moving window of size 5.

3.2 Moving Window Approach

One approach to *forget* older points is to have a moving window over the timeseries/ data stream and estimate the model over this window. At the same time, we would like to maintain the efficiency of the RLS algorithm.

3.2.1 Using Forward–Backward RLS

One method to achieve this would be to derive a backward recursion and alternately use the forward and backward recursions as follows. Suppose the current window spans $[t_1, \ldots, t_w]$. When a new data point comes in at t_{w+1}, we use the forward recursion (equations 7–9) to derive a model for the points in $[t_1, \ldots, t_w, t_{w+1}]$. Then, we use the backward recursion to remove the effects of the point at t_1, and thus get a model for the points in $[t_2, \ldots, t_w, t_{w+1}]$.

In what follows, we derive the backward recursion for RLS. Suppose we have $\hat{\theta}_{old}$ and P_{old} over certain data S, where

$$P_{old} = \left\{ \sum_{k \in S} \varphi(k) \varphi^T(k) \right\}^{-1} \tag{10}$$

$$\hat{\theta}_{old} = P_{old} \sum_{k \in S} [\varphi(k) y(k)] \quad (11)$$

We want to find $\hat{\theta}_{new}$ and P_{new} that correspond to the same data with a given point s removed, i.e. $S - \{s\}$. By definition,

$$P_{new}^{-1} = P_{old}^{-1} - \varphi(s)\varphi^T(s) \quad (12)$$

Premultiplying the above equation with P_{new} and postmultiplying with P_{old} we get

$$P_{old} = P_{new} - P_{new}\varphi(s)\varphi^T(s)P_{old} \quad (13)$$

$$\Rightarrow \frac{P_{old}\varphi(s)\varphi^T(s)P_{old}}{[1 - \varphi^T(s)P_{old}\varphi(s)]} = P_{new}\varphi(s)\varphi^T(s)P_{old}, \quad (14)$$

which leads to

$$\hat{\theta}_{new} = \hat{\theta}_{old} - L_{new}[y(t) - \hat{\theta}_{old}^T \varphi(s)], \quad (15)$$

$$L_{new} = \frac{P_{old}\varphi(s)}{[1 - \varphi^T(s)P_{old}\varphi(s)]}, \quad (16)$$

$$P_{new} = P_{old} + \frac{P_{old}\varphi(s)\varphi^T(s)P_{old}}{[1 - \varphi^T(s)P_{old}\varphi(s)]}. \quad (17)$$

Equations 15–17 form the Backward RLS (BRLS) algorithm. However, BRLS has a serious drawback that has to do with numerical instability.

3.2.2 Using GEMM

Ganti et al [4] describe the Generic Model Maintenance Algorithm (GEMM) that takes any incremental model maintenance algorithm and transforms it into an algorithm that allows restrictions on the data span dimension.

The GEMM algorithm uses the incremental model maintenance algorithm (for the unrestricted window) to perform computations over moving window. It achieves this by maintaining a set of models as follows. Say the current window spans over time $[t_1 \ldots t_w]$. This window is going to overlap with the next $w - 1$ time units. The GEMM algorithm maintains w models at all times. One for the current window, and the remaining $w - 1$ for the future windows. These are incrementally maintained as time progresses.

We can employ GEMM together with the RLS algorithm to compute the parameters of a linear model of streaming data over a moving window. However, the disadvantage of this method is that it needs to maintain several models, which means that it requires more space.

3.3 Forgetting Factor Approach

In the case of a moving window, the model remembers (reflects) only data items within the current window span. The system either remembers a data point or has forgotten it (0/1). Instead, if the system can give different importance to the data points depending on their age or their reliability/importance, such a model could be a better representation of the data stream. The idea is to have a forgetting factor for the data points.

Equations 19– 21 describe an online algorithm for computing the parameters of a linear model for a timeseries using a forgetting factor $\lambda(t)$ and weights α_t for the data points.

Equation 18 illustrates the role played by α_t. $\lambda(t)$ is the forgetting factor.

$$\bar{R}(t) = \lambda(t)\bar{R}(t-1) + \alpha_t \varphi(t)\varphi^T(t), \quad (18)$$

$$\hat{\theta}(t) = \hat{\theta}(t-1) + L(t)[y(t) - \varphi^T(t)\hat{\theta}(t-1)], \quad (19)$$

$$L(t) = \frac{P(t-1)\varphi(t)}{[\lambda(t)/\alpha_t] + \varphi^T(t)P(t-1)\varphi(t)}, \quad (20)$$

$$P(t) = \frac{1}{\lambda(t)} \left\{ \alpha_t P(t-1) - L(t)\varphi^T(t)P(t-1) \right\}. \quad (21)$$

An example forgetting factor would be the exponential decay $\lambda(t) = \lambda_0$. Note that in this case, the half–life of a data point is $-1/\lg \lambda_0$.

4 Experimental Results

We report preliminary results of experiments with simulated as well as real timeseries, using the forgetting factor approach with $\lambda(t) = 0.95$ and $\alpha_t = 1$, for all t.

For all these experiments, we use the same set of values for the parameters of our outlier/change point detection tests , which are given in Table 2. Note that the parameters of our detection algorithm are inter–related and the most critical ones are: the windowSize, the outlierSensitivity, and the forgetting factor.

The results of the experiments are shown in Figure 4. [3]

Figures 4(a)–(d) are results from data that are synthetically generated. All the outliers and change points have been correctly identified (as can be seen visually). Figures 4(e) and (f) are experimental results over the real (traffic) datasets used in [5]. Evaluating the accuracy of an outlier/change detection algorithm in the case of synthetic data is straightforward, since outliers/change points are known. In the case of real data it is not as straightforward. We use the approach used by [5], i.e. to compare the results with visual change point detection.

[3] Due to space limitations, we few selected results are presented here.

The results indicate that in the synthetic datasets, the algorithm detected all the outliers and change points accurately (A change point within the $minChangeDistance$ of the actual change point is considered accurate). In the case of the real datasets, they are comparable to the results produced in [5].

Table 2. Parameters used for the experiments

windowSize	=	7
minDetectionWindow	=	2× *windowSize*
minErrorThreshold	=	0.001
maxErrorThreshold	=	5
overshootSensitivity	=	2.5
outlierSensitivity	=	2× *overshootSensitivity*
maxErrorOvershoots	=	*windowSize/1.2*
minChangeDistance	=	*windowSize*
changeOutlierWindow	=	*2 windowSize / 3*
changeOutlierCount	=	*windowSize/3-1*
changeResetWindow	=	*windowSize/3*
gainThreshold	=	1/3
forgetting factor $\lambda(t)$	=	0.95

5 Conclusion

In this paper we consider the problem of activity monitoring over streaming data. More specifically we consider the problem of detecting outliers and change points. It has several applications such as building alarms or alert systems when outliers or change points are detected. We discuss desiderata of such a system. Assuming that the data to be modeled by a linear model, we describe an adaptive incremental method for detecting outliers and change points in data streams. Our algorithm uses (a) intuitive criteria for labeling a data point as an outlier or as a change point, and (b) an adaptive incremental model estimation method. In this paper, we use a forgetting factor–based RLS algorithm for adaptive incremental model estimation (GEMM or BRLS could be some other alternatives). We present few experiment results using both simulated and real data, which show that our algorithms for change and outlier detection could accurately detect these events. Future work includes methods for optimally choosing the parameters of our algorithms, for accurate and timely detection of outliers/change points (in the probabilistic sense), and further experimental analysis with real datasets.

References

[1] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. OPTICS-OF: Identifying local outliers. In *Principles of Data Mining and Knowledge Discovery*, pages 262–270, 1999.

[2] U. C. Bureau. www.census.gov/population/www/estimates/st_stts.html.

[3] T. Fawcett and F. J. Provost. Activity monitoring: Noticing interesting changes in behavior. In *Knowledge Discovery and Data Mining*, pages 53–62, 1999.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan. DEMON: Mining and monitoring evolving data. In *ICDE*, pages 439–448, 2000.

[5] V. Guralnik and J. Srivastava. Event detection from time series data. In *Knowledge Discovery and Data Mining*, pages 33–42, 1999.

[6] S. B. Guthery. Partition regression. *Journal of American Statistical Association*, 69:945–947, 1994.

[7] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *IEEE International Conference on Data Mining, ICDM*, pages 289–296, 2001.

[8] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 392–403, 1998.

[9] L. Ljung. *System Identification: Theory for the User*. Prentice Hall, Upper Saddle River, NJ, 1999.

[10] L. Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification*. M.I.T. Press, Cambridge, MA, 1983.

[11] J. M. Pea, F. Famili, and S. Ltourneau. Data mining to detect abnormal behavior in aerospace data. In *KDD*, pages 390–397, 2000.

[12] N. Raghavan, R. M. Bell, and M. Schonlau. Defection detection. In *KDD*, pages 506–515, 2000.

[13] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., 1987.

[14] K. Yamanishi, J. ichi Takeuchi, G. J. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *Knowledge Discovery and Data Mining*, pages 320–324, 2000.

[15] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *ICDE*, pages 13–22, 2000.

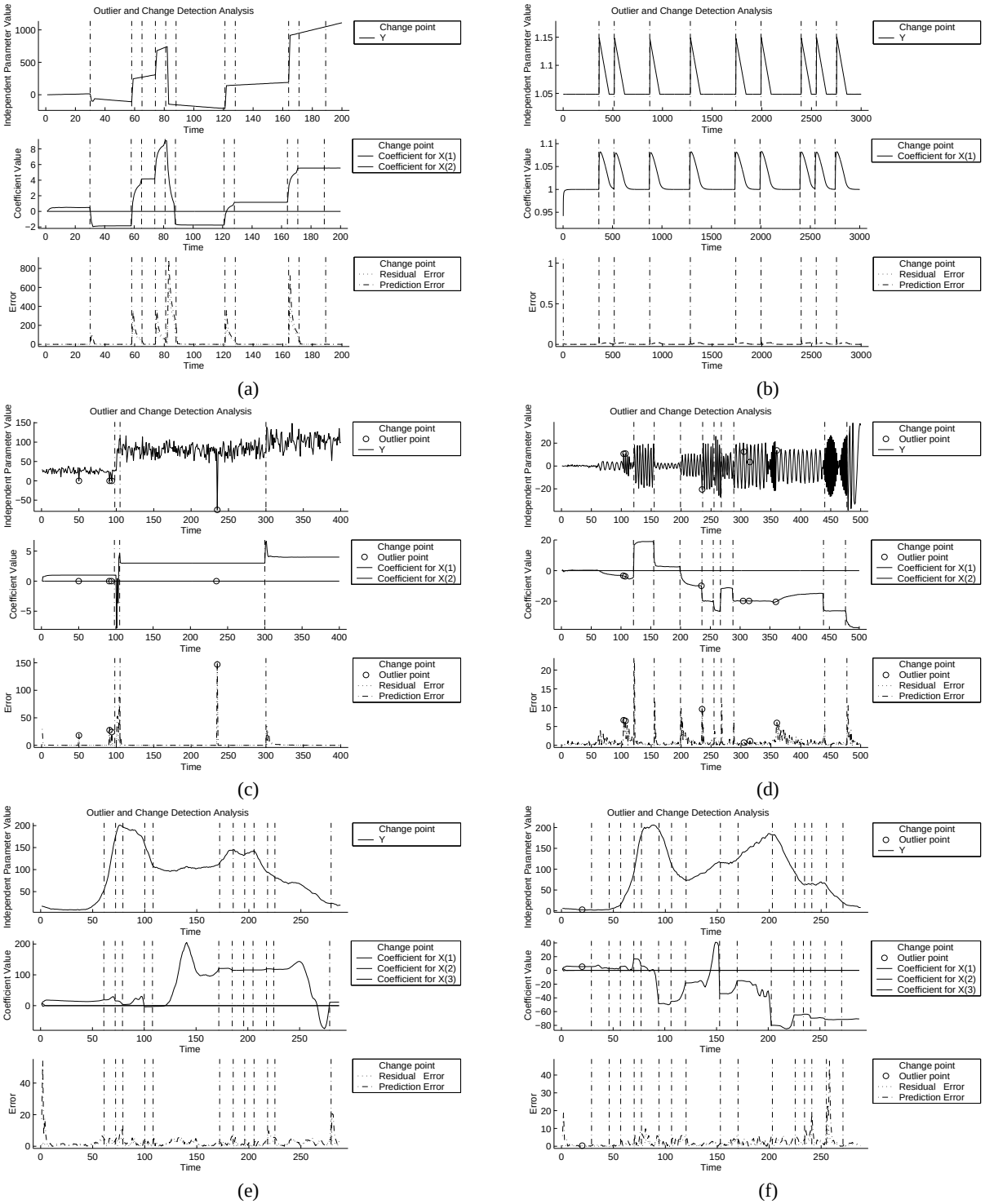

Figure 2. Results from RLS with forgetting factor