Applying the Q_n Estimator Online

Robin Nunkesser[1], Karen Schettlinger[2], and Roland Fried[2]

[1] Department of Computer Science, Univ. Dortmund, 44221 Dortmund
Robin.Nunkesser@udo.edu
[2] Department of Statistics, Univ. Dortmund, 44221 Dortmund
{schettlinger,fried}@statistik.uni-dortmund.de

Abstract. Reliable automatic methods are needed for statistical online monitoring of noisy time series. Application of a robust scale estimator allows to use adaptive thresholds for the detection of outliers and level shifts. We propose a fast update algorithm for the Q_n estimator and show by simulations that it leads to more powerful tests than other highly robust scale estimators.

1 Introduction

Reliable online analysis of high frequency time series is an important requirement for real-time decision support. For example, automatic alarm systems currently used in intensive care produce a high rate of false alarms due to measurement artifacts, patient movements, or transient fluctuations around the chosen alarm limit. Preprocessing the data by extracting the underlying level (the signal) and variability of the monitored physiological time series, such as heart rate or blood pressure can improve the false alarm rate. Additionally, it is necessary to detect relevant changes in the extracted signal since they might point at serious changes in the patient's condition.

The high number of artifacts observed in many time series requires the application of *robust* methods which are able to withstand some largely deviating values. However, many robust methods are computationally too demanding for real time application if efficient algorithms are not available.

Gather and Fried (2003) recommend Rousseeuw and Croux's (1993) Q_n estimator to measure the variability of the noise in robust signal extraction. The Q_n possesses a breakdown point of 50%, i.e. it can resist up to almost 50% large outliers without becoming extremely biased. Additionally, its Gaussian efficiency is 82% in large samples, which is much higher than that of other robust scale estimators: for example, the asymptotic efficiency of the median absolute deviation about the median (MAD) is only 36%. However, in an online application to moving time windows the MAD can be updated in $\mathcal{O}(\log n)$ time (Bernholt et al. (2006)), while the fastest algorithm known so far for the

Q_n needs $\mathcal{O}(n \log n)$ time (Croux and Rousseeuw (1992)), where n is the width of the time window.

In this paper, we construct an update algorithm for the Q_n estimator which, in practice, is substantially faster than the offline algorithm and implies an advantage for online application. The algorithm is easy to implement and can also be used to compute the Hodges-Lehmann location estimator (HL) online. Additionally, we show by simulation that the Q_n leads to resistant rules for shift detection which have higher power than rules using other highly robust scale estimators. This better power can be explained by the well-known high efficiency of the Q_n for estimation of the variability.

Section 2 presents the update algorithm for the Q_n. Section 3 describes a comparative study of rules for level shift detection which apply a robust scale estimator for fixing the thresholds. Section 4 draws some conclusions.

2 An update algorithm for the Q_n and the HL estimator

For data $x_1, \ldots, x_n$, $x_i \in \mathbb{R}$ and $k = \binom{\lfloor n/2 \rfloor + 1}{2}$, $\lfloor a \rfloor$ denoting the largest integer not larger than a, the Q_n scale estimator is defined as

$$\hat{\sigma}^{(Q)} = c_n^{(Q)} \left\{ |x_i - x_j|, 1 \leq i < j \leq n \right\}_{(k)} \ ,$$

corresponding to approximately the first quartile of all pairwise differences. Here, $c_n^{(Q)}$ denotes a finite sample correction factor for achieving unbiasedness for the estimation of the standard deviation σ at Gaussian samples of size n. For online analysis of a time series $x_1, \ldots, x_N$, we can apply the Q_n to a moving time window $x_{t-n+1}, \ldots, x_t$ of width $n < N$, always adding the incoming observation x_{t+1} and deleting the oldest observation x_{t-n+1} when moving the time window from t to $t+1$. Addition of x_{t+1} and deletion of x_{t-n+1} is called an *update* in the following.

It is possible to compute the Q_n as well as the HL estimator of n observations with an algorithm by Johnson and Mizoguchi (1978) in running time $\mathcal{O}(n \log n)$, which has been proved to be optimal for offline calculation. An optimal online update algorithm therefore needs at least $\mathcal{O}(\log n)$ time for insertion or deletion, respectively, since otherwise we could construct an algorithm faster than $\mathcal{O}(n \log n)$ for calculating the Q_n from scratch. The $\mathcal{O}(\log n)$ time bound was achieved for $k = 1$ by Bespamyatnikh (1998). For larger k - as needed for the computation of Q_n or the HL estimator - the problem gets more difficult and to our knowledge there is no online algorithm, yet. Following an idea of Smid (1991), we use a buffer of possible solutions to get an online algorithm for general k, because it is easy to implement and achieves a good running time in practice. Theoretically, the worst case amortized time per update may not be better than the offline algorithm, because $k = \mathcal{O}(n^2)$ in our case. However, we can show that our algorithm runs substantially faster for many data sets.

Lemma 1. *It is possible to compute the Q_n and the HL estimator by computing the kth order statistic in a multiset of form $X + Y = \{x_i + y_j \mid x_i \in X \text{ and } y_j \in Y\}$.*

Proof. For $X = \{x_1, \ldots, x_n\}$, $k' = \binom{\lfloor n/2 \rfloor + 1}{2}$, and $k = k' + n + \binom{n}{2}$ we may compute the Q_n in the following way:

$$c_n^{(Q)}\{|x_i - x_j|, 1 \le i < j \le n\}_{(k')} = c_n^{(Q)}\{x_{(i)} - x_{(n-j+1)}, 1 \le i, j \le n\}_{(k)} \ .$$

Therefore me may compute the Q_n by computing the kth order statistic in $X + (-X)$.

To compute the HL estimator $\hat{\mu} = \text{median}\,\{(x_i + x_j)/2, 1 \le i \le j \le n\}$, we only need to compute the median element in $X/2 + X/2$ following the convention that in multisets of form $X + X$ exactly one of $x_i + x_j$ and $x_j + x_i$ appears for each i and j. □

To compute the kth order statistic in a multiset of form $X + Y$, we use the algorithm of Johnson and Mizoguchi (1978). Due to Lemma 1, we only consider the online version of this algorithm in the following.

2.1 Online Algorithm

To understand the online algorithm it is helpful to look at some properties of the offline algorithm. It is convenient to visualize the algorithm working on a partially sorted matrix $B = (b_{ij})$ with $b_{ij} = x_{(i)} + y_{(j)}$, although B is, of course, never constructed. The algorithm utilizes, that $x_{(i)} + y_{(j)} \le x_{(i)} + y_{(\ell)}$ and $x_{(j)} + y_{(i)} \le x_{(\ell)} + y_{(i)}$ for $j \le \ell$. In consecutive steps, a matrix element is selected, regions in the matrix are determined to be certainly smaller or certainly greater than this element, and parts of the matrix are excluded from further consideration according to a case differentiation. As soon as less than n elements remain for consideration, they are sorted and the sought-after element is returned. The algorithm may easily be extended to compute a *buffer* $\mathcal{B}$ of size s of matrix elements $b_{(k-\lfloor (s-1)/2 \rfloor):n^2}, \ldots, b_{(k+\lfloor s/2 \rfloor):n^2}$.

To achieve a better computation time in online application, we use balanced trees, more precisely indexed AVL-trees, as the main data structure. Inserting, deleting, finding and determining the rank of an element needs $\mathcal{O}(\log n)$ time in this data structure. We additionally use two pointers for each element in a balanced tree. In detail, we store X, Y, and $\mathcal{B}$ in separate balanced trees and let the pointers of an element $b_{ij} = x_{(i)} + y_{(j)} \in \mathcal{B}$ point to $x_{(i)} \in X$ and $y_{(j)} \in Y$, respectively. The first and second pointer of an element $x_{(i)} \in X$ points to the smallest and greatest element such that $b_{ij} \in \mathcal{B}$ for $1 \le j \le n$. The pointers for an element $y_{(j)} \in Y$ are defined analogously.

Insertion and deletion of data points into the buffer $\mathcal{B}$ correspond to the insertion and deletion of matrix rows or columns in B. We only consider insertions into and deletions from X in the following, because they are similar to insertions into and deletions from Y.

Deletion of element x_{del}

1. Search in X for x_{del} and determine its rank i and the elements b_s and b_g pointed at.
2. Determine $y_{(j)}$ and $y_{(\ell)}$ with the help of the pointers such that $b_s = x_{(i)} + y_{(j)}$ and $b_g = x_{(i)} + y_{(\ell)}$.
3. Find all elements $b_m = x_{(i)} + y_{(m)} \in \mathcal{B}$ with $j \leq m \leq \ell$.
4. Delete these elements b_m from $\mathcal{B}$, delete x_{del} from X, and update the pointers accordingly.
5. Compute the new position of the kth element in $\mathcal{B}$.

Insertion of element x_{ins}

1. Determine the smallest element b_s and the greatest element b_g in $\mathcal{B}$.
2. Determine with a binary search the smallest j such that $x_{\text{ins}} + y_{(j)} \geq b_s$ and the greatest ℓ such that $x_{\text{ins}} + y_{(\ell)} \leq b_g$.
3. Compute all elements $b_m = x_{\text{ins}} + y_{(m)}$ with $j \leq m \leq \ell$.
4. Insert these elements b_m into $\mathcal{B}$, insert x_{ins} into X and update pointers to and from the inserted elements accordingly.
5. Compute the new position of the kth element in $\mathcal{B}$.

It is easy to see, that we need a maximum of $\mathcal{O}(|\text{deleted elements}| \log n)$ and $\mathcal{O}(|\text{inserted elements}| \log n)$ time for deletion and insertion, respectively. After deletion and insertion we determine the new position of the kth element in $\mathcal{B}$ and return the new solution or recompute $\mathcal{B}$ with the offline algorithm if the kth element is not in $\mathcal{B}$ any more. We may also introduce bounds on the size of $\mathcal{B}$ in order to maintain linear size and to recompute $\mathcal{B}$ if these bounds are violated.

For the running time we have to consider the number of elements in the buffer that depend on the inserted or deleted element and the amount the kth element may move in the buffer.

Theorem 1. *For a constant signal with stationary noise, the expected amortized time per update is* $\mathcal{O}(\log n)$.

Proof. In a constant signal with stationary noise, data points are *exchangeable* in the sense that the rank of each data point in the set of all data points is equiprobable. Assume w.l.o.g. that we only insert into and delete from X. Consider for each rank i of an element in X the number of buffer elements depending on it, i.e. $|\{i \mid b_{ij} \in \mathcal{B}\}|$. With $\mathcal{O}(n)$ elements in $\mathcal{B}$ and equiprobable ranks of the observations inserted into or deleted from X, the expected number of buffer elements depending on an observation is $\mathcal{O}(1)$. Thus, the expected number of buffer elements to delete or insert during an update step is also $\mathcal{O}(1)$ and the expected time we spend for the update is $\mathcal{O}(\log n)$.

To calculate the amortized running time, we have to consider the number of times $\mathcal{B}$ has to be recomputed. With equiprobable ranks, the expected amount the kth element moves in the buffer for a deletion and a subsequent insertion is 0. Thus, the expected time the buffer has to be recomputed is also 0 and consequently, the expected amortized time per update is $\mathcal{O}(\log n)$. □

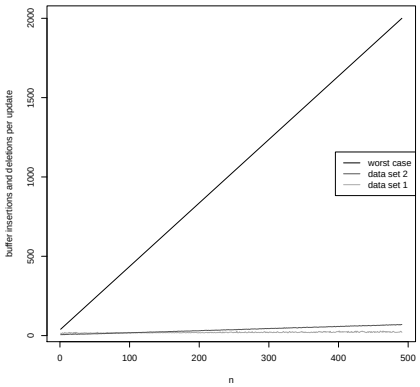

Fig. 1. Insertions and deletions needed for an update with growing window size n.

2.2 Running Time Simulations

To show the good performance of the algorithm in practice, we conducted some running time simulations for online computation of the Q_n. The first data set for the simulations suits the conditions of Theorem 1, i.e. it consists of a constant signal with standard normal noise and an additional 10% outliers of size 8. The second data set is the same in the first third of the time period, before an upward shift of size 8 and a linear upward trend in the second third and another downward shift of size 8 and a linear downward trend in the final third occur. The reason to look at this data set is to analyze situations with shifts, trends and trend changes, because these are not covered by Theorem 1.

We analyzed the average number of buffer insertions and deletions needed for an update when performing $3n$ updates of windows of size n with $10 \leq n \leq 500$. Recall, that the insertions and deletions directly determine the running time. A variable number of updates assures similar conditions for all window widths. Additionally, we analyzed the position of $\mathcal{B}$ over time visualized in the matrix B when performing 3000 updates with a window of size 1000.

We see in Figure 1 that the number of buffer insertions and deletions for the first data set seems to be constant as expected, apart from a slight increase caused by the 10% outliers. The second data set causes a stronger increase, but is still far from the theoretical worst case of $4n$ insertions and deletions.

Considering Figure 2 we gain some insight into the observed number of update steps. For the first data set, elements of $\mathcal{B}$ are restricted to a small region in the matrix B. This region is recovered for the first third of the second data set in the right-hand side figure. The trends in the second data set cause $\mathcal{B}$ to be in an additional, even more concentrated diagonal region, which is even better for the algorithm. The cause for the increased running time is the time it takes to adapt to trend changes. After a trend change there is a short period, in which parts of $\mathcal{B}$ are situated in a wider region of the matrix B.

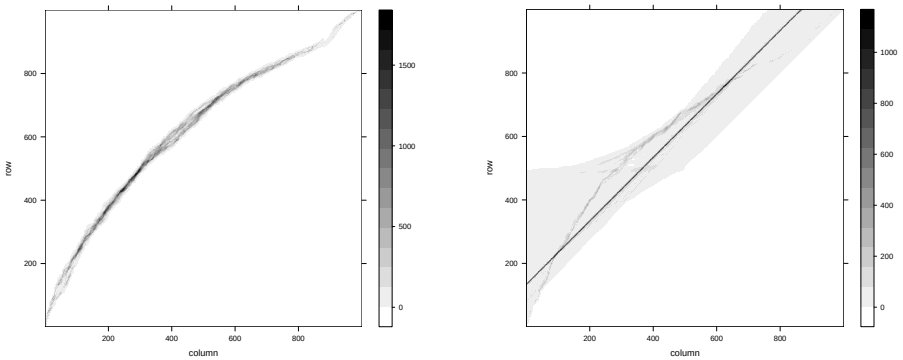

Fig. 2. Positions of $\mathcal{B}$ in the matrix B for data set 1 (left) and 2 (right).

3 Comparative Study

An important task in signal extraction is the fast and reliable detection of abrupt level shifts. Comparison of two medians calculated from different windows has been suggested for the detection of such edges in images (Bovik and Munson (1986), Hwang and Haddad (1994)). This approach has been found to give good results also in signal processing (Fried (2007)). Similar as for the two-sample t-test, an estimate of the noise variance is needed for standardization. Robust scale estimators like the Q_n can be applied for this task. Assuming that the noise variance can vary over time but is locally constant within each window, we calculate both the median and the Q_n separately from two time windows $y_{t-h+1}, \ldots, y_t$ and $y_{t+1}, \ldots, y_{t+k}$ for the detection of a level shift between times t and $t+1$. Let $\tilde{\mu}_{t-}$ and $\tilde{\mu}_{t+}$ be the medians from the two time windows, and $\hat{\sigma}_{t-}$ and $\hat{\sigma}_{t+}$ be the scale estimate for the left and the right window of possibly different widths h and k. An asymptotically standard normal test statistic in case of a (locally) constant signal and Gaussian noise with a constant variance is

$$\frac{\tilde{\mu}_{t+} - \tilde{\mu}_{t-}}{\sqrt{0.5\pi(\hat{\sigma}_{t-}^2/h + \hat{\sigma}_{t+}^2/k)}}$$

Critical values for small sample sizes can be derived by simulation.

Figure 3 compares the efficiencies of the Q_n, the median absolute deviation about the median (MAD) and the interquartile range (IQR) measured as the percentage variance of the empirical standard deviation as a function of the sample size n, derived from 200000 simulation runs for each n. Obviously, the Q_n is much more efficient than the other, 'classical' robust scale estimators.

The higher efficiency of the Q_n is an intuitive explanation for median comparisons standardized by the Q_n having higher power than those standardized by the MAD or the IQR if the windows are not very short. The power functions depicted in Figure 3 for the case $h = k = 15$ have been derived from

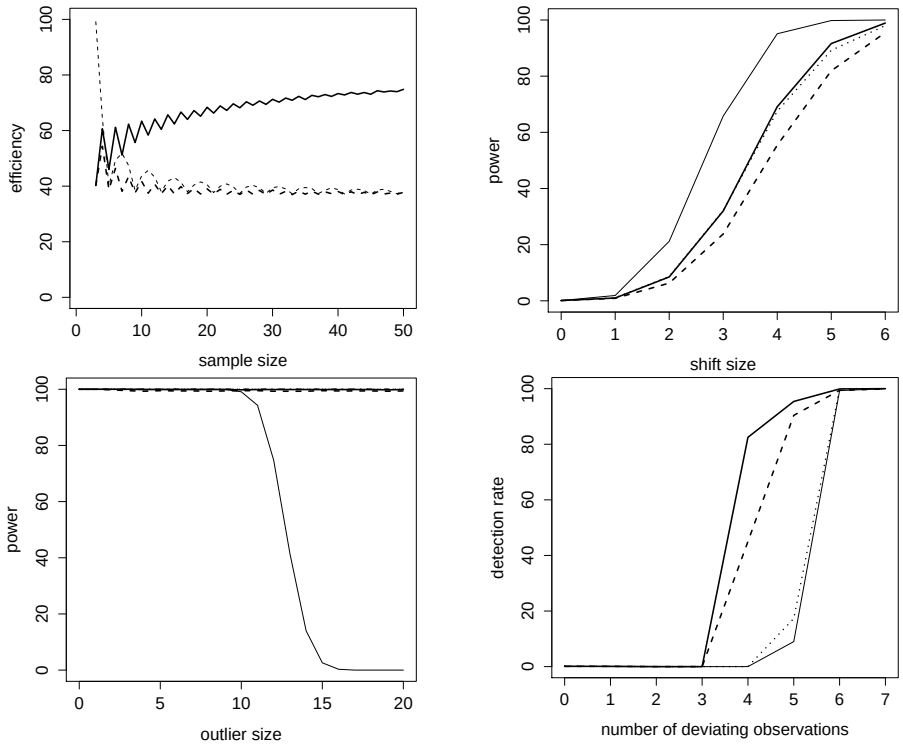

Fig. 3. Gaussian efficiencies (top left), power of shift detection (top right), power for a 10σ-shift in case of an outlier of increasing size (bottom left), and detection rate in case of an increasing number of deviating observations (bottom right): Q_n (solid), MAD (dashed), IQR (dotted), and S_n (dashed-dot). The two-sample t-test (thin solid) is included for the reason of comparison.

shifts of several heights $\delta = 0, 1, \ldots, 6$ overlaid by standard Gaussian noise, using 10000 simulation runs each. The two-sample t-test, which is included for the reason of comparison, offers under Gaussian assumptions higher power than all the median comparisons, of course. However, Figure 3 shows that its power can drop down to zero because of a single outlier, even if the shift is huge. To see this, a shift of fixed size 10σ was generated, and a single outlier of increasing size into the opposite direction of the shift inserted briefly after the shift. The median comparisons are not affected by a single outlier even if windows as short as $h = k = 7$ are used.

As a final exercise, we treat shift detection in case of an increasing number of deviating observations in the right-hand window. Since a few outliers should neither mask a shift nor cause false detection when the signal is constant, we would like a test to resist the deviating observations until more than half of the observations are shifted, and to detect a shift from then on. Figure 3 shows the detection rates calculated as the percentage of cases in which a shift was

detected for $h = k = 7$. Median comparisons with the Q_n behave as desired, while a few outliers can mask a shift when using the IQR for standardization, similar as for the t-test. This can be explained by the IQR having a smaller breakdown point than the Q_n and the MAD.

4 Conclusions

The proposed new update algorithm for calculation of the Q_n scale estimator or the Hodges-Lehmann location estimator in a moving time window shows good running time behavior in different data situations. The real time application of these estimators, which are both robust and quite efficient, is thus rendered possible. This is interesting for practice since the comparative studies reported here show that the good efficiency of the Q_n for instance improves edge detection as compared to other robust estimators.

Acknowledgements

The financial support of the Deutsche Forschungsgemeinschaft (SFB 475, "Reduction of complexity in multivariate data structures") is gratefully acknowledged.

References

BERNHOLT, T., FRIED, R., GATHER, U., WEGENER, I. (2006): Modified Repeated Median Filters. *Statistics and Computing, 16, 177–192.*

BESPAMYATNIKH, S. N. (1998): An Optimal Algorithm for Closest-Pair Maintenance. *Discrete and Computational Geometry, 19 (2), 175–195.*

BOVIK, A. C., MUNSON, D. C. Jr. (1986): Edge Detection using Median Comparisons. *Computer Vision, Graphics, and Image Processing, 33, 377–389.*

CROUX, C., ROUSSEEUW, P. J. (1992): Time-Efficient Algorithms for Two Highly Robust Estimators of Scale. *Computational Statistics, 1, 411–428.*

FRIED, R. (2007): On the Robust Detection of Edges in Time Series Filtering. *Computational Statistics & Data Analysis, to appear.*

GATHER, U., FRIED, R. (2003): Robust Estimation of Scale for Local Linear Temporal Trends. *Tatra Mountains Mathematical Publications, 26, 87–101.*

HWANG, H., HADDAD, R. A. (1994): Multilevel Nonlinear Filters for Edge Detection and Noise Suppression. *IEEE Trans. Signal Processing, 42, 249–258.*

JOHNSON, D. B., MIZOGUCHI, T. (1978): Selecting the kth Element in $X + Y$ and $X_1 + X_2 + ... + X_m$. *SIAM Journal on Computing, 7 (2), 147–153.*

ROUSSEEUW, P.J. and CROUX, C. (1993): Alternatives to the Median Absolute Deviation. *Journal of the American Statistical Association, 88, 1273–1283.*

SMID, M. (1991): Maintaining the Minimal Distance of a Point Set in Less than Linear Time. *Algorithms Review, 2, 33–44.*