Computing the Update of the Repeated Median Regression Line in Linear Time

Thorsten Bernholt[a] and Roland Fried[b*]

[a]Lehrstuhl Informatik 2, Universität Dortmund, Germany
bernholt@ls2.cs.uni-dortmund.de

[b]Fachbereich Statistik, Universität Dortmund, Germany
fried@statistik.uni-dortmund.de

September 18, 2002

Abstract

The repeated median line estimator is a highly robust method for fitting a regression line to a set of n data points in the plane. In this paper, we consider the problem of updating the estimate after a point is removed from or added to the data set. This problem occurs e.g. in statistical online monitoring, where the computational effort is often critical. We present a deterministic algorithm for the update working in $O(n)$ time and $O(n^2)$ space.

Keywords: Robust regression, time series analysis, robust filtering, repeated median, computational geometry, efficient algorithms

1 Introduction

A fundamental problem in modern data analysis is robust fitting of a straight line to a sample of data points in the plane. Robustness is essential in applications where data are routinely collected and time-consuming screening of the data is not possible prior to the data analysis. In the computer age such applications are encountered rather as a rule than as an exception. We are at risk of

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

drawing wrong conclusions when using non-robust methods which do not provide protection against spurious data ("outliers") caused by e.g. measurement artifacts. For example, it is well known that the classical least squares estimator is not robust at all. Moving a single data point far out of the data cloud may change the least squares estimate completely [9]. In order to cope with such outlying points robust approaches have been proposed for line fitting, e.g. the Theil-Sen estimator [7], the least median of squares estimator [8], deepest regression [1],and the repeated median estimator [10]. A common measure of the robustness of an estimator is its finite sample replacement breakdown point. This breakdown point is the minimal fraction of data points that may carry the estimate 'beyond all bounds' when it is replaced by arbitrary values. The repeated median was the first regression estimator to attain a breakdown point of 50 % asymptotically, i.e. for a large sample size, which is the optimum for a regression equivariant estimator [9]:

Definition 1.1 (Repeated Median) *Given n points $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{R}^2$, $x_i \neq x_j$, denote the slope of the line through (x_i, y_i) and (x_j, y_j) by $a_{ij} = \frac{y_i - y_j}{x_i - x_j}$. The repeated median estimator (β_{RM}, μ_{RM}) is defined by*

$$\begin{aligned} \beta_{RM} &= \operatorname*{med}_{i=1\dots n} \quad \operatorname*{med}_{j=1\dots n, j\neq i} (a_{ij}) \, , \\ \mu_{RM} &= \operatorname*{med}_{i=1\dots n} \quad (y_i - \beta_{RM}\, x_i) \, . \end{aligned}$$

Here, we define the median $\operatorname{med} C$ of a set $C = \{c_1, \dots, c_n\}$ as the element with rank $\lfloor n/2 \rfloor$ in the sorted order of C.

Since it is possible to calculate the median in linear time, the repeated median estimator can be computed in $O(n^2)$ time. Stein and Werman [11] present a sophisticated deterministic algorithm running in $O(n \log^2 n)$ time. A randomised algorithm is given by Matoušek, Mount and Netanyahu [7] with an expected running time of $O(n \log n)$.

The repeated median has been used recently for online signal extraction [4]. For robust approximation of an underlying signal from a time series, the data points are processed by moving a window along the time axis, which contains exactly n subsequent observations, and calculating the repeated median for each window. In other words, starting from a set of n points a sequence of update steps is performed. In each step, one point is deleted at the start of the window and one point is inserted at the end of the window before calculating the repeated median for the modified data set. In this way a smooth, locally almost linear signal can be extracted from the data. The repeated median shows very satisfactory performance in this setting as it guarantees both protection against a large number of outliers (measured by the breakdown point and bias curves) and moderate variability in an outlier free data set (measured by the variance).

For online processing of high frequency data the computation time needed is critical. Although a straightforward implementation of the repeated median may be sufficient for processing time series which are sampled every minute, a faster algorithm is called for when the variables are observed much faster. In intensive care for instance, medical devices measure physiological variables at least once a second. The question is whether we can reduce the computation time and benefit from prior calculations since the problem of computing the repeated median becomes an update problem here. In this paper, we present an algorithm which performs an update step in $O(n)$ time and $O(n^2)$ space. In Section 2, the main idea of the algorithm is described. The details of the subroutines are given in Section 3. A special case is treated in Section 4.

2 The Algorithm

In the following $(x_1, y_1), \ldots, (x_n, y_n)$ denotes a sample of data points in the plane. According to the point-line duality, we map the point (x_i, y_i) to the dual line l_i defined by $v = x_i u + y_i$. If we use the term "slope" in the following, we will always refer to lines in the dual space. As we process data points from time series, the x-coordinate measures time. Hence all x_i are distinct and the sequence $x_1, \ldots, x_n$ is increasing and thus there are no vertical lines and no two lines have the same slope. In the dual space, we say that the point (u, v) is located *left of* the line l if there exists a constant $c > 0$ such that the point $(u + c, v)$ is located on the line l. The terms *right of*, *above* and *below* are defined in a similar way.

Let (u_{ij}, v_{ij}) be the intersection point of the lines l_i and l_j. Since the equation $a_{ij} = -u_{ij}$ holds true, we have to find the median $m_i = \text{med}_{j=1..n, j \neq i}\,(-u_{ij})$ on each line l_i and the global median $\beta_{\text{RM}} = \text{med}_{i=1..n}\,(m_i)$. In Section 2 and 3 we assume that at most two lines are intersecting in one point. The other case is handled in Section 4.

In the online scenario, one line l_i is deleted and another line l_k is inserted into the arrangement. Considering the line l_i, one intersection point (u_{ij}, v_{ij}) is deleted and one intersection point (u_{ik}, v_{ik}) is inserted. What happens to the median m_i on l_i? If the inserted and deleted points are located on different sides of the current median, the new median is one of the two intersection points on the line l_i in the neighborhood of m_i. If both points are located on the same side of the median m_i, the median does not change.

To compute the n new medians, we need a special data structure, a hammock graph [2], to represent the arrangement of lines. An arrangement consists of vertices, line segments and faces. The hammock graph allows the algorithm to walk around a face in clockwise order and to walk along a line visiting each line segment in increasing or decreasing order. Each step needs $O(1)$ time, and $O(n)$ steps are sufficient to insert or delete a line. The details are described in the next section.

After computing all n new medians, it is easy to determine β_{RM} and μ_{RM} in

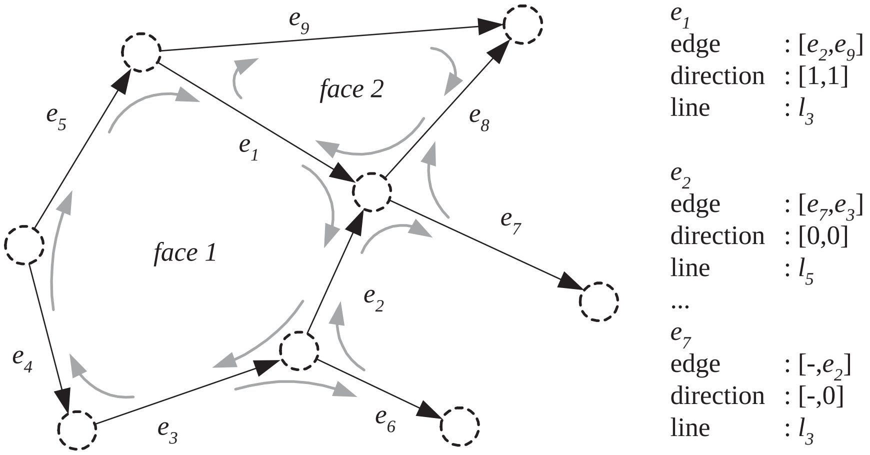


Figure 1: The edges of a hammock graph are connected facewise. The pointers are displayed as grey arrows. The implicit vertices are drawn as dashed circles.

linear time using algorithms from [6]. However, for a practical implementation, it is more convenient to make use of the Quickselect algorithm running in expected linear time, presented in [3]. The following theorem summarises the results of this paper:

Theorem 2.1 *An update of the repeated median estimator can be computed in linear time.*

3 The Hammock Graph

In this section we describe the hammock graph which can be used to store an arrangement of n lines, illustrated in Figure 1. It is organised as a doubly connected edge list [5]. As we need a left and right boundary of the arrangement, we add two vertical lines, line L located in the negative infimum and line R located in the positive infimum. In this section we consider *simple* arrangements, i. e. at most two lines intersect in one point, and in the primal problem no two values a_{ij} are equal. The other case of a degenerate arrangement is discussed in Section 4. Besides there is no need to consider the problem of two parallel lines since we process data points from time series and the slopes of the lines are strictly increasing.

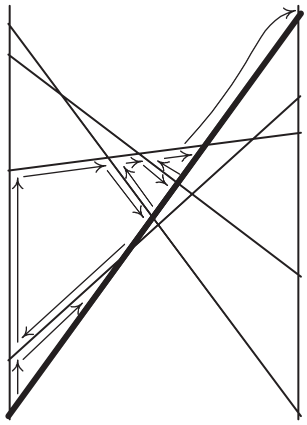
The lines of the arrangement divide the space into faces so that each line segment is adjacent to exactly two faces. A line segment is represented by a directed edge in the data structure and the description of the edge e_i consists of five entries. The u-coordinates of the incident intersection points are stored in the entries u_{Left} and u_{Right}. The next edges walking around the two adjacent faces in clockwise direction are stored in the entry "edge", containing the edge e_j incident to the right intersection point of e_i, and the edge incident to the left intersection point. Each edge has a designated direction, the arrowhead always points to the right hand side. If the next edge e_j points to the reverse direction as e_i with respect to the walk around the face, then the first entry in "direction" is "1", otherwise "0". The second entry behaves similarly. We say that a line l *supports* an edge e if this edge e represents a line segment of l. We denote this by $l(e)$. The line supporting the edge is stored in the entry "line".

In contrast to usual graphs, no vertices are stored in the hammock graph. To simplify the description, we will use the term *implicit vertex*. In the figures, each implicit vertex is displayed as a dashed circle.

This hammock graph allows two basic operations, the walk around faces and the walk along the edges of a line. To determine the next edge in both walks, $O(1)$ time is sufficient since we assume that no more than two lines intersect in one point. In the next three subsections, the operations relevant for an update are described.

3.1 Inserting a line

The empty hammock graph consists of two edges supported by L and R. The first n lines are inserted consecutively to construct the initial hammock graph. After inserting n lines, each line supports $n+2$ edges, n edges between L and R, one edge left of L and one edge right of R. These two additional edges are necessary for deleting a line.



As we process time series data, the newest line l_k in the kth step of the construction of the initial graph has a larger slope than any previous one and it will intersect L below all other intersections. Denote this intersected edge on L by e_1. In order to find the intersections of l_k with the lines $l_1, \ldots, l_{k-1}$, we start at e_1 and walk around the adjacent face as displayed in the Figure aside this paragraph. For each edge g_i we determine the line $l(g_i)$ and calculate the intersection point (u, v) and compare it with the u-coordinates stored in g_i. If $u_{\text{Left}} \leq u \leq u_{\text{Right}}$, the next edge e_2 intersected by l_k is found. We continue walking around the second adjacent face from e_2. In this way we find all edges $e_1, \ldots, e_{k+1}$ intersected by l_k including the edge e_{k+1} supported by R, which is located above all other edges of R. Note that we do not have to walk around the last face since we can determine e_{k+1} directly after reaching the kth edge.

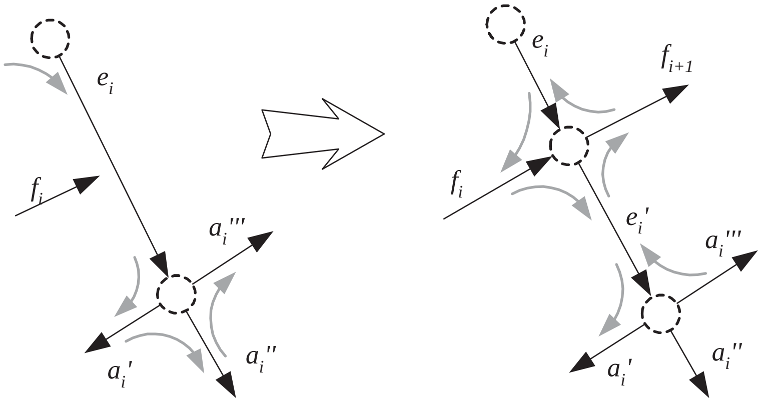


Figure 2: The dissection of a line needed by the insertion operation.

Now we have to insert the new edges $f_1, \ldots, f_{k+2}$ supported by l_k. Starting with f_i, we have to dissect e_i into e_i and e_i' and connect them with f_i and f_{i+1} according to Figure 2. Calculate the u-coordinate of the new intersection point and store it in the corresponding entry. Note that e_i can be orientated in both directions and it is important that the new edge e_i' has the same direction as e_i to ensure $u_{\text{Left}} \leq u_{\text{Right}}$. The entry "line" has to be updated as well. The following lemma is taken from [2] and it also follows from the Zone Theorem [5]:

Lemma 3.1 *A line can be inserted into a hammock graph consisting of $n-1$ lines in time $O(n)$.*

3.2 Deleting a line

In the first part of an update step, the oldest line denoted by l_1 must be deleted. As the oldest line is intersecting L above all others, the left-most edge f_1 supported by line l_1 can be found easily. Following the six pointers in Figure 3 on the following page, we can determine the edges e_i', f_{i+1}, e_i, a_i', a_i'' and a_i'''. Note that the edges a_i', a_i'' and a_i''' exist (for $n \geq 2$) since we added additional edges beyond the lines L and R. We have to delete e_i' and f_i and connect e_i with a_i' and a_i'''. The entry u_{Right} of e_i also has to be corrected. We proceed with the edge f_{i+1} until the edge f_{n+2} is found. Hence, we have shown the following:

Corollary 3.2 *A line can be deleted from a hammock graph consisting of n lines in time $O(n)$.*

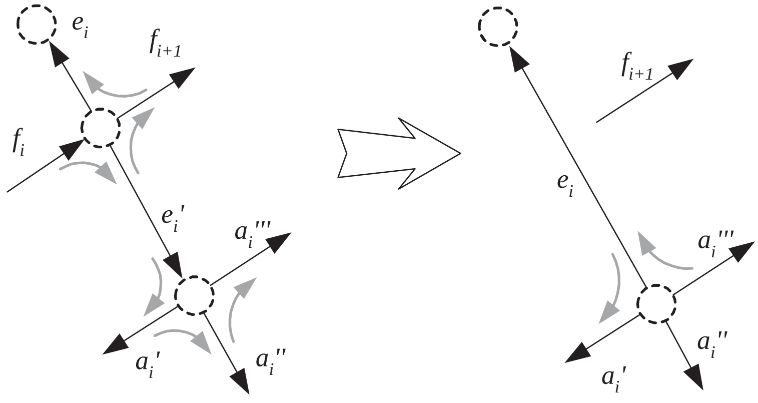


Figure 3: The basic operation needed to delete a line resp. the supported edges f_i.

3.3 Updating the medians

We have to determine the median intersection point on each line. As there are no vertices in the hammock graph, we store for each line l_i a median edge w_i. By convention, let the entry u_{Right} be the current median m_i of this line.

If the line l^* is inserted or deleted, we have to determine whether we have to move the median m_i and, if so, into which direction. To this end, we count the intersection points on l_i located left of the current median. Considering the total number of intersection points of l_i, the new median can be found easily. To adjust the count, we determine the position of the intersection of l_i and l^*, denoted by v^*, in the following way: Denote the intersection point representing the median m_i by v_i. Let l_q be the line intersecting l_i in the point v_i. We have to distinguish four cases, which are shown in Table 1 on the next page. As the cases can be handled similarly, we just consider one case. In the upper left case of Table 1 on the following page, the slope of l_q is larger than the slope of l_i and the intersection point c of the lines l_q and l^* is located below l_i, as displayed in Figure 4 on the next page. As the inserted line has the largest slope and the deleted line the smallest one, it follows that v^* is located left of v_i. Instead of determining c directly, we additionally mark visited lines in the insertion operation. If l_q is marked then c is below l_i, respectively above in the deletion operation. This allows us to count the intersection points on l_i located left of the current median v_i.

It may happen that the median edge w_i is involved in the insertion or deletion operation. More precisely, w_i may be one of the edges e_i or e_i'. In this case, we temporarily take one of the neighbors as w_i and determine the correct median edge after handling that line. Without considering the costs of the insertion and deletion operation, only $O(1)$ time is needed to find the new median edge since only a constant number of edges has to be examined. Therefore, we

	c below l_i	c above l_i
slope(l_q) $>$ slope(l_i)	left	right
slope(l_q) $<$ slope(l_i)	right	left

Table 1: The entries give the position of the intersection point v^* of the deleted/inserted line with respect to the median-edge w_i.

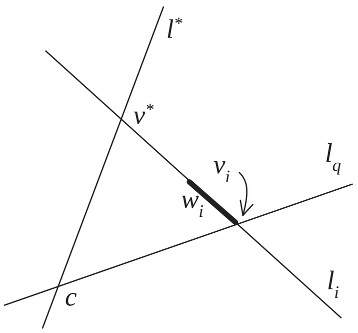


Figure 4: The upper left case of Table 1 is displayed.

obtain the following corollary:

Corollary 3.3 *Given a hammock graph consisting of n lines, the medians $m_1, \ldots, m_n$ can be updated in time $O(n)$, after one line was deleted or inserted.*

4 Degenerate Arrangements

Updating the repeated median is of potential interest in every application with data being collected sequentially. In case of a random design, when regressing one dependent variable on another random variable, three or more points (x_i, y_i) can be located on the same line. Although the probability of this occurring will often be small we nevertheless need to ensure that also in this degenerate case the repeated median is computed correctly. The problem is that in such situations more than two lines are intersecting in one point. Such a multiple intersection point is not stored directly in the hammock graph. Instead of this, for each pair of two lines a separate implicit vertex is stored. This leads to edges with $u_{\text{Left}} = u_{\text{Right}}$. We call a subgraph of the hammock an *agglomeration* if the coordinates of all edges have the same value. An edge e is called a *border edge* if exactly one implicit vertex belongs to an agglomeration or both implicit vertices belong to different agglomerations.

Since in the deletion and in the median update operation no coordinates are involved, these operations are working correctly in this special situation. In the following, we analyze how an agglomeration is handled by the insertion operation.

Lemma 4.1 *The algorithm described in Section 3.1 inserts a line correctly even if the hammock graph contains one or more agglomerations.*

Proof. We will show that the line l_n defined by $v = x_n u + y_n$, which intersects one or more agglomerations, is inserted in the same way as the line l_n^ϵ defined by $v = x_n u + y_n + \epsilon$ for sufficiently small $\epsilon > 0$. For that purpose, we divide the faces into three groups. A face is called *interior* if all edges of the face belong to the same agglomeration. If only some edges belong to an agglomeration, then the face is called *border* face. The other faces are called *usual.*

The new line is inserted from left to right. It is obvious that usual faces are handled in the same way. Take an arbitrary agglomeration A in which k lines intersect and thus $2k$ border edges belong to A. As l_n^ϵ is located above A, it intersects exactly k border edges of A, which are located above l_n. Consider the first border face f belonging to A that is found and entered on an edge that is not a border edge of A. Recall the condition used in the insertion operation to detect if a point (u, v) is located on an edge: $u_{\text{Left}} \leq u \leq u_{\text{Right}}$. The insertion operation walks around the face f in clockwise order and thus the first border edge left of l_n and adjacent to A is found and dissected. The following $k - 1$ border faces exactly contain two border edges that belong to A and are also found by the insertion operation. There might be an agglomeration B located above l_n that has edges with A in common. Since the coordinates of A and B are different, B does not interfere the insertion of l_n. The $(k + 1)$st border face is left on a non-border edge or on a border edge that belongs to a distinct agglomeration. Thus we have shown that the same edges in the hammock graph are dissected, no matter if we insert l_n or l_n^ϵ. □

5 Conclusions

In this paper, we have described an algorithm which computes an update of the repeated median in linear time. This operation is useful in the context of time series filtering. We have considered the correctness of the algorithm even in the case of degenerate inputs, which is important in high risk environments like intensive care online monitoring [4]. The algorithm was implemented and compared with the naive approach. In general, our algorithm is faster when using windows with at least 10 data points, which can be considered as a minimal length for getting useful estimates [4]. Empirical results show that the computing time is reduced by a factor of 15 for time intervals with 500 points, a factor of 30 for 1000 points and a factor of 50 for 2000 points. This time saving can be crucial in applications where many variables which are sampled in short time intervals need to be processed online at the same time [**?**].

Acknowledgement The authors would like to thank Detlef Sieling, Ingo Wegener and Carsten Witt for helpful comments on the presentation.

References

[1] S. V. Aelst, P. J. Rousseeuw, M. Hubert, and A. Struyf. The deepest regression method. *Journal of Multivariate Analysis*, 81(1):138–166, 2002.

[2] B. Chazelle. Reporting and counting segment intersections. *Journal of Computer and Systems Science*, 32:156–182, 1986.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[4] P. L. Davies, R. Fried, and U. Gather. Robust signal extraction for online monitoring data. *Technical Report 02/2002, SFB 475, University of Dortmund*, 2002.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, 2000.

[6] D. Dor and U. Zwick. Selecting the median. *SIAM Journal on Computing*, 28(5):1722–1758, 1999.

[7] J. Matoušek, D. M. Mount, and N. Netanyahu. Efficient randomized algorithms for the repeated median line estimator. *Algorithmica*, 20(2):136–150, 1998.

[8] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79:871–880, 1984.

[9] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley & Sons, 1987.

[10] A. F. Siegel. Robust regression using repeated medians. *Biometrika*, 69(1):242–244, 1982.

[11] A. Stein and M. Werman. Finding the repeated median regression line. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, pages 409–413, 1992.