# A Practical Approximation Algorithm for the LTS Estimator

David M. Mount* Nathan S. Netanyahu† Christine D. Piatko‡ Ruth Silverman§
Angela Y. Wu¶

Draft: July 12, 2013

Dedicated to the memory of our dear friend and longtime colleague, Ruth Silverman

**Abstract**

The linear least trimmed squares (LTS) estimator is a statistical technique for fitting a linear model to a set of points. It was proposed by Rousseeuw as a robust alternative to the classical least squares estimator. Given a set of $n$ points in $\mathbb{R}^d$, the objective is to minimize the sum of the smallest 50% squared residuals (or more generally any given fraction). There exist practical heuristics for computing the linear LTS estimator, but they provide no guarantees on the accuracy of final result. There also exist lower-bound results that suggest that in the worst case, solving LTS (exactly or approximately) requires $\Omega(n^{d-1})$ time.

We present two results in an effort to reconcile this disparity. First, we introduce measure of the condition of a point set. We analyze a simple randomized algorithm for LTS and show that the probability that it succeeds in finding an approximately optimal solution can be related to the input's condition value. Second, we present an approximation algorithm for LTS. We prove that on termination, this algorithm guarantees finding a fit of a given desired accuracy. We also present empirical evidence that for well conditioned point sets, this algorithm converges rapidly to an accurate result.

**Keywords:** Robust estimation, linear estimation, least trimmed squares estimator, approximation algorithms, computational geometry

*Department of Computer Science, University of Maryland, College Park, Maryland. Partially supported by NSF grant CCF-1117259 and ONR grant N00014-08-1-1015. Email: mount@cs.umd.edu.

†Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel and Center for Automation Research, University of Maryland, College Park, Maryland. Email: nathan@{cs.biu.ac.il; cfar.umd.edu}.

‡The Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland, christine.piatko@jhuapl.edu.

§Center for Automation Research, University of Maryland, College Park, Maryland.

¶Department of Computer Science, American University, Washington, DC. Email: awu@american.edu.

# 1 Introduction

In standard *linear regression* (with intercept), an $n$-element point set $P = \{p_1, \ldots, p_n\}$ is given, where each point consists of some number of independent variables and one dependent variable. Letting $d$ denote the total number of variables, we wish to express the dependent variable as a linear function of $d-1$ independent variables. More formally, for $1 \leq i \leq n$, let $p_i = (x_{i,1}, \ldots, x_{i,d-1}, y_i) \in \mathbb{R}^d$. The objective is to compute a $(d-1)$-dimensional hyperplane, represented as a coefficient vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d) \in \mathbb{R}^d$ so that

$$y_i \;=\; \sum_{j=1}^{d-1} \beta_j x_{i,j} + \beta_d + e_i, \quad \text{for } i = 1, 2, \ldots, n,$$

where the $e_i$'s are the errors. We refer to the coefficients $\{\beta_1, \ldots, \beta_{d-1}\}$ as the hyperplane's *slopes* and $\beta_d$ is its $y$-intercept. Given such a vector $\boldsymbol{\beta}$, the *ith residual*, denoted by $r_i(\boldsymbol{\beta}, P)$, is defined to be $y_i - \left(\sum_{j=1}^{d-1} \beta_j x_{i,j} + \beta_d\right)$. Let $r_{[i]}(\boldsymbol{\beta}, P)$ denote the $i$th smallest residual in terms of absolute value. Throughout, we consider the $y$-coordinate axis to be the *vertical* direction, and so the $i$th residual is just the signed vertical distance from the hyperplane to $p_i$.

Robust estimators (see, e.g., [20]) have been introduced in order to eliminate sensitivity to outliers, that is, points that fail to follow the linear pattern of the majority of the points. The basic measure of the robustness of an estimator is its *breakdown point*, that is, the fraction (up to 50%) of outlying data points that can corrupt the estimator arbitrarily. The most widely studied robust linear estimator is Rousseeuw's *least median of squares estimator* (LMS) [18], which is defined to be the hyperplane that minimizes the median squared residual. (In general, an integer trimming parameter $h$ is given, and the objective is to find the hyperplane that minimizes the $h$th smallest squared residual.) A number of papers, both practical and theoretical, have been devoted to solving this problem in the plane and in higher dimensions [1, 3, 4, 14, 16, 17, 24].

It has been observed by Rousseuw and Leroy [20] that LMS may not be the best estimator from the perspective of statistical properties. They argue in support of the *least trimmed squares* (or LTS) linear estimator [18]. Given an $n$-element point set $P$ and a positive integer *trimming parameter* $h \leq n$, this estimator is defined to be the nonvertical hyperplane that minimizes the *sum* (as opposed to the maximum) of the $h$ smallest squared residuals. More formally, given a trimming parameter $h$, the *LTS cost* of a hyperplane $\boldsymbol{\beta}$ is defined to be

$$\Delta_{\boldsymbol{\beta}}^{(\text{LTS})}(P, h) \;=\; \sum_{i=1}^{h} r_{[i]}^2(\boldsymbol{\beta}, P).$$

The *LTS estimator* is a $(d-1)$-dimensional hyperplane $\boldsymbol{\beta}$ of minimum LTS cost, which we denote by $\boldsymbol{\beta}^{(\text{LTS})}(P, h)$. We let $\Delta^{(\text{LTS})}(P, h)$ denote the associated LTS cost of this hyperplane. The *LTS problem* is that of computing this hyperplane. We refer to the points having the $h$ smallest squared residuals as *inliers*, and the remaining points as *outliers*. This generalizes the ordinary least squares estimator (when $h = n$). Typically, $h$ is set to some constant fraction of $n$ based on the expected number of outliers. To guarantee a unique solution it is often assumed that $h$ is at least $n/2$. The statistical properties of LTS are discussed in [18] and [20]. Because we will only considering the LTS problem here, we will simplify the notation by dropping the reference to LTS, and express the above quantities as $\Delta_{\boldsymbol{\beta}}(P, h)$, $\boldsymbol{\beta}(P, h)$, and $\Delta(P, h)$.

To date, the computational complexity of LTS is less well understood than that of LMS. Hössjer [11] presented an $O(n^2 \log n)$ algorithm for LTS in the plane based on plane sweep. In a companion paper [15], we presented an exact algorithm whose running time is $O(n^2)$ for the planar case and runs in $O(n^{d+1})$ time in any fixed dimension $d$. For large $n$ these running times may be unacceptably slow, even in spaces of moderate dimension.

Given these relatively high running times, it is natural to consider whether this problem can be solved approximately. There are a few possible ways to formulate LTS as an approximation problem, either by approximating the residual, by approximating the quantile, or both. The following formulations were introduced in [15]. The approximation parameters $\varepsilon_r$ and $\varepsilon_q$ denote the allowed residual and quantile errors, respectively.

**Residual Approximation:** The requirement of minimizing the sum of squared residuals is relaxed. Given $0 < \varepsilon_r$, an *$\varepsilon_r$-residual approximation* is any hyperplane $\boldsymbol{\beta}$ such that

$$\Delta_{\boldsymbol{\beta}}(P, h) \;\leq\; (1 + \varepsilon_r)\, \Delta(P, h).$$

**Quantile Approximation:** As we shall see, much of the complexity of LTS arises because of the requirement that *exactly* $h$ points be considered. We can relax this requirement by introducing a parameter $0 < \varepsilon_q < h/n$ and requiring that the fraction of inliers used is smaller by $\varepsilon_q$. Let $h^- = h - \lfloor n\varepsilon_q \rfloor$. An *$\varepsilon_q$-quantile approximation* is any hyperplane $\boldsymbol{\beta}$ such that

$$\frac{\Delta_{\boldsymbol{\beta}}(P, h^-)}{h^-} \;\leq\; \frac{\Delta(P, h)}{h}.$$

(The normalizing factors $1/h^-$ and $1/h$ are provided since the costs involve sums over a different number of terms.)

**Hybrid Approximation:** The above approximations can be merged into a single approximation. Given $\varepsilon_r$ and $\varepsilon_q$ as in the previous two approximations, let $h^-$ be as defined above. An *$(\varepsilon_r, \varepsilon_q)$-hybrid approximation* is any hyperplane $\boldsymbol{\beta}$ such that

$$\frac{\Delta_{\boldsymbol{\beta}}(P, h^-)}{h^-} \;\leq\; (1 + \varepsilon_r)\, \frac{\Delta(P, h)}{h}.$$

LTS and LMS are robust versions of the well known least squares ($L_2$) and Chebyshev ($L_\infty$) estimators, respectively. A third example is the *least trimmed sum of absolute residuals*, or *LTA*. This is a trimmed version of the $L_1$ estimator, in which the objective is to minimize the sum of squares of the $h$ smallest *absolute values* of the residuals. By analogy, approximations can be defined for the other trimmed estimators. Such algorithms have been presented for LMS and LTA [4, 5, 9]. In an earlier paper [15], we presented an approximation algorithm for LTS with a running time of roughly $O(n^d/h)$. In the same paper, we presented lower bounds for the LTS and LTA problems, which show that substantial improvements to these bounds, either exact or approximate, are unlikely, since they would imply better bounds for a longstanding open problem in computational geometry, called affine degeneracy. In summary, LTS appears to be a hard problem to solve in the worst case.

There is, however, a very simple and practical approach to LTS. This is the Fast-LTS heuristic of Rousseeuw and Van Driessen [21]. This algorithm (which will be described in detail in Section 2) is based on a combination of random sampling and local improvement. It is based on repeatedly

fitting a hyperplane to a small random sample of points, which is called an *elemental fit* (formal definitions are given later), and then applying a small number of *local improvement steps*, each of which is guaranteed to map an existing fit to one with a lower LTS cost. In practice this approach works very well. However, it provides no assurance on the quality of the final fit. Another example of a local search heuristic is Hawkins' feasible point algorithm [10], but it does not offer any formal performance guarantees either.

While Fast-LTS works well in practice, it does not represent a truly satisfactory algorithmic solution to the LTS problem. It implicitly assumes that the data is sufficiently well conditioned that random elemental fits will be sufficiently close to the optimal fit that subsequent local improvements will converge to the optimum, or close to it. If the data set is very poorly conditioned, however, Fast-LTS may require a very large number of random trials before it succeeds. This raises the question of what constitutes a well-conditioned point set, and whether there is an algorithm that can provide guarantees on the quality of the result.

We present two results to address these questions. First, in Section 2, we present an analysis of a simple randomized $O(n)$-time heuristic, which is based on performing repeated elemental fits. Our analysis extends upon existing results by considering the effect of numerical condition of the inliers on the accuracy of the result. We introduce a numerical measure of the *condition* of *well-conditioned point sets* for multidimensional linear regression. We show that with high probability this heuristic provides a *constant factor approximation* to LTS, under the assumption that the point set is sufficiently well conditioned. Second, in Section 3, we present an approximation algorithm for LTS. The user provides approximation parameters $\varepsilon_r$ and $\varepsilon_q$, and the algorithm returns a hyperplane that is an $(\varepsilon_r, \varepsilon_q)$-hybrid approximation. The algorithm's running time depends on a number of factors, including the number of points, the dimension of the space, and the condition of the point set. We have implemented this algorithm, and in Section 5, we present empirical evidence that for low-dimensional inputs, this algorithm can provide efficiently provide guarantees on the accuracy of the final output.

## 2 Numerical Condition and Elemental Fits

Consider a set of $n$ points $P$ in $\mathbb{R}^d$ and a trimming parameter $h \leq n$. We assume that the points are in general position, so that any subset of $d$ points defines a unique $(d-1)$-dimensional hyperplane. Any hyperplane generated in this manner is called an *elemental fit*. Elemental fits play an important roles in Fast-LTS (described below). In this section we introduce a measure on the numerical condition of a point set, and we establish formal bounds on the accuracy of random elemental fits in terms of this measure.

We begin with a brief presentation of Rousseeuw and van Driessen's Fast-LTS algorithm. (Our description is slightly simplified, and we refer the reader to [21] for details.) The algorithm consists of a series of stages. Each starts by generating a random subset of $P$ of size $d$ and fits a hyperplane $\boldsymbol{\beta}$ to these points. It then generates a series of local improvements as follows. The $h$ points of $P$ that have the smallest squared residuals with respect to $\boldsymbol{\beta}$ are determined. Let $\boldsymbol{\beta}'$ be the least squares linear estimator of these points. Clearly, the LTS cost of $\boldsymbol{\beta}'$ is not greater than that of $\boldsymbol{\beta}$. We set $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$ and repeat the process. This is called a *C-step* or *concentration step*. C-steps are repeated a fixed number of times. (By default, two C-steps are performed.) This constitutes a single stage.

The overall algorithm operates by performing a number of stages (500 by default), where each

stage starts with a new random elemental fit. The ten results with the lowest costs are saved. For each of these ten, C-steps are applied until converging to a local minimum. The algorithm's final output is the hyperplane with the lowest overall cost. Assuming constant dimension, each C-step can be performed in $O(n)$ time, and thus the overall running time is $O(kn)$, where $k$ is the number of stages.

It is easy to see why Fast-LTS performs well in practice. If we succeed in sampling $d$ inliers (which would be expected to occur with probability roughly $(h/n)^d$, assuming at least $h$ inliers) and these points are well distributed, then the resulting elemental fit should be sufficiently close to the optimal solution that the subsequent C-steps will converge to the fit that is very close to the optimum. Thus, although there are no guarantees of good performance, if the data are reasonably well conditioned, then after a sufficiently large number of random starts, it is reasonable to believe that at least one of the resulting fits will be close to the optimum that combined with C-steps, it will produce a nearly optimal solution.

The objective of this section is formalize this intuition. We will ignore the effect of C-steps, and focus only on analyzing the performance of repeated random elemental fits. The lower bounds mentioned earlier [15] imply that any approach based on taking a small number of random samples cannot hope to provide an accurate approximation for all data sets. These lower bounds, however, are based on nearly degenerate point sets. We wish to provide sufficient conditions on the properties of point sets so that with constant probability, a random elemental fit will provide a reasonably good approximation to the optimum LTS hyperplane.

Rousseeuw and Leroy provide an analysis of the number of elemental fits needed to obtain a good fit with a given probability (see Chapter 5 of [20]), but their analysis makes the tacit assumption that the inliers lie on the optimum LTS fit and are otherwise in general position. Such an argument ignores the issue of the numerical stability of the elemental fit. For example, if the inliers are clustered close to a low-dimensional flat, then the resulting elemental fit will be highly unstable.

Given $P$ and $h$, recall that $\boldsymbol{\beta}_h^*(P)$ denotes the optimum LTS estimator for $P$. Let $\mathcal{E}(P)$ denote the collection of $d$-element subsets of the index set $\{1, \ldots, n\}$. Given $E = \{i_1, \ldots, i_d\} \in \mathcal{E}$, let $X_E$ denote the $d \times d$ matrix whose columns are the coordinate vectors of the corresponding independent variables:

$$X_E \;=\; \left[\boldsymbol{x}_{i_1} \;\middle|\; \boldsymbol{x}_{i_2} \;\middle|\; \ldots \;\middle|\; \boldsymbol{x}_{i_d}\right].$$

Let $\boldsymbol{y}_E$ be the row $d$-vector of associated dependent variables $(y_{i_1}, \ldots, y_{i_d})$. If we assume general position, these vectors are linearly independent, and so there is a unique $d$-element row vector $\boldsymbol{\beta}'$ such that

$$\boldsymbol{y}_E \;=\; \boldsymbol{\beta}' X_E, \qquad \text{that is} \qquad \boldsymbol{\beta}' \;=\; \boldsymbol{y}_E X_E^{-1}.$$

(See Fig. 1.) For our subsequent analysis, we also define $\boldsymbol{y}_E^* = \boldsymbol{\beta}^* X_E$ to be the vector of $y$-values resulting from evaluating the optimum LTS hyperplane at the points of the elemental set. Thus, we have $\boldsymbol{\beta}^* = \boldsymbol{y}_E^* X_E^{-1}$.

We begin with a few standard definitions [6, 7]. Consider a $d$-vector $\boldsymbol{x}$ and a $d \times d$ matrix $X$. Let $X_{i,j}$ denote the element in row $i$ and column $j$ of $X$, and let $X_j$ denote the $j$th column vector.

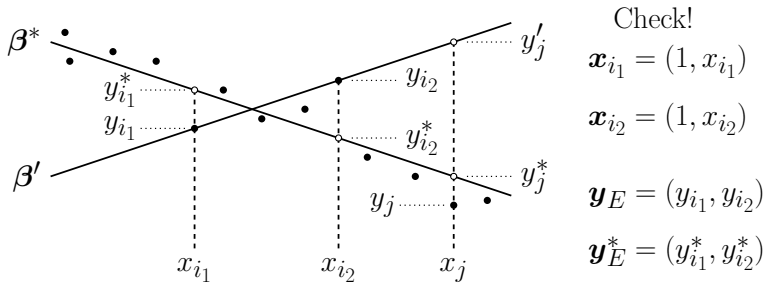



Fig. 1: Estimated and optimal fits for $E = \{i_1, i_2\}$ and $\boldsymbol{y}_E = (y_{i_1}, y_{i_2})$. Solid points indicate points of $P$ and hollow points are computed quantities.

The norms of $\boldsymbol{x}$ and $X$ ($L_2$ and Frobenius, respectively) are defined to be

$$\|\boldsymbol{x}\| \;=\; \left(\sum_i x_i^2\right)^{1/2} \qquad \|X\| \;=\; \left(\sum_j \sum_i X_{i,j}^2\right)^{1/2} \;=\; \left(\sum_j \|X_j\|^2\right)^{1/2}.$$

Let $\kappa(X)$ denote the *condition number* of $X$ relative to this norm, that is, $\kappa(X) = \|X\| \cdot \|X^{-1}\|$. The concept of condition number is well known in matrix algebra as a means to characterize the stability of linear systems involving $X$ [7]. It is a dimensionless quantity, where lower values mean greater stability.

Our characterization of a well-conditioned set is based on two assumptions made on the independent variables of the inliers. Let $P$ be an $n$-element point set in $\mathbb{R}^d$, and given a trimming parameter $h$, let $H \subseteq \{1, 2, \ldots, n\}$ denote the $h$-element index set of the inliers relative to $P$'s optimal LTS hyperplane. The first assumption states that with constant probability, a random elemental set of inliers provides a stable elemental fit. This is expressed by bounding the median condition number of $X_E$ for $E \in \mathcal{E}(H)$. (That is, we are taking the median over the collection of all $d$-element subsets of the $h$ inliers.) This assumption alone is not sufficient, since even a minuscule error in the placement of the hyperplane can be fatal if any inlier is located so far away that its associated residual dominates the overall cost. Such a point is called a *leverage point* [19]. The second assumption effectively implies that there are no leverage point inliers, by bounding the ratio between the average squared norm and median squared norm of the points $\boldsymbol{x}_i$ for $i \in H$. More formally, given two positive real parameters $\gamma$ and $\lambda$, we say that $P$ is $(\gamma, \lambda)$-*well conditioned* if

$$\text{(i):}\;\; \operatorname*{med}_{E \in \mathcal{E}(H)} \kappa(X_E) \;\leq\; \gamma \qquad\qquad \text{(ii):}\;\; \frac{\sum_{j \in H} \|\boldsymbol{x}_j\|^2}{h \cdot \operatorname{med}_{j \in H} \|\boldsymbol{x}_j\|^2} \;\leq\; \lambda^2.$$

Note that these assumptions constrain only the independent variables (not the dependent variables), and they apply only to the set of inliers of the optimal fit (and thus there are no assumptions imposed on the outliers). Also note that the use of the median in the two assumptions is not critical. At the expense of a constant factor in the analysis, we could have replaced the median with the $k$th smallest value, where $h/k$ is a constant.

The main result of this section is given in the following theorem. It shows that depending on the condition of the point set, after a sufficiently large number of repetitions of random elemental fits, we can achieve a constant-factor (residual) approximation for LTS with arbitrarily high confidence.

**Theorem 1** *Let $P$ be an $n$-element point set in $\mathbb{R}^d$ that is $(\gamma, \lambda)$-well conditioned for some $\gamma$ and $\lambda$. Let $h$ be a trimming parameter, where $h = \lceil qn \rceil$, for some constant $0 < q \leq 1$. Then for any $0 < \delta < 1$, after $O((\ln(1/\delta))/q^d)$ elemental fits, with probability at least $1 - \delta$, at least one of these fits $\boldsymbol{\beta}'$ will satisfy*

$$\Delta_{\boldsymbol{\beta}'}(P, h) \;\leq\; (1 + \sqrt{2} \cdot \gamma \cdot \lambda)\Delta(P, h).$$

The proof relies on the following lemma, which shows that with constant probability, a single elemental fit provides the desired approximation bound.

**Lemma 1** *Let $P$ be an $n$-element point set in $\mathbb{R}^d$ that is $(\gamma, \lambda)$-well conditioned for some $\gamma$ and $\lambda$. Let $h$ be a trimming parameter, where $h = \lceil qn \rceil$, for some constant $0 < q \leq 1$. Let $E$ be a random element of $\mathcal{E}(P)$, and let $\boldsymbol{\beta}'$ denote the resulting random elemental fit. There exists a constant $c$ such that with probability at least $c \cdot q^d$, we have*

$$\Delta_{\boldsymbol{\beta}'}(P, h) \;\leq\; (1 + \sqrt{2} \cdot \gamma \cdot \lambda)\Delta(P, h).$$

Assuming this lemma, Theorem 1 follows directly. In particular, by repeating $m$ independent random elemental fits, the probability of failing to achieve the stated approximation bound is at most $(1 - c \cdot q^d)^m \leq \exp(-cmq^d)$, for some constant $c$. By selecting $m \geq (\ln(1/\delta))/(c\,q^d)$, the probability of failure is at most $\delta$, from which Theorem 1 follows.

The remainder of the section is devoted to proving Lemma 1. Recall the definitions given earlier for the norms of a matrix $X$, column vector $\boldsymbol{x}$, and row vector $\boldsymbol{y}$. The following are easy consequences of these definitions and the Cauchy-Schwarz inequality:

$$|\boldsymbol{y}\boldsymbol{x}| \;\leq\; \|\boldsymbol{y}\| \cdot \|\boldsymbol{x}\| \qquad\qquad \|\boldsymbol{y}X\| \;\leq\; \|\boldsymbol{y}\| \cdot \|X\|.$$

With probability at least $q$, a random index from $\{1, 2, \ldots, n\}$ is drawn from $H$, and so the probability that a random element of $\mathcal{E}(P)$ lies in $\mathcal{E}(H)$ approaches $q^d$ for sufficiently large $n$. (Note that we sample without replacement.) The remainder of the proof is conditioned on this assumption.

Let $\boldsymbol{\beta}^*$ denote the optimal LTS hyperplane. Since $P$ and $h$ are fixed throughout, let $\Delta^*$ and $\Delta'$ denote $\Delta_{\boldsymbol{\beta}^*}(P, h)$ and $\Delta_{\boldsymbol{\beta}'}(P, h)$, respectively. Let $H$ denote indices of the $h$ points of $P$ whose residuals with respect to $\boldsymbol{\beta}^*$ are the smallest. We make use of the following technical result. The proof involves fairly straightforward manipulations using standard inequalities from linear algebra.

**Lemma 2** *Given $\Delta^*$ and $\Delta'$ as defined above,*

$$\frac{\Delta'}{\Delta^*} - 1 \;\leq\; \left( \frac{\|\boldsymbol{y}_E - \boldsymbol{y}_E^*\|^2}{\sum_{j \in H} (y_j - y_j^*)^2} \right)^{1/2} \cdot \; (\|X_E^{-1}\| \cdot \|X_E\|) \; \cdot \left( \frac{\sum_{j \in H} \|\boldsymbol{x}_j\|^2}{\|X_E\|^2} \right)^{1/2}.$$

**Proof**: Recall that $\boldsymbol{\beta}^*$ denotes the optimal LTS hyperplane, $\Delta^*$ and $\Delta'$ denote $\Delta_{\boldsymbol{\beta}^*}(P, h)$ and $\Delta_{\boldsymbol{\beta}'}(P, h)$, respectively, and $H$ denotes the indices of the $h$ points of $P$ whose residuals with respect to $\boldsymbol{\beta}^*$ are the smallest, that is, the indices of the inliers. By definition of the LTS cost we have

$$h(\Delta^*)^2 \;=\; \sum_{j \in H} r_j^2(\boldsymbol{\beta}^*, P) \;=\; \sum_{j \in H} (y_j - \boldsymbol{\beta}^* \boldsymbol{x}_j)^2.$$

Let $H'$ denote the indices of the $h$ points of $P$ whose squared residuals with respect to $\boldsymbol{\beta}'$ are the smallest. Therefore,

$$h(\Delta')^2 \;=\; \sum_{j\in H'} r_j^2(\boldsymbol{\beta}', P) \;=\; \sum_{j\in H'} (y_j - \boldsymbol{\beta}'\boldsymbol{x}_j)^2 \;\le\; \sum_{j\in H} (y_j - \boldsymbol{\beta}'\boldsymbol{x}_j)^2.$$

(See, for example, the right side of Fig. 1.) Thus, we have

$$\sqrt{h}(\Delta' - \Delta^*) \;\le\; \left(\sum_{j\in H} (y_j - \boldsymbol{\beta}'\boldsymbol{x}_j)^2\right)^{1/2} \;-\; \left(\sum_{j\in H} (y_j - \boldsymbol{\beta}^*\boldsymbol{x}_j)^2\right)^{1/2}.$$

We may interpret $\Delta'$ as the Euclidean distance between two points in $\mathbb{R}^h$, particularly $(y_1, \ldots, y_h)$ and $(\boldsymbol{\beta}'\boldsymbol{x}_1, \ldots, \boldsymbol{\beta}'\boldsymbol{x}_h)$. Similarly, we may interpret $\Delta^*$ as the Euclidean distance between $(y_1, \ldots, y_h)$ and $(\boldsymbol{\beta}^*\boldsymbol{x}_1, \ldots \boldsymbol{\beta}^*\boldsymbol{x}_h)$. By the triangle inequality and the above inequalities regarding norms, we have

$$\begin{aligned}
\sqrt{h}(\Delta' - \Delta^*) \;&\le\; \left(\textstyle\sum_{j\in H} (\boldsymbol{\beta}'\boldsymbol{x}_j - \boldsymbol{\beta}^*\boldsymbol{x}_j)^2\right)^{1/2} \;\le\; \left(\textstyle\sum_{j\in H} ((\boldsymbol{\beta}' - \boldsymbol{\beta}^*)\boldsymbol{x}_j)^2\right)^{1/2} \\
&\le\; \left(\textstyle\sum_{j\in H} \|\boldsymbol{\beta}' - \boldsymbol{\beta}^*\|^2 \cdot \|\boldsymbol{x}_j\|^2\right)^{1/2} \;\le\; \|\boldsymbol{\beta}' - \boldsymbol{\beta}^*\| \left(\textstyle\sum_{j\in H} \|\boldsymbol{x}_j\|^2\right)^{1/2}.
\end{aligned}$$

By the earlier observations relating $\boldsymbol{\beta}'$ and $\boldsymbol{\beta}^*$ to $\boldsymbol{y}_E$ and $\boldsymbol{y}_E^*$, we obtain

$$\begin{aligned}
\sqrt{h}(\Delta' - \Delta^*) \;&\le\; \|\boldsymbol{y}_E X_E^{-1} - \boldsymbol{y}_E^* X_E^{-1}\| \left(\textstyle\sum_{j\in H} \|\boldsymbol{x}_j\|^2\right)^{1/2} \\
&\le\; \|(\boldsymbol{y}_E - \boldsymbol{y}_E^*)\| \cdot \|X_E^{-1}\| \left(\textstyle\sum_{j\in H} \|\boldsymbol{x}_j\|^2\right)^{1/2}.
\end{aligned}$$

For each $\boldsymbol{x}_j \in P$, let $y_j^* = \boldsymbol{\beta}^*\boldsymbol{x}_j$. We can write $\Delta^*$ as $\left(\frac{1}{h}\sum_{j\in H}(y_j - y_j^*)^2\right)^{1/2}$, from which it follows that

$$\begin{aligned}
\frac{\Delta'}{\Delta^*} - 1 \;&=\; \frac{\sqrt{h}(\Delta' - \Delta^*)}{\sqrt{h}\Delta^*} \;\le\; \|\boldsymbol{y}_E - \boldsymbol{y}_E^*\| \cdot \|X_E^{-1}\| \left(\textstyle\sum_{j\in H} \|\boldsymbol{x}_j\|^2\right)^{1/2} \Big/ \left(\textstyle\sum_{j\in H} (y_j - y_j^*)^2\right)^{1/2} \\
&\le\; \left(\frac{\|\boldsymbol{y}_E - \boldsymbol{y}_E^*\|^2}{\sum_{j\in H}(y_j - y_j^*)^2}\right)^{1/2} \cdot \left(\|X_E^{-1}\| \cdot \|X_E\|\right) \cdot \left(\frac{\sum_{j\in H} \|\boldsymbol{x}_j\|^2}{\|X_E\|^2}\right)^{1/2},
\end{aligned}$$

where in the last inequality we have simply multiplied and divided by $\|X_E\|$. This completes the proof. □

Given this result, let us analyze each of these three factors separately. By definition, the middle factor is just $\kappa(X_E)$. To analyze the first factor, observe that $\|\boldsymbol{y}_E - \boldsymbol{y}_E^*\|^2 = \sum_{j\in E}(y_j - y_j^*)^2$. By our earlier assumption, $E$ is a random subset of $H$, and so with probability approaching $1/2^d$ for large $n$, every element of $\{(y_j - y_j^*)^2 \;:\; j \in E\}$ is at most $\text{med}_{j\in H}(y_j - y_j^*)^2$. Since at least half of the elements of the multiset $\{(y_j - y_j^*)^2 \;:\; j \in H\}$ exceed the median of the set, it follows that $\sum_{j\in H}(y_j - y_j^*)^2 \ge (h/2)\,\text{med}_{j\in H}(y_j - y_j^*)^2$. Thus, with probability approaching $1/2^d$ (and under the assumption that $E \subseteq H$) we have

$$\frac{\|\boldsymbol{y}_E - \boldsymbol{y}_E^*\|^2}{\sum_{j\in H}(y_j - y_j^*)^2} \;\le\; \frac{\text{med}_{j\in H}(y_j - y_j^*)^2}{(h/2)\,\text{med}_{j\in H}(y_j - y_j^*)^2} \;\le\; \frac{2}{h}.$$

To analyze the third factor, recall that from the definition of the Frobenius norm, $\|X_E\|^2 = \sum_{j \in E} \|\boldsymbol{x}_j\|^2$. By our earlier assumption, $E$ is a random subset of $H$, and so with probability at least $1 - (1/2)^d \geq 1/2$, at least one element of $\{\|\boldsymbol{x}_j\|^2 \ : \ j \in E\}$ exceeds $\text{med}_{j \in H} \|\boldsymbol{x}_j\|^2$, which implies that $\|X_E\|^2$ exceeds the median as well. Combining this with property (ii) of well-conditioned sets, with probability at least $1/2$ (and under the assumption that $E \subseteq H$) we have

$$\frac{\sum_{j \in H} \|\boldsymbol{x}_j\|^2}{\|X_E\|^2} \ \leq \ \frac{\lambda^2 \, h \cdot \text{med}_{j \in H} \|\boldsymbol{x}_j\|^2}{\text{med}_{j \in H} \|\boldsymbol{x}_j\|^2} \ \leq \ \lambda^2 h.$$

Combining our bounds on all three factors together with condition (i) of well-conditioned sets, we conclude that for large $n$, with probability approaching $q^d(1/2^d)(1/2) = (q/2)^d/2 = \Omega(q^d)$, we have

$$\frac{\Delta'}{\Delta^*} - 1 \ \leq \ \left(\frac{2}{h}\right)^{1/2} \kappa(X_E) \, (\lambda\sqrt{h}) \ \leq \ \sqrt{2} \cdot \kappa(X_E) \cdot \lambda.$$

Since $E$ is a random element of $\mathcal{E}(H)$, with probability at least $1/2$, $\kappa(X_E)$ is not greater than $\text{med}_{F \in \mathcal{E}(H)} \kappa(X_F)$, and so by condition (i) of well-conditioned sets we have $\kappa(X_E) \leq \gamma$, which yields

$$\frac{\Delta'}{\Delta^*} \ \leq \ 1 + \sqrt{2} \cdot \kappa(X_E) \cdot \lambda \ \leq \ 1 + \sqrt{2} \cdot \gamma \cdot \lambda,$$

with probability at least $(q/2)^d/4$. By setting $c = 1/2^{d+2}$ we obtain the desired conclusion, thus completing the proof of Lemma 1.

In the typical case, $h = \lceil n/2 \rceil$, and the bounds of the lemma hold with probability roughly $1/4^{d+1}$. Although this simple analysis ignores the effect of the C-steps (which we leave as an open problem), it does suggest a more principled approach for selecting the number of iterations for Fast-LTS, based on the failure probability $\delta$, the trimming quantile $q = h/n$, and the dimension $d$.

## 3 An Adaptive Approximation Algorithm

In this section we present an $(\varepsilon_r, \varepsilon_q)$-hybrid approximation algorithm for LTS, which we call *Adaptive-LTS*. The input to the algorithm is an $n$-element point set $P$, a trimming parameter $h \leq n$, a residual approximation parameter $\varepsilon_r \geq 0$, and a quantile approximation parameter $\varepsilon_q$, where $0 \leq \varepsilon_q < h/n$. This algorithm is based on a technique called *geometric branch-and-bound*, which involves a recursive subdivision of the solution space in search of the optimal solution. Geometric branch-and-bound has been used in other applications of geometric search, in particular it has been applied in the context of image registration by Huttenlocher and Rucklidge [12, 22, 23], Hagedoorn and Veltkamp [8], and Mount *et al.* [13].

Before presenting the algorithm we provide a brief overview of the approach. The solution to the LTS problem can be represented as a $d$-dimensional vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$, consisting of the coefficients of the estimated hyperplane. For a given point set $P$ and trimming parameter $h$, we can associate each point $\boldsymbol{\beta} \in \mathbb{R}^d$ with the associated LTS cost, $\Delta_{\boldsymbol{\beta}}(P, h)$. This defines a scalar field over $\mathbb{R}^d$, called the *solution space*. The LTS problem reduces to computing the lowest point in this space.

In general, geometric branch-and-bound solves a problem approximately by subdividing the solution space into regions (typically hyperrectangles), called *cells*. For a given cell $C$, let $\Delta_C(P, h)$

denote the smallest value of $\Delta_{\boldsymbol{\beta}}(P,h)$, among all $\boldsymbol{\beta} \in C$. For our problem, this is the minimum LTS cost under the constraint that the coefficient vector of the hyperplane is chosen from $C$. The algorithm computes a lower bound and an upper bound on $\Delta_C(P,h)$ (which we describe below). If the lower bound is sufficiently large (depending on $\varepsilon_r$ and $\varepsilon_q$) compared to the smallest upper bound among all the other cells, we may safely conclude that $C$ cannot contain the optimal solution to the LTS problem, and hence we may eliminate it from further consideration. We say that $C$ is *killed* when this happens. Otherwise, $C$ is said to be *active*.

A geometric branch-and-bound algorithm runs in a sequence of *stages*. At the start of any stage, the algorithm maintains a collection of all the currently active cells. During each stage, one of the active cells is selected and is processed by subdividing it into two (or generally more) smaller cells, called its *children*. We apply the above tests to each of these children. If they survive, they are added to the collection of active cells for future processing. The algorithm terminates when no more active cells remain.

The precise implementation of a geometric branch-and-bound algorithm depends on many elements, such as:

- How are cells represented?
- What is the initial cell?
- How is a cell subdivided?
- How are the upper and lower bounds computed for each cell?
- Under what conditions is a cell killed?
- Among the active cells, which cell is selected next for processing?

In the rest of this section, we will describe each of these elements in greater detail.

**Cells and their representation.** Although a solution to LTS is fully specified by the $d$-vector of coefficients, we will use a slightly more concise representation. Recall that a hyperplane is represented by the equation $y = \sum_{j=1}^{d-1} \beta_j x_j + \beta_d$, where $\beta_d$ is the intercept term. Rather than store all $d$-coefficients, we will store only the first $d-1$ coefficients, the *slope coefficients*. In particular, we represent a solution to LTS by a vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{d-1}) \in \mathbb{R}^{d-1}$. We will show in Section 4 that given such a vector, it is possible to compute the optimum value of $\beta_d$ in $O(n \log n)$ time by solving a 1-dimensional LTS problem. Because the time to process a cell is already $O(n)$, this represents only a modest increase in the processing time for each cell. Because running times tend to grow exponentially with the dimension of the solution space, this reduction in dimension can significantly reduce the algorithm's overall running time. Given $\boldsymbol{\beta} \in \mathbb{R}^{d-1}$, let $\Delta_{\boldsymbol{\beta}}$ denote the minimum LTS cost achievable by extending $\boldsymbol{\beta}$ to a $d$-dimensional vector.

For our algorithm, a *cell* $C$ is a closed, axis-parallel hyperrectangle in $\mathbb{R}^{d-1}$. It is represented by a pair of vectors $\boldsymbol{\beta}^-, \boldsymbol{\beta}^+ \in \mathbb{R}^{d-1}$, and consists of the Cartesian product of intervals $\prod_{i=1}^{d-1}[\beta_i^-, \beta_i^+]$. Let $\Delta_C(P,h) = \min_{\boldsymbol{\beta} \in C} \Delta_{\boldsymbol{\beta}}(P,h)$ denote the smallest LTS cost of any point of $C$.

**Sampled elemental fits.** We employ random elemental fits to help guide the search algorithm. Before the algorithm begins, a user-specified number of randomly sampled elemental fits is generated. (We used 512 in our experiments.) Each elemental fit is associated with a $(d-1)$-dimensional vector of slope coefficients. Let $S_0$ denote this set of vectors. For each active cell $C$, let $S(C)$ denote the elements of $S_0$ that lie within $C$. Let $B(C)$ denote the minimum bounding hyperrectangle that encloses the elements of $S(C)$. Each active cell $C$ stores the elements of $S(C)$ in a list and $B(C)$.

**Initial cell.** We generate the initial cell by computing the smallest axis-aligned hyperrectangle in $\mathbb{R}^{d-1}$ that encloses the elements of $S_0$. (Our implementation allows this hyperrectangle to be expanded about its center by a user-specified scale factor, and it also has an option that allows the user to override this and specify the initial cell.)

**Cell subdivision.** We will discuss how active cells are selected for processing below. Once a cell $C$ is selected, it is subdivided into two hyperrectangles by an axis parallel hyperplane. This hyperplane is chosen as follows. First, if $S(C)$ is not empty, then the algorithm computes the longest side length of $B(C)$. It then sorts the vectors of $S(C)$ along the axis that is parallel to this side, and splits the set by a hyperplane that is orthogonal to this side and passes through the median of the sorted sequence (see Fig. 2(a)). If $S(C)$ is empty, the algorithm bisects the cell through its midpoint by a hyperplane that is orthogonal to the cell's longest side length (see Fig. 2(b)). The resulting cells, denoted $C_0$ and $C_1$ in the figure, are called $C$'s *children*.

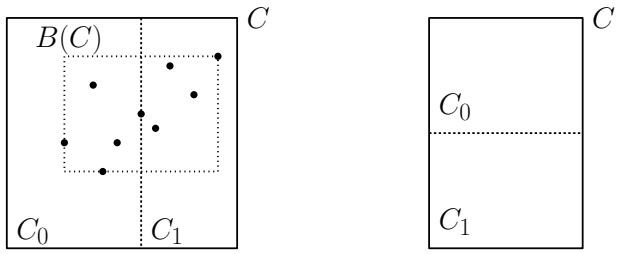



Fig. 2: Subdividing a cell (a) when $S(C)$ is nonempty and (b) when $S(C)$ is empty.

An example of the subdivision process is shown in Fig. 3 for a 2-dimensional solution space. In (a) we show a cell $C$ and its samples $S(C)$. In (b) we show the subdivision that results by repeatedly splitting each cell through its median sample. Note that the subdivision has the desirable property that it tends to produce more cells in regions of space where the density of samples is higher. In (c) we show the effect of two additional steps of subdivision by splitting the cell through its midpoint. (This assumes that all the cells are selected for processing and none are killed.)

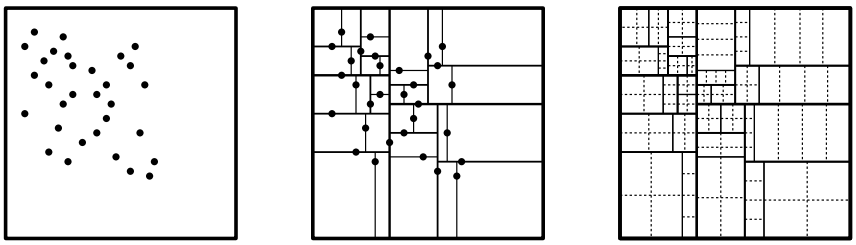



Fig. 3: Example of the subdivision process: (a) initial cell and samples, (b) sample-based subdivision, (c) two more stages of midpoint subdivision.

**Upper and lower bounds.** Recall that the upper bound for a cell $C$, which we denote by $\Delta^+(C)$, is an upper bound on the smallest LTS cost among all hyperplanes whose slope coefficients lie within $C$. To compute such an upper bound, it suffices to sample a representative slope from within the cell and then compute the $y$-intercept of the hyperplane with this slope that minimizes the LTS cost (assuming the trimming parameter $h^-$). Our algorithm chooses the representative slope as follows. If $S(C)$ is nonempty, we take the median point that was chosen in the cell-subdivision process. Otherwise, we take the cell's midpoint. In Section 4, we will show how to compute the optimum $y$-intercept by reduction to a 1-dimensional LTS problem, which can be solved in $O(n \log n)$ time. To further improve the upper bound, we then apply a user-specified number of C-steps. (We used two C-steps in our experiments.)

The computation of the lower bound for the cell $C$, denoted $\Delta^-(C)$, is more complex than the upper bound, because it requires consideration of all the slope coefficient vectors within the cell. The approach is similar to the upper bound computation, but rather than computing the optimum $y$-intercept for a specific slope, we compute the optimum $y$-intercept under the assumption that the slope could be *any* point of $C$ (assuming the trimming parameter $h$). In Section 4, we will show how to do this in $O(n \log n)$ time by solving a problem that we call the 1-dimensional interval LTS problem.

**Killing cells.** Recall that our objective is to report a hyperplane $\boldsymbol{\beta}$ that satisfies

$$\frac{\Delta_{\boldsymbol{\beta}}(P, h^-)}{h^-} \;\leq\; (1 + \varepsilon_r)\, \frac{\Delta(P, h)}{h}.$$

where $h^- = h - \lfloor n\varepsilon_q \rfloor$. Let $\widehat{\boldsymbol{\beta}}$ denote the hyperplane associated with the smallest upper bound encountered so far in the search (assuming the trimming parameter $h^-$). Given a cell $C$, we assert that $C$ can be killed if

$$\Delta^-(C) \;>\; \frac{h}{(1 + \varepsilon_r)h^-} \Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-). \tag{1}$$

The reason is that for any $\boldsymbol{\beta}$ in such a cell, we have $\Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-)/h^- < (1 + \varepsilon_r)\Delta_{\boldsymbol{\beta}}(P, h)/h$. Thus, $C$ cannot provide a solution that would contradict the hypothesis that $\widehat{\boldsymbol{\beta}}$ is a valid hybrid approximation. Therefore, it is safe to eliminate $C$ from further consideration. (Note that future iterations might change $\widehat{\boldsymbol{\beta}}$, but only in a manner than would decrease $\Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-)$. Thus, the decision to kill a cell never needs to be reconsidered.)

It is not hard to show that whenever the ratio between a cell's upper bound and lower bound becomes smaller than $(1 + \varepsilon_r)h^-/h$, the cell will be killed. This is because $\widehat{\boldsymbol{\beta}}$ was selected from the processed cell having the smallest value of $\Delta^+$, and so

$$\Delta^-(C) \;>\; \frac{h}{(1 + \varepsilon_r)h^-} \Delta^+(C) \;\geq\; \frac{h}{(1 + \varepsilon_r)h^-} \Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-).$$

This implies that $C$ satisfies the condition in Eq. (1) for being killed. It follows that the algorithm will terminate once all cells have been subdivided to such a small size that this condition holds.

**Cell processing order.** An important element in the design of an efficient geometric branch-and-bound algorithm is the order in which active cells are selected for processing. When a cell is selected for processing, there are two desirable outcomes we might hope for. First, the representative

hyperplane from this cell will have a lower LTS cost than the best hyperplane $\widehat{\boldsymbol{\beta}}$ seen so far. This will bring us closer to the optimum LTS cost, and it has the benefit that future cells are more likely to be killed by the condition of Eq. (1). Second, the lower bound of this cell will be so high that the cell will be killed, thus reducing the number of active cells that need to be considered. This raises the question of whether there are easily computable properties of the active cells that are highly correlated with either of these desired outcomes. We considered the following potential criteria for selecting the next cell:

(1) *Max-samples*: Select the active cell that contains the largest number of randomly sampled elemental fits, that is, the cell $C$ that maximizes $|S(C)|$.

(2) *Min-lower-bound*: Select the active cell having the smallest lower bound, $\Delta^-(C)$.

(3) *Min-upper-bound*: Select the active cell having the smallest lower bound, $\Delta^+(C)$.

(4) *Oldest-cell*: Select the active cell that has been waiting the longest to be processed.

Based on our experience, none of these criteria alone is satisfactory throughout the entire search. At the start of the algorithm, when many cells have a large number of samples, criterion (1) tends to work well because it favors cells that are representative of typical elemental fits. But as the algorithm proceeds, and the number of samples per cell decreases, this criteria cannot distinguish one cell from another. Criteria (2) and (3) are generally good in the middle phases of the algorithm. But when these two criteria are used exclusively, cells that are far from the LTS optimum are never processed. Adding criterion (4) remedies this.

Our approach to selecting the next active cell employs an adaptive strategy. The algorithm assigns a weight to each of the four criteria. Initially, all the criteria are given the same weight. At the start of each stage, the algorithm selects the next criteria randomly based on these weights. That is, if the current weights are denoted $w_1, \ldots, w_4$, criterion $i$ is selected with probability $w_i/(w_1 + \cdots + w_4)$. We select the next cell using this criterion. After processing this cell, we either reward or penalize this criterion depending on the outcome. First, we compute the ratio between the cell's new lower bound and the best LTS cost seen so far, that is, $\rho^- = \Delta^-(C)/\Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-)$. Observe that $0 \leq \rho^- \leq 1$, and $\rho^-$ increases as the lower bound approaches the current best LTS cost. With probability $\rho^-$ we increase the current criteria's weight 50%, and otherwise we decrease it by 10%. Second, we compute the inverse of the ratio between the cell's new upper bound and the best LTS cost seen so far, that is, $\rho^+ = \Delta_{\widehat{\boldsymbol{\beta}}}(P, h^-)/\Delta^+(C)$. Again, $0 \leq \rho^+ \leq 1$, and $\rho^+$ increases as the upper bound approaches the best LTS cost. With probability $\rho^+$ we increase the current criteria's weight 50%, and otherwise we decrease it by 10%.

**The complete algorithm.** The algorithm begins by generating the initial sample $S_0$ of random elemental fits and the initial cell $C_0$ as the smallest bounding hyperrectangle containing this cell. This cell is added to the set of active cells. The above cell-selection strategy is applied to determine the next cell $C$ from this set. By the subdivision process, it splits this cell into two children cells $C_0$ and $C_1$. The set of samples $S(C)$ is then partitioned and assigned to these two cells, as $S(C_0)$ and $S(C_1)$. Next, for each of these children, the lower and upper bounds are computed as described earlier. If the upper bound cost is smaller than the current best LTS cost, then we update the current best LTS fit $\widehat{\boldsymbol{\beta}}$ and its associated cost. For each child cell, we test whether it can be killed

by applying the condition of Eq. (1). If the cell is not killed, it is added to the list of active cells. In either case, the selection criteria that was used to determine this cell is rewarded or penalized, as described above. When the set of active cells is empty (or after a user-specified number of stages) the algorithm terminates and reports the current best fit $\widehat{\boldsymbol{\beta}}$ and its cost.

By the remarks made earlier regarding when cells are killed, if the initial cell contains a valid hybrid approximation and the algorithm terminates due to all cells being killed, the resulting hyperplane $\widehat{\boldsymbol{\beta}}$ is a valid $(\varepsilon_r, \varepsilon_q)$-hybrid approximation to the LTS estimator for $P$.

Unfortunately, it is not easy to provide a good asymptotic bound on the algorithm's execution time. Unlike Fast-LTS, which always runs for a fixed number of iterations irrespective of the structure of the data set, the branch-and-bound algorithm adapts its behavior in accordance with the structural properties of the point set, and the desired accuracy of the final result. We can, however, bound the running time of each stage. In the next section we will show that the upper and lower bounds, $\Delta^+(C)$ and $\Delta^-(C)$, can each be computed in $O(n \log n)$ time, where $n$ is the number of points. Otherwise, the most time consuming part of each stage is the time needed to compute the next active cell. If $m$ denotes the maximum number of active cells at any time, and $s$ denotes the total number of stages until termination, the algorithm's total running time is $O(s(m + n \log n))$. In Section 5 we will present empirical evidence of the algorithm's efficiency on a variety of inputs.

# 4 Computing Upper and Lower Bounds

In this section we describe how to compute the upper and lower bounds for a given cell $C$ in the branch-and-bound algorithm described in the previous section. Recall that a hyperplane is represented as a vector of coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$ in $\mathbb{R}^d$, but our solution space stores only the slope coefficients $(\beta_1, \ldots, \beta_{d-1})$. Each cell $C$ is represented by a pair of vectors $\boldsymbol{\beta}^-, \boldsymbol{\beta}^+ \in \mathbb{R}^{d-1}$, and consists of the Cartesian product of intervals $\prod_{i=1}^{d-1}[\beta_i^-, \beta_i^+]$, which we denote by $[\boldsymbol{\beta}^-, \boldsymbol{\beta}^+]$. Let $\Delta_C(P, h) = \min_{\boldsymbol{\beta} \in C} \Delta_{\boldsymbol{\beta}}(P, h)$ denote the smallest LTS cost of any point of $C$. Our objective is to compute upper and lower bounds on this quantity.

In the previous section we explained how the algorithm samples a representative vector of slope coefficients from $C$. To compute the cell's upper bound, $\Delta^+(C)$, we compute the value of the $y$-intercept coefficient that, when combined with the representative vector of slope coefficients, produces the hyperplane that minimizes the LTS cost. It is straightforward to show that this can be reduced to solving a 1-dimensional LTS problem, which can then be solved in $O(n \log n)$ time [20]. We will present the complete algorithm, since it will be illustrative when we generalize this to the more complex problem of computing the cell's lower bound.

Let $P$ denote the set of $n$ points in $\mathbb{R}^d$, and let $h$ denote the trimming parameter. (Recall from Section 3 that the trimming parameter is $h^-$ when computing the upper bound, but we will refer to it as $h$ in this section to simplify the notation.) Let $p_i = (\boldsymbol{x}_i, y_i)$ denote the $i$th point of $P$, where $\boldsymbol{x}_i \in \mathbb{R}^{d-1}$ and $y_i \in \mathbb{R}$. The squared residual of $p_i$ with respect to a hyperplane $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$, which we denote by $r_i^2(\boldsymbol{\beta})$, is

$$r_i^2(\boldsymbol{\beta}) \;=\; \left(y_i - \sum\nolimits_{j=1}^{d-1} \beta_j x_{i,j} - \beta_d\right)^2 \;=\; \left(\beta_d - \left(y_i - \sum\nolimits_{j=1}^{d-1} \beta_j x_{i,j}\right)\right)^2. \tag{2}$$

Let $b_i = y_i - \sum_{j=1}^{d-1} \beta_j x_{i,j}$. We can visualize $b_i$ as the intersection of $y$-axis and the hyperplane passing through $p_i$ whose slope coefficients match $\boldsymbol{\beta}$ (see Fig. 4(a)). Let $B = \{b_1, \ldots, b_n\}$. For any hyperplane $\boldsymbol{\beta}$, the squared distance $(\beta_d - b_i)^2$ is equal to $r_i^2(\boldsymbol{\beta})$. Therefore, the desired intercept $\beta_d$ of the LTS hyperplane whose slope coefficients match $\boldsymbol{\beta}$'s is the point on the $y$-axis that minimizes the sum of squared distances to its $h$ closest points in $B$. That is, computing $\beta_d$ reduces to solving the (1-dimensional) LTS problem on the $n$-element set $B$.

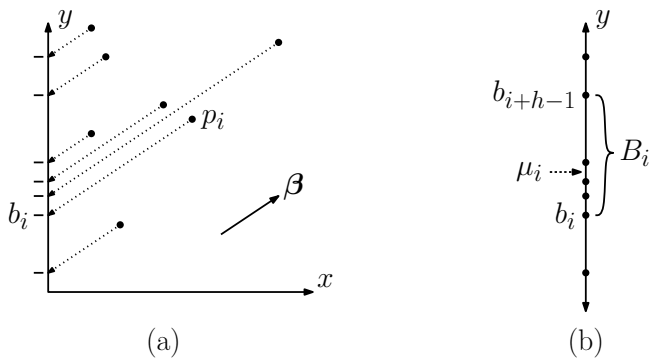



Fig. 4: Computing the optimum $y$-intercept for a given slope $\boldsymbol{\beta}$: (a) projecting the points and (b) the subset $\{b_i, \ldots, b_{i+h-1}\}$.

Before presenting the algorithm for the 1-dimensional LTS problem, we begin with a few observations. First, let us assume that the points of $B$ are sorted in nondecreasing order and have been renumbered so that $b_1 \leq \cdots \leq b_n$. Clearly, the subset of $h$ points $B$ that minimizes the sum of squared distances to any point on the $y$-axis consists of $h$ consecutive points of $B$. For $1 \leq i \leq n-h+1$, define $B_i = \{b_i, \ldots, b_{i+h-1}\}$ (see Fig. 4(b)). The desired upper bound is just the minimum among these $n-h+1$ least-squares costs. This can be computed easily in $O(hn) = O(n^2)$ time, but we will show that it can be solved by an incremental approach in $O(n)$ time.

The point that minimizes the sum of squared deviates within each subset $B_i$ is just the mean of the subset, which we denote by $\mu_i$. The least-squares cost is the subset's variance, which we denote by $\sigma_i^2$. Define $\textsc{sum}(i) = \sum_{k=i}^{i+h-1} b_k$ and $\textsc{ssq}(i) = \sum_{k=i}^{i+h-1} b_k^2$. Given these it is well known that the mean and variance can be computed as

$$\mu_i = \frac{1}{h} \sum_{k=i}^{i+h-1} b_k = \frac{\textsc{sum}(i)}{h}, \tag{3}$$

$$\begin{aligned} \sigma_i^2 &= \frac{1}{h} \sum_{k=i}^{i+h-1} (b_k - \mu_i)^2 = \frac{1}{h} \left( \sum_{k=i}^{i+h-1} b_k^2 - 2\mu_i \sum_{k=i}^{i+h-1} b_k + h \cdot \mu_i^2 \right) \\ &= \frac{\textsc{ssq}(i)}{h} - \mu_i^2. \end{aligned} \tag{4}$$

Thus, it suffices to show how to compute $\textsc{sum}(i)$ and $\textsc{ssq}(i)$, for $1 \leq i \leq n-h+1$. We begin by computing the values of $\textsc{sum}(1)$ and $\textsc{ssq}(1)$ in $O(h)$ time by brute force. As we increment the value of $i$, we can update the quantities of interest in $O(1)$ time each. In particular, $\textsc{sum}(i+1) = \textsc{sum}(i) + (b_{i+h} - b_i)$ and $\textsc{ssq}(i+1) = \textsc{ssq}(i) + (b_{i+h}^2 - b_i^2)$. Thus, given the initial $O(n \log n)$ time to sort the points and the $O(h)$ time to compute $\textsc{sum}(1)$ and $\textsc{ssq}(1)$, the remainder of the computation

runs in constant time for each $i$. Therefore, the overall running time is $O(n \log n + h + (n-h+1)) = O(n \log n)$ time.

Next, we will show how to generalize this to compute the lower bound, $\Delta^-(C)$. Recall that the objective is to compute a lower bound on the LTS cost of *any* hyperplane $\boldsymbol{\beta}$ whose slope components lie within the $(d-1)$-dimensional hyperrectangle $[\boldsymbol{\beta}^-, \boldsymbol{\beta}^+]$. We will demonstrate that this can be reduced to the problem of solving a variant of the 1-dimensional LTS problem over a collection of $n$ intervals, which we call the *interval LTS problem*. We will present an $O(n \log n)$ time algorithm for this problem.

Our approach is to generalize the solution used in the case of the upper bound. Recall that $p_i = (\boldsymbol{x}_i, y_i)$ is the $i$th point of $P$, where $\boldsymbol{x}_i \in \mathbb{R}^{d-1}$ and $y_i \in \mathbb{R}$. For any hyperplane $\boldsymbol{\beta}$, recall the formula of Eq. (2) for the squared residual $r_i^2(\boldsymbol{\beta})$. We seek the value of $\beta_d$ that minimizes the sum of the $h$ smallest squared residuals subject to the constraint that $\boldsymbol{\beta}$'s slope coefficients lie within $[\boldsymbol{\beta}^-, \boldsymbol{\beta}^+]$. Although we do not know the exact values of the these slope coefficients, we do have bounds on them. Using the fact that $\beta_j^- \le \beta_j \le \beta_j^+$, for $1 \le j \le d-1$, we can derive upper and lower bounds on the second term in the squared residual formula (Eq. (2)) by a straightforward application of interval arithmetic. In particular, for $1 \le i \le n$,

$$b_i^+ = y_i - \sum_{j=1}^{d-1} \beta'_j x_{i,j}, \quad \text{where } \beta'_j = \begin{cases} \beta_j^- & \text{if } x_{i,j} \ge 0, \\ \beta_j^+ & \text{if } x_{i,j} < 0. \end{cases} \tag{5}$$

$$b_i^- = y_i - \sum_{j=1}^{d-1} \beta''_j x_{i,j}, \quad \text{where } \beta''_j = \begin{cases} \beta_j^+ & \text{if } x_{i,j} \ge 0, \\ \beta_j^- & \text{if } x_{i,j} < 0. \end{cases} \tag{6}$$

These quantities are analogous to the values $b_i$ used in the upper bound, subject to the uncertainty in the slope coefficients. It is easy to verify that $b_i^- \le b_i^+$, and for any $\boldsymbol{\beta}$ whose slope coefficients lie within $[\boldsymbol{\beta}^-, \boldsymbol{\beta}^+]$, the value of $r_i^2(\boldsymbol{\beta})$ is of the form $(\beta_d - b_i)^2$, for some $b_i \in [b_i^-, b_i^+]$. Analogous to the case of the upper bound computation, this is equivalent to saying that any hyperplane passing through $p_i$ along the direction of $\boldsymbol{\beta}$ intersects the $y$-axis at a point in the interval $[b_i^-, b_i^+]$ (see Fig. 5). For the purposes of obtaining a lower bound on the LTS cost, we may take $b_i$ to be the closest point in $[b_i^-, b_i^+]$ to the hypothesized intercept $\beta_d$. Thus, we have reduced our lower bound problem to computing the value of $\beta_d$ that minimizes the sum of the smallest $h$ squared distances to a collection of $n$ intervals. This is a generalization of the 1-dimensional LTS problem, but where instead of points, we now have intervals.

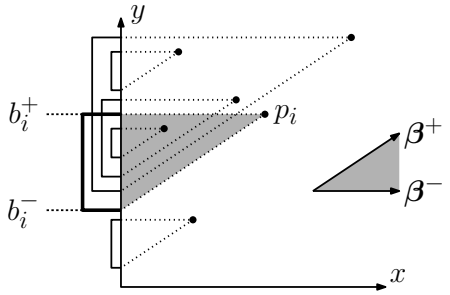



Fig. 5: Reducing the lower bound to the interval LTS problem.

More formally, we define the *interval LTS problem* as follows. Given a point $b$ and a closed interval $[b^-, b^+]$ define the *distance* from $b$ to this interval to be the minimum distance from $b$ to any point of the interval. (The distance will be zero if $b$ is contained within the interval.) Given a set $B$ of $n$ nonempty, closed intervals on the real line and a trimming parameter $h$, for any real $b$, define $\Delta_b(B, h)$ to be the sum of squared distances from $b$ to its closest $h$ intervals of $B$. The interval LTS problem is that of computing the minimum value of $\Delta_b(B, h)$, over all reals $b$, which we denote by $\Delta(B, h)$. Summarizing the analysis to this point, we obtain the following reduction.

**Lemma 3** *Consider an $n$-element point set $P$ in $\mathbb{R}^d$, a trimming parameter $h$, and a cell $C$ defined by the slope bounds $\boldsymbol{\beta}^-, \boldsymbol{\beta}^+ \in \mathbb{R}^{d-1}$. Let $B$ denote the set of $n$ intervals $[b_i^-, b_i^+]$ defined in Eqs. (5) and (6) above. Then for any $\boldsymbol{\beta} \in C$, $\Delta(B, h) \leq \Delta_{\boldsymbol{\beta}}(P, h)$.*

The rest of this section will be devoted to presenting an $O(n \log n)$ time algorithm that solves the interval LTS problem for a given set $B$ of $n$ intervals and a trimming parameter $h$. Our approach is a generalization of the algorithm for the upper-bound processing. First, we will identify a collection of $O(n)$ subsets of intervals of size $h$, such that the one with the lowest cost will be the solution to the interval LTS problem. Second, we will show how to compute the LTS cost of these subsets efficiently through an incremental approach. To avoid discussion of messy degenerate cases, we will make the general-position assumption that the interval endpoints of $B$ are distinct. This can always be achieved through an infinitesimal perturbation of the endpoint coordinates.

Recall that in the upper-bound computation, we computed the LTS cost of each $h$ consecutive points. In order to generalize this concept to the interval case, we sort the left endpoints $B$ in nondecreasing order and let $b_{[j]}^-$ denote the $j$th smallest left endpoint. Similarly, we sort the right endpoints and let $b_{[i]}^+$ denote the $i$th smallest right endpoint. Define $I_i$ to be the (possibly empty) interval $\left[b_{[i]}^+, b_{[i+h-1]}^-\right]$, and define $B_i$ to be the (possibly empty) subset of intervals of $B$ whose right endpoints are at least $b_{[i]}^+$, and whose left endpoints are at most $b_{[i+h-1]}^-$ (see Fig. 6). To avoid future ambiguity with the term "interval," we will refer to any interval formed from the endpoints of $B$ as a *span*, and each interval $I_i$ is called an *$h$-span*. We next prove a technical result about the sets $B_i$.

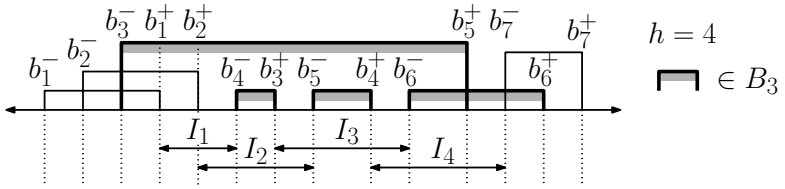



Fig. 6: The $h$-spans $I_i$ and the intervals in the subset $B_3$ are highlighted ($h = 4$).

**Lemma 4** *Let $(B, h)$ be an instance of the interval LTS problem, and let $B_i$ be the subsets of $B$ defined above, for $1 \leq i \leq n - h + 1$. If for any $i$, $b_{[i]}^+ > b_{[i+h-1]}^-$, then $\Delta(B, h) = 0$. Otherwise, each set $B_i$ contains exactly $h$ intervals of $B$.*

**Proof**: Observe that (by our general-position assumption) $i - 1$ intervals of $B$ lie strictly to the left of $b_{[i]}^+$, and $n - (i + h - 1)$ intervals lie strictly to the right of $b_{[i+h-1]}^-$. If $b_{[i]}^+ > b_{[i+h-1]}^-$, let $k$

denote the number of intervals of $B$ that contain the span $\left[b^-_{[i+h-1]}, b^+_{[i]}\right]$. Every interval of $B$ is either among the $i-1$ intervals to the left of $b^+_{[i]}$, and/or the $n-(i+h-1)$ intervals that lie to the right of $b^-_{[i+h-1]}$, or the $k$ that contain $\left[b^-_{[i+h-1]}, b^+_{[i]}\right]$. Therefore,

$$n \;\leq\; (i-1) + (n-(i+h-1)) + k \;\leq\; n-h+k,$$

from which we conclude that $k \geq h$. Thus, every point of the span $\left[b^-_{[i+h-1]}, b^+_{[i]}\right]$ is contained within $h$ intervals of $B$. This implies that the distance from any such point to all of its $h$ closest intervals is zero. Therefore, $\Delta(B_i, h) = 0$, and so $\Delta(B, h) = 0$.

On the other hand, if $b^+_{[i]} \leq b^-_{[i+h-1]}$, then the $i-1$ intervals of $B$ to the left of $b^+_{[i]}$ are disjoint from the $n-(i+h-1)$ intervals to the right of $b^-_{[i+h-1]}$. Since the remaining $n$ intervals of $B$ are in $B_i$, it follows that $|B_i| = n-(i-1)-(n-(i+h-1)) = h$, as desired. □

Henceforth, let us assume that $b^+_{[i]} > b^-_{[i+h-1]}$ for all $i$, since if we ever detect that this is not the case, we may immediately report that $\Delta(B, h)$ is zero and terminate the algorithm. The subsets $B_i$ were chosen to be the "tightest" subsets of $B$ having $h$ elements. Next, we prove formally that for the purposes of solving the interval LTS problem, it suffices to consider just these subsets. For $1 \leq i \leq n-h+1$, define $\Delta(B_i, h)$ to be the LTS cost of the interval LTS problem using only the intervals of $B_i$.

**Lemma 5** *Given an instance $(B, h)$ of the interval LTS problem, $\Delta(B, h) = \Delta(B_i, h)$ for some $i$, where $1 \leq i \leq n-h+1$.*

**Proof**: Let $B'$ denote the subset of $B$ of size $h$ that achieves the minimum LTS cost. Suppose towards a contradiction that $\Delta(B', h)$ is strictly smaller than $\Delta(B_i, h)$ for all $i$. Let $b^+_{[i]}$ and $b^-_{[j]}$ denote the leftmost right endpoint and rightmost left endpoint in $B'$, respectively. Since the span from $b^+_{[i]}$ to $b^-_{[j]}$ must overlap at least $h$ intervals of $B$, by the same reasoning as in Lemma 4, we have $j \geq i+h-1$. Further, we may assume that $j > i+h-1$, for otherwise $B'$ would be equal to $B_i$. Thus, $B'$ spans at least $h+1$ intervals, which implies that there exists an interval, call it $I$, that is in $B_i$ but not in $B'$ (see Fig. 7(a)).

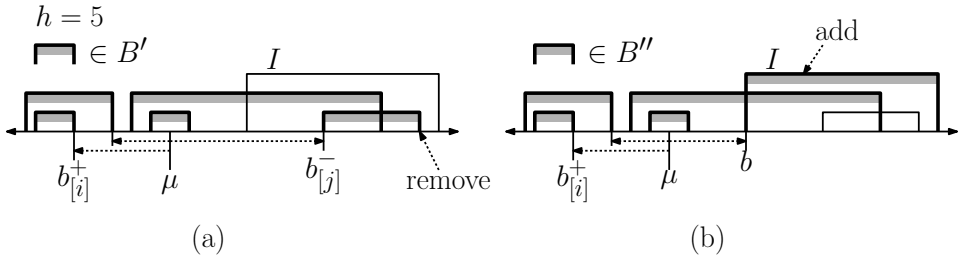



Fig. 7: Proof of Lemma 5. By replacing the interval whose left endpoint is anchored to $b^-[j]$ with the one anchored to $b$, we decrease the LTS cost.

First, we assert that $b^+_{[i]} < b^-_{[j]}$. To see why, observe that otherwise, $b^+_{[i]}$ is contained within $h+1$ intervals (by the same reasoning as in Lemma 4), implying that $\Delta(B_i, h) = 0$, which would violate our hypothesis that none of the $B_i$'s is optimal. Given this assertion, let $\mu$ denote the point that

minimizes the sum of squared distances to the intervals of $B'$ (see Fig. 7(a)). Clearly, $b^+_{[i]} \le \mu \le b^-_{[j]}$. By left-right symmetry, we may assume without loss of generality that $\mu$ is closer to $b^+_{[i]}$ than it is to $b^-_{[j]}$. (More specifically, if we negate all the interval endpoints, we apply the proof with the roles of $b^+_{[i]}$ and $b^-_{[j]}$ reversed.) Because $b^-_{[j]}$ is the rightmost left endpoint in $B'$, the distance from $\mu$ to $b^-_{[j]}$ is the largest among all the intervals of $B$ in whose closest endpoint to $\mu$ lies in the span $[b^+_{[i]}, b^-_{[j]}]$, and in particular this is true for the interval $I$. Consider the set $B''$ formed by taking $B'$ and replacing the interval anchored at $b^-_{[j]}$ with $I$ (see Fig. 7). The LTS cost of $B''$ is clearly smaller than the LTS cost of $B'$, which contradicts our hypothesis that $B'$ achieves the minimum LTS cost. □

From the above lemma, it follows that in order to solve the interval LTS problem, it suffices to compute the LTS cost of each of the subsets $B_i$, for $1 \le i \le n - h + 1$. Just as we did in the upper bound computation, we will adopt an incremental approach. Recall that $\Delta(B_i, h)$ denotes the interval LTS cost of $B_i$, and let $\mu_i$ denote the point that minimizes the sum of squared distances from itself to all of the intervals of $B_i$. Our approach will be to compute the initial values, $\mu_1$ and $\Delta(B_1, h)$, explicitly by brute force in $O(h)$ time. Then, for $i$ running from 2 up to $n - h + 1$, we will show how to update the subsequent values $\mu_i$ and $\Delta(B_i, h)$, each in constant time.

In the upper bound algorithm, we observed that the value that minimizes the sum of squared distances to a set of numbers is just the mean of the set. However, generalizing this simple observations to a set of intervals is less obvious. Given an arbitrary interval $[b^-, b^+]$ of $B$, the function that maps any point $x$ on the real line to its squared distance to this interval is clearly convex. (It is not strictly convex because the function is zero for all points that lie within the interval.) Since $\Delta(B_i, h)$ is the minimum of the sum of $h$ such functions, and by our assumption that no $h$ intervals of $B$ contain in a single point, it follows that this function is strictly convex, and hence $\mu_i$ is its unique local minimum. Given an initial estimate on the location of $\mu_i$, we can exploit convexity to determine the direction in which to move to locate it. As the algorithm transitions from $B_{i-1}$ to $B_i$, we remove one interval from the left (the one anchored at $b^+_{[i-1]}$) and add one interval to the right (the one anchored at $b^-_{[i+h-1]}$). Since $b^-_{[i+h-1]} > b^+_{[i-1]}$, we have $\mu_i \ge \mu_{i-1}$. Thus, once $\mu_{i-1}$ is computed, the search for $\mu_i$ can proceed monotonically to the right.

To make this more formal, let $\{b_1, \ldots, b_{2n}\}$ denote the sorted sequence of the interval endpoints of $B$ in nondecreasing order. This implicitly defines $2n - 1$ intervals of the form $[b_{k-1}, b_k]$, for $1 < k \le 2n$ where $\mu_i$ might lie. Again, to avoid ambiguity with the term "interval," we refer to these intervals as *cells* (see Fig. 8(a)). All the points within the interior of any given cell have the property that they lie to the left/right/contained within the same intervals of $B$. Our algorithm operates by maintaining three indices, $i$, $j$, and $k$. We will set $j = i + h - 1$ so that $I_i = \left[b^+_{[i]}, b^-_{[j]}\right]$ is the current $h$-span of interest. The index $k$ specifies the *current cell*, $[b_k, b_{k+1}]$, which represents the current candidate for where $\mu_i$ is located. The algorithm repeatedly increases $k$ as long as $\mu_i$ is determined to lie to the right of the current cell.

How do we determine whether the current cell contains $\mu_i$? First, we classify the every interval of $B_i$ as lying to the left, right, or containing the cell's interior. The intervals of $B_i$ that contain the cell's interior contribute zero to the squared distance, and so they may be ignored. For those intervals that lie entirely to the cell's left (resp., right), the interval's right (resp., left) endpoint determines the squared distance (see Fig. 8(b)). Let us call this endpoint the *relevant endpoint* of the associated interval. As in the computation of the upper bound, we will maintain two quantities

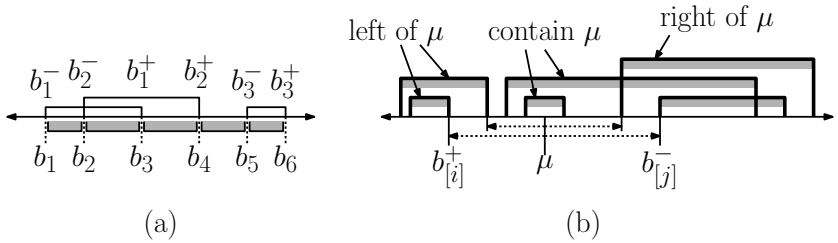



Fig. 8: Cells and contributions: (a) The sorted endpoints $\langle b_1, \ldots, b_{2n} \rangle$ and the associated cells (shaded), (b) the mean $\mu_i$ and the distance contributions from the intervals of $B_i$.

SUM and SSQ, which will be updated throughout the algorithm. The first stores the sum of the relevant endpoints and the second stores the sum of square of the relevant endpoints. As in Eqs. (3) and (4), the value $\mu_i$ and the LTS cost $\Delta(B_i, h) = \sigma_i^2$ will be derived from these quantities.

After $k$ is incremented, we are now either entering an interval of $B$ (if $b_k$ is a left endpoint) or leaving an interval (if $b_k$ is a right endpoint). In the first case the interval we are entering was contributing its left endpoint to (SUM and SSQ) and is now not making a contribution. In the second case, the interval we are leaving was not contributing and is now contributing its right endpoint. The algorithm maintains $\Delta_{\text{best}}$, which stores the minimum LTS cost of any $B_i$ encountered thus far. We can now present the complete algorithm.

(1) [Initializations:]

  (a) [Sort:] Sort the left endpoints of $B$ as $\langle b_{[1]}^-, \ldots, b_{[n]}^- \rangle$, the right endpoints as $\langle b_{[1]}^+, \ldots, b_{[n]}^+ \rangle$, and all the endpoints as $\langle b_1, \ldots, b_{2n} \rangle$.

  (b) [Initialize costs:] Set $\text{SUM} \leftarrow \sum_{1 \le j \le h} b_{[j]}^-$, $\text{SSQ} \leftarrow \sum_{1 \le j \le h} (b_{[j]}^-)^2$, and $\Delta_{\text{best}} \leftarrow +\infty$.

  (c) [Initialize indices:] Let $i \leftarrow k \leftarrow 1$ and $j \leftarrow h$.

(2) While $i \le n - h - 1$ do:

  (a) [Check for $h$-fold interval overlap:] If $b^+[i] \ge b^-[j]$, set $\Delta_{\text{best}} \leftarrow 0$ and return.

  (b) [Update $\mu$:] Set $\mu \leftarrow \text{SUM}/h$. (Recall Eq. (3).)

  (c) [Find the cell that contains $\mu$:] While $b_{k+1} < \mu$ do:

    (i) [Advance to the next cell:] Set $k \leftarrow k + 1$.

    (ii) [Entering an interval?] If $b_k$ is the left endpoint of some interval $[b^-, b^+]$ of $B$, set $\text{SUM} \leftarrow \text{SUM} - b^-$ and $\text{SSQ} \leftarrow \text{SSQ} - (b^-)^2$. (This interval was contributing its left endpoint to the LTS cost but is no longer contributing.)

    (iii) [Exiting an interval?] Otherwise, $b_k$ is the right endpoint of some interval $[b^-, b^+]$ of $B$. Set $\text{SUM} \leftarrow \text{SUM} + b^+$ and $\text{SSQ} \leftarrow \text{SSQ} + (b^+)^2$. (This interval was not contributing to the LTS cost and is now contributing its right endpoint.)

    (iv) [Update $\mu$:] Set $\mu \leftarrow \text{SUM}/h$.

  (d) [Update LTS cost:] Set $\sigma^2 \leftarrow \text{SSQ}/h - \mu^2$ (see Eq. (4)). If $\sigma^2 < \Delta_{\text{best}}$ then $\Delta_{\text{best}} \leftarrow \sigma^2$.

  (e) [Remove contribution of $b^+[i]$:] $\text{SUM} \leftarrow \text{SUM} - b^+[i]$ and $\text{SSQ} \leftarrow \text{SSQ} - (b^+[i])^2$.

  (f) [Advance to next subset:] Set $i \leftarrow i + 1$ and $j \leftarrow j + 1$.

(g) [Add contribution of $b^-[j]$:] SUM $\leftarrow$ SUM $+ b^-[j]$ and SSQ $\leftarrow$ SSQ $+ (b^-[j])^2$.

(3) Return $\Delta_{\text{best}}$ as the final LTS cost

The algorithm's correctness has already been justified. To establish the algorithm's running time, observe that Step (1a) takes $O(n \log n)$ time, and Step (1b) takes $O(h)$ time. All the other steps take only constant time. Each time the loop of Step (2c) is executed, the value of $k$ is increased. Since $k$ cannot go beyond the last cell (since clearly $\mu \leq b_{2n}$), the total number of iterations of this loop throughout the entire algorithm is at most $2n$. Since the loop of Step (2) is repeated at most $n - h + 1$ times, the overall running time is $O(n \log n + h + 2n + (n - h - 1)) = O(n \log n)$. In summary we have the following result, which together with Lemma 3 proves that the LTS lower bound can be computed for each cell in $O(n \log n)$ time.

**Theorem 2** *Given a collection $B$ of $n$ intervals on the real line and a trimming parameter $h$, it is possible to solve the interval LTS problem on $B$ in $O(n \log n)$ time.*

# 5 Empirical Analysis

As mentioned in the introduction, while Fast-LTS is efficient in practice, it does not provide guarantees on the accuracy of its output. In contrast, on termination, Adaptive-LTS provides guarantees the accuracy of its results. In this section we show that when presented with inputs of low dimension (that is, having a small number of independent variables), Adaptive-LTS performs quite efficiently and can often provide good bounds on the accuracy of the final result. We will demonstrate this empirically on a number of synthetically generated distributions and one actual data set, which was derived from an application in astronomy.

Before presenting our results, let us remark on the structure that is common to all our experiments. We implemented both Rouseeuw and van Driessen's Fast-LTS and our Adaptive-LTS in C++ (compiled by g++ 4.5.2 with optimization level "-O3" and running on a dual-core Intel Pentium-D processor under Ubuntu Linux 11.04).

Both Fast-LTS and Adaptive-LTS operate in a series of stages. Each stage of our implementation of Fast-LTS consists of a single random elemental fit followed by two C-steps. (This is the same number used by Rousseeuw and van Driessen in [21].) After each stage we output the LTS cost of the resulting fit. Each experiment consisted of 500 such stages (the default number of stages for FastLTS). Our plots for Fast-LTS show, for each stage, the LTS cost of the current fit and the minimum LTS cost among all the fits seen so far. After these stages, Fast-LTS takes the ten best fits and runs a large number of C-steps on each. Because we are primarily interested in the convergence properties of both algorithms, we have omitted these final C-steps, but it could easily have been added to our implementation.

For Adaptive-LTS, each stage consists of the processing of a single cell. Recall that this involves sampling a hyperplane from the current cell to which two C-steps are then applied. In addition, a lower bound is computed for the current cell. After each stage, we output the LTS-cost of the current fitting hyperplane and the minimum lower bound among all active cells at this point of the search. Although Adaptive-LTS has a termination condition, for the sake of comparison with FastLTS we disabled the termination condition and instead ran the algorithm for 500 stages. Our plots for Adaptive-LTS show, for each stage, the LTS cost of the current fit, the minimum LTS cost among all the fits seen so far, and the minimum lower bound at this point of the algorithm.

### 5.1 Synthetically Generated Data

For these experiments, we generated a data set consisting of 1000 points in $\mathbb{R}^2$ and $\mathbb{R}^3$. In order to simulate point sets that contain outliers, each data set consisted of the union of points drawn from two or more distributions. One of these, the *inlier distribution*, contains points that lie close to a randomly generated hyperplane. The others, the *outlier distributions*, contain points that are drawn from some other distribution. We have aimed to generate data sets with various degrees of numerical conditioning, ranging from nice to nasty.

**hyperplane+uniform (2-dimensional):** Our first experiment involved a 2-dimensional, well-conditioned distribution, consisting of 1000 points. First, a line of the form $y = \beta_1 x + \beta_2$ was generated, where $\beta_1$ was sampled uniformly from $[-0.25, +0.25]$, and $\beta_2$ was sampled uniformly from $[-1, +1]$. Next, 550 points (55% inliers) were generated with the $x$-coordinate sampled uniformly from $[-1, +1]$, and the $y$-coordinates generated by computing the value of the line at these $x$-coordinates and adding a Gaussian deviate with mean 0 and standard deviation 0.01. The remaining 450 points (45% outliers) were sampled uniformly from the square $[-1, +1]^2$. Because the inliers were drawn from a hyperplane (actually a line in this case) and the outliers were drawn from a uniform distribution, we call this distribution HYPERPLANE+UNIFORM (see Fig. 9).

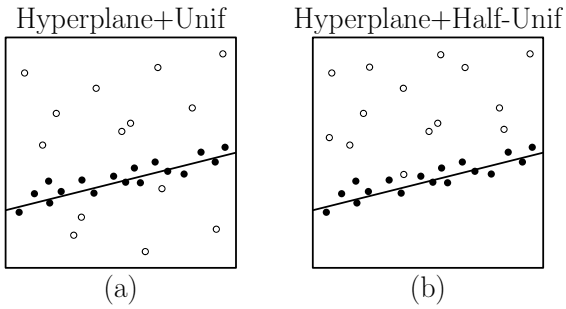
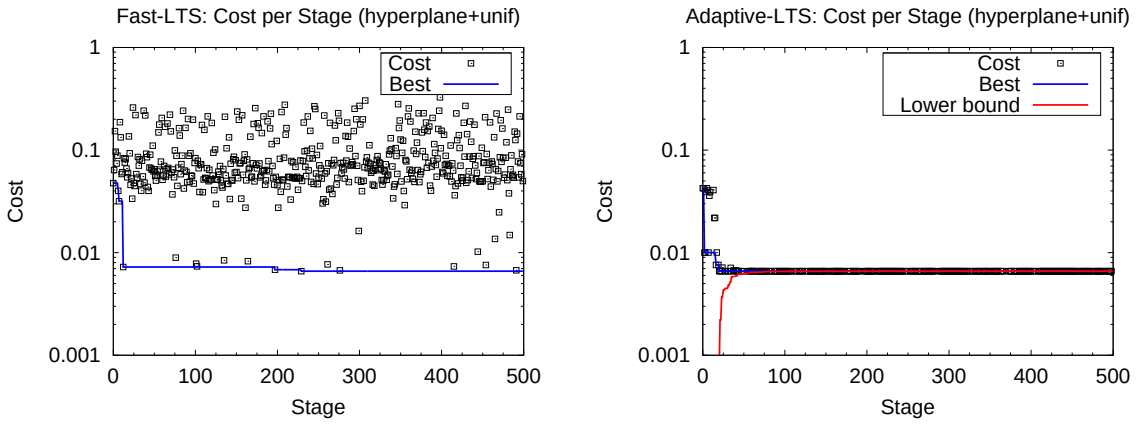



Fig. 9: The HYPERPLANE+UNIFORM and HYPERPLANE+HALF-UNIF distributions.

We ran both Fast-LTS and Adaptive-LTS on the resulting data set for 500 stages. Although in most of our experiments, we set the trimming parameter close to 50%, to make this instance particularly easy to solve, we ran both algorithms with a relatively low trimming parameter of $h = 0.1$ (that is, 10% outliers). In Fig. 10(a) and (b) we show the results for Fast-LTS and Adaptive-LTS, respectively, where the $x$-axis indicates the stage number and the $y$-axis indicates the LTS-cost. (Note that the $y$-axis is on a logarithmic scale.) Each (small square) point of each plot indicates the LTS-cost of the sampled hyperplane. The upper (blue or "Best") curve indicates the minimum LTS-cost of any hyperplane seen so far. The lower (red or "Lower bound") of Fig. 10(b) indicates the minimum lower bound of any active cell. The optimum LTS-cost lies somewhere between these two curves, and therefore, the ratio between the current best and current lower bound limits the maximum relative error that would result by terminating the algorithm and outputting the current best fit.

As mentioned earlier, this is a well-conditioned distribution, and therefore it not surprising that both algorithms rapidly converge to a solution that is nearly optimal. Given only information on the best LTS cost, a user of Fast-LTS would not know when to terminate the search. In contrast, a user of Adaptive-LTS would be able to make intelligent decisions as to when to terminate. For




(a) (b)

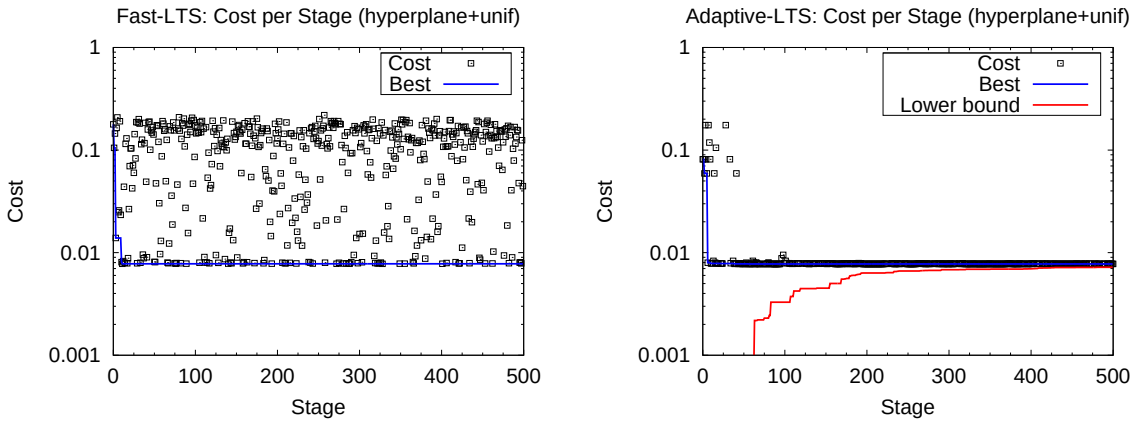
Fig. 10: LTS cost versus stage number for the HYPERPLANE+UNIFORM distribution ($d = 2$, $h = 0.1$) for (a): Fast-LTS and (b): Adaptive-LTS.

example, after just 75 iterations, the gap between the best upper and lower bounds is smaller than 1%. Thus, terminating the algorithm at this point would imply an approximate error of at most 1%. It is also interesting to note that Fast-LTS generates elemental fits at random, and hence the pattern of plotted points remains consistently random throughout all 500 stages. In contrast, the sampled hyperplanes generated by Adaptive-LTS shows signs of converging to a subregion of the configuration space where the most promising solutions reside.

**hyperplane+uniform (3-dimensional):** Our second experiment involved the same distribution, but in $\mathbb{R}^3$ and using a lareger value of $h$. First, a plane of the form $y = \beta_1 x_1 + \beta_2 x_2 + \beta_3$ was generated, where $\beta_1$ and $\beta_2$ were sampled uniformly from $[-0.25, +0.25]$, and $\beta_2$ was sampled uniformly from $[-1, +1]$. As before, 550 points (55% inliers) were generated with the $x_1$- and $x_2$-coordinates sampled uniformly from $[-1, +1]$, and the $y$-coordinates generated by computing the value of the plane at these $x$-coordinates and adding a Gaussian deviate with mean 0 and standard deviation 0.01. The remaining 450 points (45% outliers) were sampled uniformly from the cube $[-1, +1]^3$.

We ran both Fast-LTS and Adaptive-LTS on the data set for 500 stages with $h = 0.5$. In Fig. 11(a) and (b) we show the results for Fast-LTS and Adaptive-LTS, respectively. Again, both algorithms converge rapidly to a solution that is nearly optimal. However, as in the previous experiment, a user of Adaptive-LTS could use the ratio between the best upper and lower bounds to limit the maximum error in the LTS cost. For example, after 200 stages the relative error is within 10%, and after 300 stages it is within 5%. Observe that the rate of convergence is slower than in the earlier 2-dimensional experiment. This is likely due to the increase in the dimension (which results in a larger search space to be pruned) and the increase in the trimming parameter (which forces the algorithm to find a hyperplane that fits a larger fraction of the data).

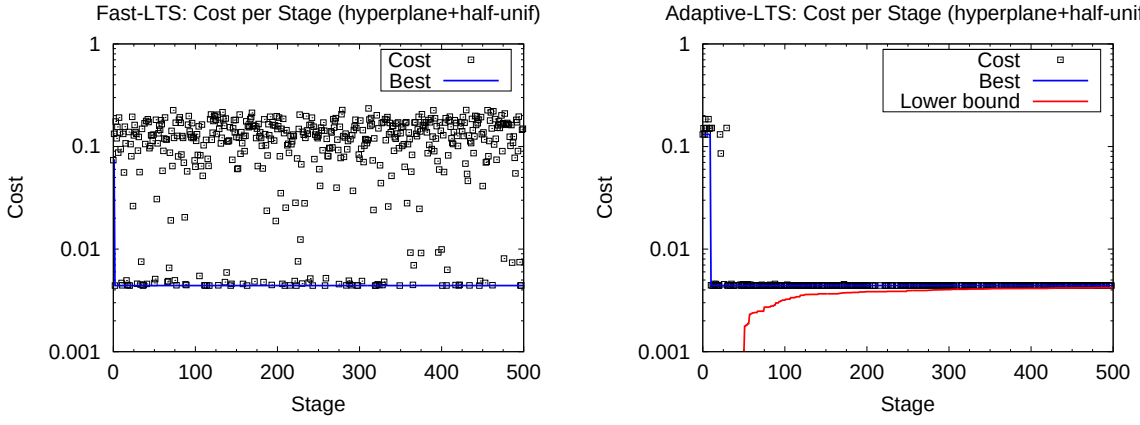
**hyperplane+half-unif:** This involved a data set consisting of 1000 points in $\mathbb{R}^3$, in which the outlier distribution was more skewed. The inliers were generated in the same manner as in the




(a) (b)

Fig. 11: LTS cost versus stage number for the HYPERPLANE+UNIFORM distribution ($d = 3$, $h = 0.5$) for (a): Fast-LTS and (b): Adaptive-LTS.

previous experiment. The outliers were sampled uniformly from the portion of the 3-dimensional hypercube $[-1, +1]^3$ that lies above the plane (see Fig. 9(b)). As before, we ran both Fast-LTS and Adaptive-LTS on the data set for 500 stages with $h = 0.5$. In Fig. 12(a) and (b) we show the results for Fast-LTS and Adaptive-LTS, respectively. The convergence properties of the two algorithms is similar to the previous experiment. After 275 stages of Adaptive-LTS, the relative error is within 10% and after 500 stages it is within 5%.




(a) (b)

Fig. 12: LTS cost versus stage number for the HYPERPLANE+HALF-UNIF distribution ($d = 3$, $h = 0.5$) for (a): Fast-LTS and (b): Adaptive-LTS.

**flat+sphere:** For our final synthetic experiment, we designed a rather pathological distribution with the intention of challenging both algorithms. The inlier distribution was chosen so that almost all the inliers are degenerate (thus resulting in many unstable elemental fits) and the remaining

few inliers are a huge distance away (thus penalizing any inaccurate elemental fits).

The point set consisted of 1000 points in $\mathbb{R}^3$ with 500 inliers and 500 outliers. Let $S$ be a large sphere centered at the origin with radius 10,000, let $\boldsymbol{\beta}$ be plane passing through the origin, and let $\ell$ be a line passing through the origin that lies on $\boldsymbol{\beta}$ (see Fig. 13). The inliers were sampled from two distributions. First, 490 *local points* were generated uniformly along a line segment of length two centered at the origin and lying on $\ell$, where each point was perturbed by a randomly oriented vector whose length was drawn from a Gaussian distribution with mean zero and standard deviation 0.10. The remaining 10 inliers, called *leverage points*, were sampled uniformly from the equatorial circle where $\boldsymbol{\beta}$ intersects $S$. We set the trimming parameter to $h = 0.5$, implying that exactly 500 points are used in the computation of the LTS cost. Finally, the outliers consisted of 500 points that were sampled uniformly from $S$.

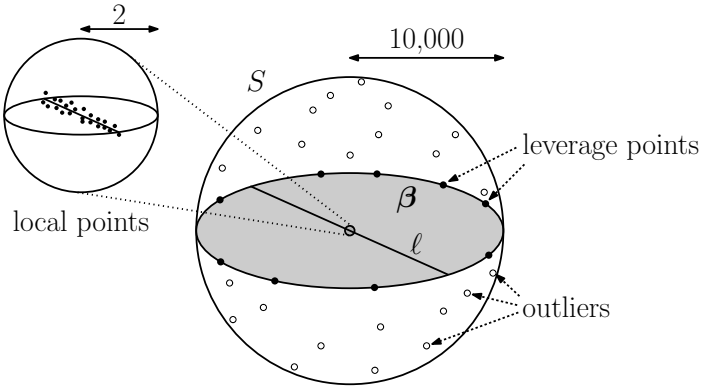



Fig. 13: The FLAT+SPHERE distribution.

To see why this distribution is particularly nasty, observe that barring lucky coincidences, the only way to obtain a low-cost LTS fit is to combine all the local points with all the leverage points to obtain a fit that lies very close to $\boldsymbol{\beta}$. Thus, any elemental fit that involves even one outlier is quite unlikely to provide a good fit. Because of their degenerate arrangement along $\ell$, any random sample of inliers that fails to include at least one leverage point is unlikely to lie close to $\boldsymbol{\beta}$. The probability of sampling three inliers such that at least one is a leverage point is roughly $(500/1000)^3 \cdot 10/500 = 0.025$. Note that by decreasing the fraction of leverage points relative to the total number of inliers, we can make this probability of a useful elemental fit arbitrarily small. This point set clearly fails both of the criteria given in Section 2, since the vast majority of inliers are degenerately positioned, and there are a nontrivial number of leverage points.

Our experiments bear out the problems of this poor numerical conditioning (see Fig. 14). Both algorithms cast about rather aimlessly until somewhere between stages 200–300, where they each succeed in sampling a suitable fit. This experiment demonstrates the principal advantage of Adaptive-LTS over Fast-LTS. A user of Fast-LTS who saw only the results of the first 200 iterations might be tempted to think that the algorithm had converged to an optimal LTS cost of roughly 3.8, rather than the final cost of roughly 1.02. Terminating the algorithm prior to stage 200 would have resulted in a relative error of over 300%. In contrast, a user of Adaptive-LTS would know by stage 500 that the relative error in the final LTS cost is less than 50%. (Although it is not evident from our plots, after roughly 1500 stages the lower bound increases to the point that it can be inferred that the approximation error is less than 10%, and after roughly 2000 stages it is less than

5%. Note that the convergence is due to improvements in the lower bound. The upper bound did not change in later stages.)

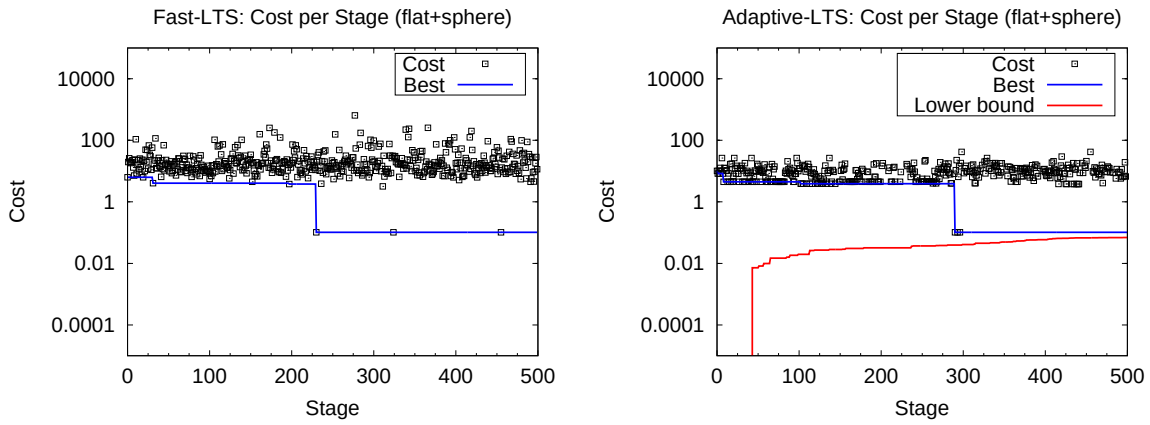



Fig. 14: LTS cost versus stage number for the FLAT+SPHERE distribution ($d = 4$, $h = 0.5$) for (a): Fast-LTS and (b): Adaptive-LTS.

**Execution Times:** So far we have analyzed only the LTS costs generated by the two algorithms. An important question how the performance of Adaptive-LTS compares with Fast-LTS. In Table 1 we summarize the execution times and LTS costs for both of these algorithms for 500 stages. In all cases, both algorithms generated essentially the same fitting hyperplane and, hence, achieved the same LTS cost. While the execution time of Adaptive-LTS was consistently higher than that of Fast-LTS, the difference is not very large. As can be seen from the table, there is only one instance (HYPERPLANE+UNIFORM in 2D) where Adaptive-LTS required more than twice the execution time of Fast-LTS, and even in this case, the ratio between the two execution times is only slightly greater than three.

Table 1: Summary of execution times and LTS costs for the synthetic experiments for 500 iterations of each algorithm.

| Experiment | | | Fast-LTS | | Adaptive-LTS | |
|---|---|---|---|---|---|---|
| Distribution | $d$ | $h$ | Time (sec) | LTS Cost | Time (sec) | LTS Cost |
| HYPERPLANE+UNIFORM | 2 | 0.1 | 0.14 | 0.00660 | 0.44 | 0.00660 |
| HYPERPLANE+UNIFORM | 3 | 0.5 | 0.47 | 0.00776 | 0.83 | 0.00776 |
| HYPERPLANE+HALF-UNIF | 3 | 0.5 | 0.48 | 0.00440 | 0.73 | 0.00440 |
| FLAT+SPHERE | 3 | 0.5 | 0.81 | 0.10220 | 1.11 | 0.10220 |

### 5.2 DPOSS Data Set

Our second experiment involved a set of points from the Digitized Palomar Sky Survey (DPOSS), from the the California Institute of Technology [2]. The data set consisted of 132,402 points in $\mathbb{R}^6$, from which we sampled 10,000 at random for each of our experiments. Of the six coordinates, we considered the problem of estimating the first coordinate MAperF, which measures the object's overall brightness in the F spectral band, as a function of one or more of the coordinates csfF, csfJ, csfN, which measure the stellar quality of the object in the spectral bands F, J, and N, respectively. Fig. 15 shows a scatter plot of MAperF versus csfF, and also shows the LTS fit (which was obtained by both our algorithm and FastLTS) and an ordinary least square (OLS) fit. We used the same experimental setup as in the synthetic experiments, by running both FastLTS and AdaptiveLTS for 500 stages.

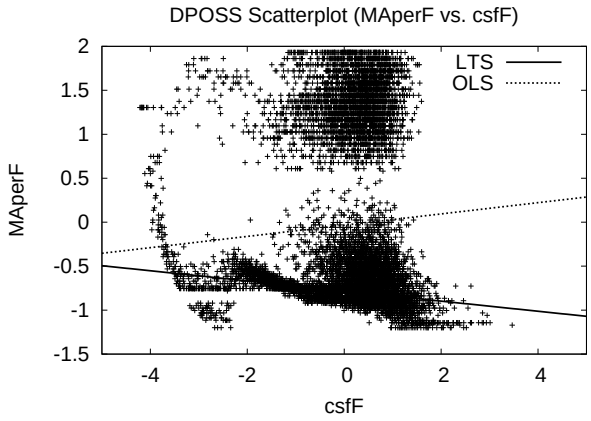
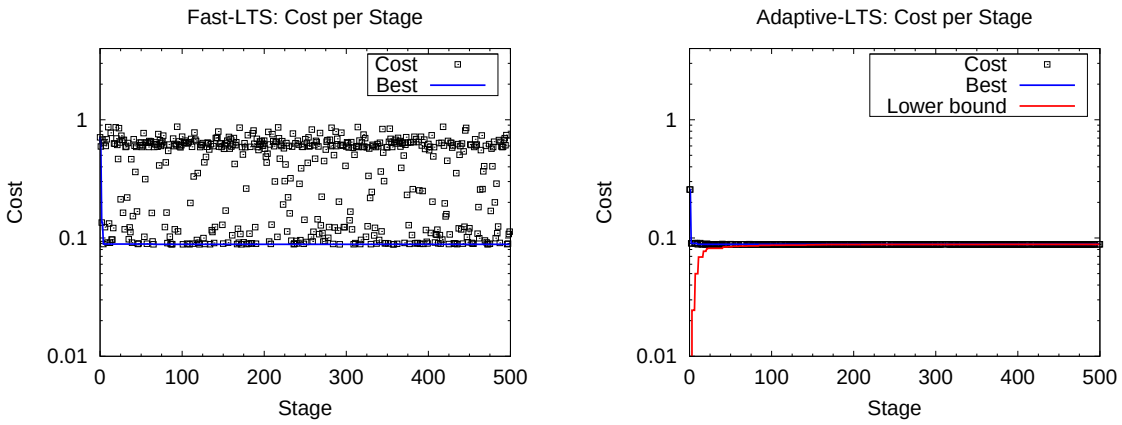



Fig. 15: The DPOSS data set (MAperF vs. csfF).

In the first experiment, we considered a 2-dimensional instance of estimating MAperF as a function of csfF (see Fig. 16). Both algorithms rapidly converged to essentially the same solution (shown in Fig. 15). Because it provides a lower bound, a user of AdaptiveLTS would know after 100 stages that the relative error in the LTS cost of this solution is less than 1%.

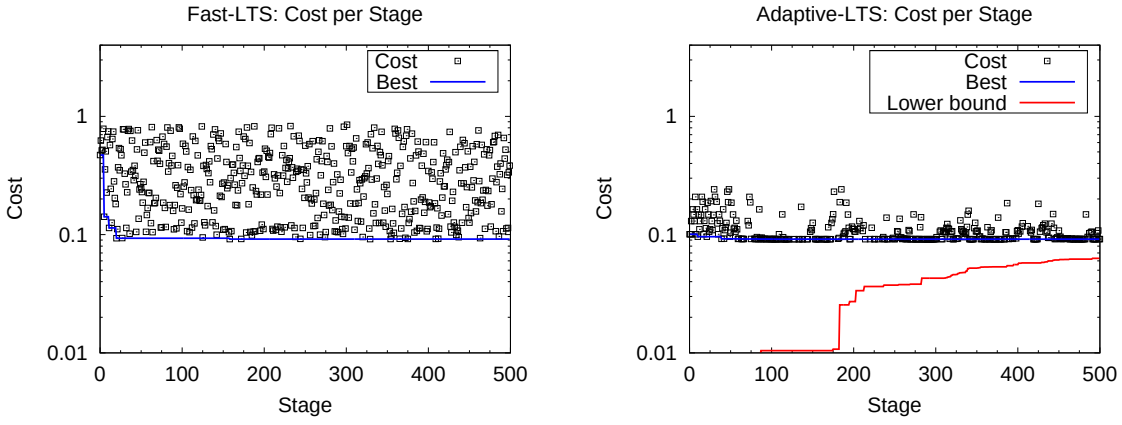
Our next experiment considered a 3-dimensional instance of estimating MAperF as a function of both csfF and csfJ (see Fig. 17). Again, both algorithms rapidly converged to essentially the same solution. Compared to the 2-dimensional example, it takes Adaptive LTS substantially longer to obtain a good lower bound. This is in part because the solution space is of higher dimension, and hence more time is required to search the space. After 500 stages the relative error in the LTS cost is less than 40%. (Although it is not evident from the plots, after roughly 800 stages the lower bound increases to the point that it can be inferred that the approximation error is less than 10%, and after roughly 1400 stages it is less than 5%. As in the synthetic experiments, the convergence is due to improvements in the lower bound. The upper bound did not change in later stages.)

Our final experiment demonstrates one of the practical limitations of AdaptiveLTS. We considered a 4-dimensional instance of estimating MAperF as a function of csfF, csfJ, and csfN (see Fig. 18). Again, both algorithms rapidly converged to essentially the same solution. However, the higher dimensional solution space takes much more time to search. After 500 stages, the lower bound is still zero and hence does not even appear on the plot. Even after 10,000 stages the lower




(a) (b)

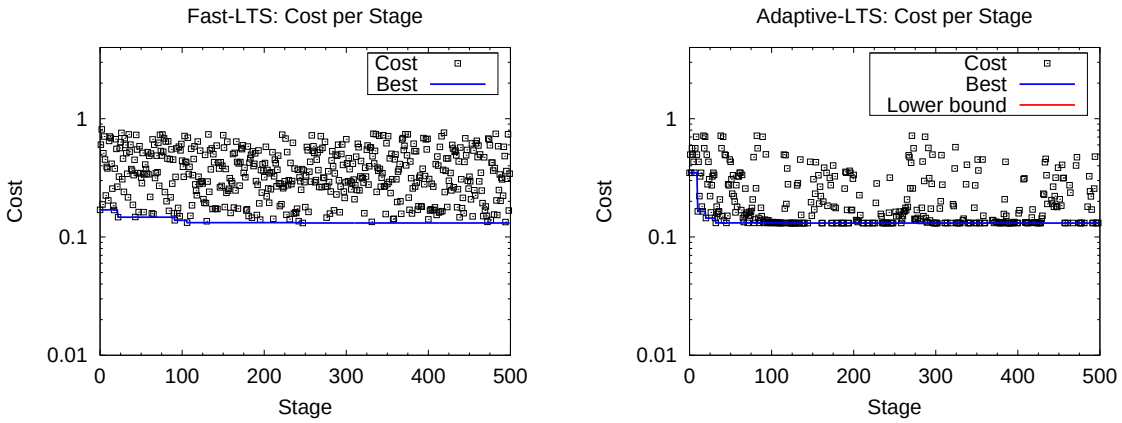
Fig. 16: LTS cost versus stage number for estimating MAperF as function of csfF in the DPOSS data set ($d = 2$) for (a): Fast-LTS and (b): Adaptive-LTS.




(a) (b)

Fig. 17: LTS cost versus stage number for the DPOSS data set ($d = 3$) for (a): Fast-LTS and (b): Adaptive-LTS.

bound provides an approximation error bound of only about 25%. Thus, even though both Fast-LTS and Adaptive-LTS find the same solution fairly soon, it takes much more time to establish a guarantee on the quality of this solution.




(a) (b)

Fig. 18: LTS cost versus stage number for the DPOSS data set ($d = 4$) for (a): Fast-LTS and (b): Adaptive-LTS.

As in our synthetic experiments, we also compared the relative performance of Adaptive-LTS and Fast-LTS. In Table 2 we summarize the execution times and LTS costs for both of these algorithms for 500 stages. Both algorithms generated essentially the same fitting hyperplane and, hence achieved the same LTS cost. Because of its higher overhead, Adaptive-LTS did require more execution time for the same number of stages, but in all instances the running time was less than 50% greater than Fast-LTS.

Table 2: Summary of execution times and LTS costs for the DPOSS experiments for 500 iterations of each algorithm.

| DPOSS | | Fast-LTS | | Adaptive-LTS | |
|---|---|---|---|---|---|
| $d$ | $h$ | Time (sec) | LTS Cost | Time (sec) | LTS Cost |
| 2 | 0.5 | 5.02 | 0.0884 | 7.31 | 0.0884 |
| 3 | 0.5 | 5.77 | 0.0915 | 7.40 | 0.0915 |
| 4 | 0.5 | 7.72 | 0.1308 | 8.54 | 0.1311 |

## 6 Concluding Remarks

[Finish this.] We would like to thank Peter Rousseeuw for providing us with the DPOSS data set.

## References


[1] T. Bernholt. Computing the least median of squares estimator in time $O(n^d)$. In *Proc. Intl. Conf. on Computational Science and Its Applications*, volume 3480 of *Springer LNCS*, pages 697–706, Berlin, 2005. Springer-Verlag.

[2] S. G. Djorgovski, R. R. Gal, S. C. Odewahn, de R. R. Carvalho, R. Brunner, G. Longo, and R. Scaramella. The Palomar digital sky survey (DPOSS). In *Wide Field Surveys in Cosmology (14th IAP meeting)*, page 89, Paris, 1998. Editions Frontieres.

[3] H. Edelsbrunner and D. L. Souvaine. Computing median-of-squares regression lines and guided topological sweep. *Journal of the American Statistical Association*, 85:115–119, 1990.

[4] J. Erickson, S. Har-Peled, and D. M. Mount. On the least median square problem. *Discrete Comput. Geom.*, 36:593–607, 2006.

[5] G. D. Da Fonseca. Fitting flats to points with outliers. Unpublished manuscript, 2010.

[6] S. H. Friedberg, A. J. Insel, and L. E. Spence. *Linear Algebra*. Prentice Hall, 4th edition, 2002.

[7] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[8] M. Hagedoorn and R. C. Veltkamp. Reliable and efficient pattern matching using an affine invariant metric. *International Journal of Computer Vision*, 31:103–115, 1999.

[9] S. Har-Peled. How to get close to the median shape. *Comput. Geom. Theory Appl.*, 36:39–51, 2007.

[10] D. M. Hawkins. The feasible solution algorithm for least trimmed squares regression. *Computational Statistics & Data Analysis*, 17:185–196, 1994.

[11] O. Hössjer. Exact computation of the least trimmed squares estimate in simple linear regression. *Computational Statistics & Data Analysis*, 19:265–282, 1995.

[12] D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the Hausdorff distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 705–706, New York, June 1993.

[13] D. M. Mount, N. S. Netanyahu, and J. Le Moigne. Efficient algorithms for robust point pattern matching. *Pattern Recognition*, 32:17–38, 1999.

[14] D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. Quantile approximation for robust statistical estimation and $k$-enclosing problems. *Internat. J. Comput. Geom. Appl.*, 10:593–608, 2000.

[15] D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. On the least trimmed squares estimator. *Algorithmica*, pages 1–36, 2012.

[16] D. M. Mount, N. S. Netanyahu, K. R. Romanik, R. Silverman, and A. Y. Yu. A practical approximation algorithm for the LMS line estimator. *Computational Statistics & Data Analysis*, 51:2461–2486, 2007.

[17] D. M. Mount, N. S. Netanyahu, and E. Zuck. Analyzing the number of samples required for an approximate Monte-Carlo LMS line estimator. In M. Hubert, G. Pison, A. Struyf, and S. Van Aelst, editors, *Theory and Applications of Recent Robust Methods*, Statistics for Industry and Technology, pages 207–219. Birkhäuser, Basel, 2004.

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

[19] P. J. Rousseeuw. A diagnostic plot for regression outliers and leverage points. *Computational Statistics & Data Analysis*, 11:127–129, 1991.

[20] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. Wiley, New York, 1987.

[21] P. J. Rousseeuw and K. van Driessen. Computing LTS regression for large data sets. *Data Min. Knowl. Discov.*, 12:29–45, 2006.

[22] W. J. Rucklidge. *Efficient visual recognition using the Hausdorff distance*. Number 1173 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1996.

[23] W. J. Rucklidge. Efficiently locating objects using the Hausdorff distance. *International Journal of Computer Vision*, 24:251–270, 1997.

[24] D. L. Souvaine and J. M. Steele. Time- and space- efficient algorithms for least median of squares regression. *Journal of the American Statistical Association*, 82:794–801, 1987.