**Noname manuscript No.**
(will be inserted by the editor)

# A comparison of algorithms for the multivariate $L_1$-median

**Heinrich Fritz · Peter Filzmoser · Christophe Croux**

Received: date / Accepted: date

**Abstract** The $L_1$-median is a robust estimator of multivariate location with good statistical properties. Several algorithms for computing the $L_1$-median are available. Problem specific algorithms can be used, but also general optimization routines. The aim is to compare different algorithms with respect to their precision and runtime. This is possible because all considered algorithms have been implemented in a standardized manner in the open source environment R. In most situations, the algorithm based on the optimization routine NLM (non-linear minimization) clearly outperforms other approaches. Its low computation time makes applications for large and high-dimensional data feasible.

**Keywords** Algorithm · Multivariate median · Optimization · Robustness

## 1 Introduction

A prominent generalization of the univariate median to higher dimensions is the geometric median, also called $L_1$-median or spatial median. For a data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$

H. Fritz
Department of Statistics and Probability Theory
Vienna University of Technology
Tel.: +43-1-58801-10733
Fax: +43-1-58801-10799
E-mail: e0325693@student.tuwien.ac.at

P. Filzmoser
Department of Statistics and Probability Theory
Vienna University of Technology
E-mail: P.Filzmoser@tuwien.ac.at

C. Croux
Faculty of Business and Economics
K.U.Leuven & Tilburg University.
E-mail: Christophe.Croux@econ.kuleuven.be

with each $\boldsymbol{x}_i \in \mathbb{R}^p$, the $L_1$-median $\hat{\boldsymbol{\mu}}$ is defined as

$$\hat{\boldsymbol{\mu}}(\boldsymbol{X}) = \operatorname*{argmin}_{\boldsymbol{\mu}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{\mu}\| \tag{1}$$

where $\|\cdot\|$ denotes the Euclidean norm. In words, the $L_1$-median is the point for which the sum of the Euclidean distances to $n$ given data points is minimal. This problem is formulated in an even more general form by Weber (1909) (*Fermat-Weber problem*), as he refers to location issues in industrial contexts. If the data points are not collinear, the solution to problem (1) is unique (Weiszfeld, 1937). The $L_1$-median has several further attractive statistical properties, like:

(a) Its breakdown point is 0.5 (Lopuhaä and Rousseeuw, 1991), i.e., only if more than 50% of the data points are contaminated, the $L_1$-median can take values beyond all bounds.
(b) It is location and orthogonally equivariant, that is for any $\boldsymbol{b} \in \mathbb{R}^p$ and orthogonal $p \times p$ matrix $\boldsymbol{L}$,
$$\hat{\boldsymbol{\mu}}(\boldsymbol{LX} + \boldsymbol{b}) = \boldsymbol{L}\hat{\boldsymbol{\mu}}(\boldsymbol{X}) + \boldsymbol{b},$$
with $\boldsymbol{LX} + \boldsymbol{b} = \{\boldsymbol{Lx}_1 + \boldsymbol{b}, \ldots, \boldsymbol{Lx}_n + \boldsymbol{b}\}$.

Property (a) makes this estimator very attractive from a robustness point of view. According to property (b), the $L_1$-median is orthogonal equivariant, but not affine equivariant. Orthogonal equivariance is already sufficient for many situations, like for principal component analysis (PCA). Therefore several authors consider the $L_1$-median in the context of robust PCA (e.g., Croux and Ruiz-Gazen, 2005; Croux et al, 2007). Note that the $L_1$-median can be extended beyond the multivariate setting to functional spaces (Gervini, 2008) and Hilbert spaces (e.g. Chaudhuri, 1996; Debruyne et al, 2010).

An iterative algorithm for finding the numerical solution of the $L_1$-median has been proposed by Weiszfeld (1937). Several other algorithms will be outlined in Section 2. The goal of this paper is to compare the algorithms with respect to their precision and runtime (Section 3). For such a comparison, a unified implementation of the algorithms has been made using C++ code embedded in the R-library `pcaPP` (R Development Core Team, 2010), see Section 2.3. The final Section 4 concludes.

## 2 Algorithms for computing the $L_1$-median

For the computation of the estimate $\hat{\boldsymbol{\mu}}$ we have to minimize the convex function

$$S(\boldsymbol{\mu}) = \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{\mu}\|. \tag{2}$$

The algorithms tested and examined here can generally be divided into two categories:

### 2.1 General optimization procedures

The minimization of (2) can be done by numerical algorithms, developed for general, non-linear optimization purposes. One can either evaluate the target function $S(\boldsymbol{\mu})$ on several points, or additionally use the first derivative of $S(\boldsymbol{\mu})$,

$$\frac{\partial S(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} = -\sum_{i=1}^{n} \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}, \tag{3}$$

or the Hessian matrix

$$\frac{\partial^2 S(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}\, \partial \boldsymbol{\mu}^t} = \sum_{i=1}^{n} \left( \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}\, \boldsymbol{I}_p - \frac{(\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^t}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|^3} \right), \tag{4}$$

where $\boldsymbol{I}_p$ is the $p \times p$ identity matrix. Since all algorithms are implemented in the software environment R (R Development Core Team, 2010), we considered general unconstrained non-linear optimization algorithms which are accessible in this environment for our purpose:

– **NM:** Nelder and Mead (1965) proposed a simplex method for minimizing functions of $p$ variables, also known as downhill simplex method. Values of the target function are compared at $p+1$ points, whereas the point with highest target value is replaced in each iteration. Rather many iterations are needed till convergence, but as this is one of the most common simplex algorithms, it is included in this comparison.
– **BFGS:** Broyden, Fletcher, Goldfarb and Shanno proposed a quasi-Newton algorithm searching for a stationary point of a function by local quadratic approximation (see, e.g., Nocedal and Wright, 2006). In contrast to *real* Newton methods, this algorithm does not require an analytical computation of the exact Hessian matrix, as this is approximated internally by the algorithm. Since in high-dimensional situations the computation of the exact Hessian matrix can be quite time consuming, algorithms that approximate the Hessian matrix internally are to be preferred in this context.
– **CG:** Conjugate gradient algorithms are iterative methods, known for their low memory requirements. Quasi-Newton methods usually converge after fewer iterations, but a single iteration of a conjugate gradient method is computationally less intensive. In the subsequent simulations, the version of Fletcher and Reeves (1964) is applied. The gradient information in equation (3) is used by this method.
– **NLM:** Non-linear minimization can be carried out by a Newton-type algorithm (Dennis and Schnabel, 1983), using the gradient information from equation (3). It provides two options regarding the Hessian matrix: an analytical representation can be provided, or the Hessian matrix is again approximated internally by secant methods. Approximating the Hessian matrix turned out to be the faster approach, particularly when high-dimensional data sets are processed. Hence in subsequent simulations the Hessian matrix is always approximated rather than analytically calculated. From the three available optimization types introduced in Dennis and Schnabel (1983), *Line Search*, *Double Dogleg*, and *More-Hebdon*, the first method is applied here, as in this context its convergence characteristics turned out to be most reliable among these three.

## 2.2 Problem specific algorithms

For the specific problem of computing the $L_1$-median, several algorithms have been proposed in the literature. Here we mention the algorithm of Weiszfeld (1937), which is the basis for an improved version by Vardi and Zhang (2000). Another algorithm, based on the steepest descent, has been introduced by Hössjer and Croux (1995), and will also be examined here.

- Weiszfeld (1937) formulated an iterative approach for solving the Fermat-Weber problem for data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and $n$ non-negative weights. In this paper all weights are equal. A current (or initial) solution $\hat{\boldsymbol{\mu}}_l$ is improved by calculating a scaled sum of all observations:

$$T_0(\boldsymbol{\mu}) = \frac{\sum\limits_{i=1}^{n} \frac{\boldsymbol{x}_i}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}}{\sum\limits_{i=1}^{n} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}} \tag{5}$$

$$\hat{\boldsymbol{\mu}}_{l+1} = \begin{cases} T_0(\hat{\boldsymbol{\mu}}_l) & \text{if } \hat{\boldsymbol{\mu}}_l \notin \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \\ \hat{\boldsymbol{\mu}}_l & \text{if } \hat{\boldsymbol{\mu}}_l \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \end{cases} \tag{6}$$

  This algorithm converges given that $\hat{\boldsymbol{\mu}}_l \notin \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ for every iteration step $l \in I\!N$.
- **VaZh:** Vardi and Zhang (2000) improved the behavior of Weiszfeld's algorithm in case that $\hat{\boldsymbol{\mu}}_l \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ appears in any iteration. The resulting algorithm is, apart from the treatment of this particular issue, quite similar, and for this reason no comparison is made with Weiszfeld's original algorithm later on in the simulation study. The proposal of Vardi and Zhang (2000) is as follows:

$$T_1(\boldsymbol{\mu}) = \frac{\sum\limits_{\boldsymbol{x}_i \neq \boldsymbol{\mu}} \frac{\boldsymbol{x}_i}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}}{\sum\limits_{\boldsymbol{x}_i \neq \boldsymbol{\mu}} \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|}} \tag{7}$$

$$\eta(\boldsymbol{\mu}) = \begin{cases} 1 & \text{if } \boldsymbol{\mu} = \boldsymbol{x}_i, \; i = 1, \ldots, n \\ 0 & \text{else} \end{cases} \tag{8}$$

$$R(\boldsymbol{\mu}) = \sum_{\boldsymbol{x}_i \neq \boldsymbol{\mu}} \frac{\boldsymbol{x}_i - \boldsymbol{\mu}}{\|\boldsymbol{x}_i - \boldsymbol{\mu}\|} \tag{9}$$

$$\gamma(\boldsymbol{\mu}) = \min\left(1, \frac{\eta(\boldsymbol{\mu})}{\|R(\boldsymbol{\mu})\|}\right) \tag{10}$$

$$\hat{\boldsymbol{\mu}}_{l+1} = (1 - \gamma(\hat{\boldsymbol{\mu}}_l))\, T_1(\hat{\boldsymbol{\mu}}_l) + \gamma(\hat{\boldsymbol{\mu}}_l)\hat{\boldsymbol{\mu}}_l. \tag{11}$$

  The case $\hat{\boldsymbol{\mu}}_l \notin \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ implies $\gamma(\hat{\boldsymbol{\mu}}_l) = 0$, and the algorithm behaves exactly as in (5). Otherwise, if $\hat{\boldsymbol{\mu}}_l \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, the sum in (7) is calculated as in (5), but only over the $\boldsymbol{x}_i \neq \hat{\boldsymbol{\mu}}_l$, whereas observation $\boldsymbol{x}_i = \hat{\boldsymbol{\mu}}_l$ is added afterwards in (11), applying weight $\gamma(\hat{\boldsymbol{\mu}}_l)$.
- **HoCr:** Hössjer and Croux (1995) proposed an approach which combines a steepest descent algorithm with step halving. The current solution is improved by stepping into the direction with steepest descent of the target function. If the target function increases after this step, the step size is being halved until the target function decreases. If the step size shrinks to zero without finding a better solution, the algorithm has converged. A detailed description of the algorithm including pseudocode is given by the authors. Their algorithm can also be used for the rank based extension of the $L_1$-median they propose.

For reasons of completeness, let us mention two other algorithms proposed in the literature for computing the $L_1$-median. Gower (1974) proposed a steepest descent algorithm combined with a bisection algorithm. It is somehow similar to the HoCr algorithm, but the use of the bisection method instead of step-halving considerably increases the computation time. Bedall and Zimmermann (1979) proposed to use the Newton-Raphson procedure with the exact expression (4) for the Hessian matrix, and is similar to the NLM method with analytical second derivative. It turned out to be much slower than the NLM procedure. The algorithms of Gower (1974) and Bedall and Zimmermann (1979) are included in the R-package `depth`. Due to their similarity with algorithms already included in the simulation study, and since they are not competitive in terms of computation speed, we do not report their performance in this paper.

## 2.3 Implementation

Due to efficiency issues (runtime and memory), all methods are implemented in C++ and are embedded in the R-library `pcaPP` (version 1.8-1). The algorithms based on general optimization methods are simply wrapping the available routines in R, see Table 1. The routine for non-linear minimization (`optif9`) originates from the UNCMIN-

**Table 1** Implementation of the optimization methods as wrapper functions.

| Algorithm | R - Optimizer | `pcaPP` - Routine |
|---|---|---|
| Nelder and Mead (NM) | `nmmin` | `l1median_NM` |
| Broyden, Fletcher, Goldfarb and Shanno (BFGS) | `vmmin` | `l1median_BFGS` |
| Conjugate gradient (CG) | `cgmin` | `l1median_CG` |
| Non-linear minimization (NLM) | `optif9` | `l1median_NLM` |
| Vardi and Zhang (VaZh) | | `l1median_VaZh` |
| Hössjer and Croux (HoCr) | | `l1median_HoCr` |

Fortran package by R.B. Schnabel which implements Newton and quasi-Newton algorithms for unconstrained minimization, see Dennis and Schnabel (1983), Schnabel et al (1985). All other mentioned routines are C-implementations which come along with R. By default the component-wise median is used as starting value for all mentioned algorithms.

The more specific $L_1$-median routines (Vardi and Zhang, 2000; Hössjer and Croux, 1995) are transcripts of the routines published in the R-library `robustX`, whereas the original implementation of the algorithm by Hössjer and Croux (1995) was made available by the authors. The implementation of the algorithm of Vardi and Zhang (2000) is changed slightly, as the original algorithm crashes if $\boldsymbol{\mu} = \boldsymbol{x}_i$ for more than one $i \in \{1, \ldots, n\}$, see equation (8). Although this might be a rare situation, equation (8) needs to be changed to

$$\eta(\boldsymbol{\mu}) = \sum_{\boldsymbol{x}_i = \boldsymbol{\mu}} 1 \tag{12}$$

in order to stabilize the algorithm in such degenerated cases.

## 2.4 Convergence

When comparing different ways of solving a problem, one major issue is to create equal and thus fair conditions for each approach which is to be examined. Here each algorithm can be tested providing exactly the same input, whereas the output quality can be compared likewise by checking the value of the target function $S(\hat{\boldsymbol{\mu}})$ of (2). However, it is not as easy to control the precision of the results of each particular algorithm. Each of the mentioned algorithms provides an input control parameter $\tau$, which is a tolerance level influencing the convergence behavior, and is mainly used to specify the desired precision. Unfortunately this tolerance level is interpreted differently, as shown in Table 2.

**Table 2** Convergence criteria for different optimization routines.

| Algorithm | Convergence criterion | Based on |
|---|---|---|
| NM<br>BFGS<br>CG | $S\left(\hat{\boldsymbol{\mu}}_{l-1}\right) - S(\hat{\boldsymbol{\mu}}_l) \leq \tau\left(S(\hat{\boldsymbol{\mu}}_l) + \tau\right)$ | Relative change of $S(\hat{\boldsymbol{\mu}})$ |
| NLM | $\bigtriangledown := \frac{\left\lvert\frac{\partial S(\hat{\boldsymbol{\mu}}_l)}{\partial \hat{\boldsymbol{\mu}}_l}\right\rvert \max\{\lVert\hat{\boldsymbol{\mu}}_l\rVert,1\}}{\max\{\lvert S(\hat{\boldsymbol{\mu}}_l)\rvert,1\}}$<br>$\max_i \bigtriangledown_i \leq \tau$ with $\bigtriangledown = (\bigtriangledown_1, \ldots, \bigtriangledown_p)^t$ | Maximum scaled gradient |
| VaZh | $\lVert\hat{\boldsymbol{\mu}}_{l-1} - \hat{\boldsymbol{\mu}}_l\rVert_1 \leq \tau\lVert\hat{\boldsymbol{\mu}}_l\rVert_1$ | Relative change of $\hat{\boldsymbol{\mu}}$ ($L_1$-norm) |
| HoCr | $\lVert\hat{\boldsymbol{\mu}}_{l-1} - \hat{\boldsymbol{\mu}}_l\rVert_2 \leq \tau$ | Change of $\hat{\boldsymbol{\mu}}$ ($L_2$-norm) |

In order to still get comparable results, we have to choose $\tau$ appropriately. This is done by examining first, which tolerance level $\tau$ leads to comparable precision in a particular situation, and then these tolerance levels are applied in all subsequent simulation settings. Details are provided in the next section.

## 3 Comparison of the algorithms

In order to test the performance of the algorithms, several types of artificial and real data samples are chosen. On a particular data set, the $L_1$-median is computed with each algorithm discussed above and the resulting estimations are compared by considering the values of the target function (2). The smallest resulting target value is used as reference, whereas deviations from this reference value indicate worse approximations of the $L_1$-median. Throughout this paper, we use this *deviation* as a measure of precision of an algorithm. The best algorithm(s) always yield a deviation of exactly zero. In the following simulated scenarios, each simulation is performed 100 times with data sets sampled from the same distribution. As measure for overall precision the 95% quantile of the resulting deviations is considered, referred to as the *95% deviation quantile*. The motivation for this measure is that it reflects the precision of the vast majority of runs, but does not account for single runs where the convergence behavior was different just by chance. Additionally, the (median) runtime in seconds is reported for each algorithm. All simulations are computed on an AMD Athlon 64 X2 4200+ Processor at 2.2 GHz.

In order to stop non converging algorithms from freezing the process, the maximum number of iterations is set to 500 for each algorithm. If not stated differently, the simulated data sets consist of $n = 1000$ observations and $p = 100$ variables. The covariance matrix $\boldsymbol{C}$ of the distribution used to generate the data is diagonal with diagonal elements $p + 1 - i$, for $i = 1, \ldots, p$. Outliers are generated by randomly selecting observations from the simulated data matrix, multiplying them with a value of 10, and shifting them in each coordinate by the value 10.

## 3.1 Adjusting tolerance levels for convergence

Before comparing any results of the different algorithms, their convergence criteria shall be examined and their tolerance levels need to be adjusted, such that they deliver equal precision for a particular simulation setting. For that purpose a simple simulation is performed with several tolerance levels $\tau$, monitoring the resulting precisions. This allows to obtain a tolerance level for each algorithm leading to "optimal" precisions. In the subsequent simulations, these tolerance levels are used and so the resulting figures are directly comparable. The algorithms are applied to 100 different $p$-variate normally distributed data sets (we used $n = 1000$ and $p = 100$ in the following) with center $\mathbf{0}$, covariance matrix $\boldsymbol{C}$ (see above) and without outliers. The tolerance level is altered between 1e-1 and 1e-20. Figure 1 shows the 95% deviation quantiles of each algorithm computed over different tolerance levels. Note that a 95% deviation quantile of exactly 0 has been truncated to 1e-15 for being able to use logarithmic scales. A value of 1e-15 (or below) is reached at a tolerance level of 1e-5 for HoCr, at 1e-6 for VaZh, at 1e-9 for CG, and at 1e-11 for NLM. This high precision cannot be reached for the algorithms BFGS and NM within the considered range of the tolerance levels; NM seems to be totally unaffected by the choice of the tolerance level. Also changing the different tuning parameters of the algorithm NM does not lead to better results in a simulation as presented here.

Considering the computation time of the algorithms with respect to the used tolerance level $\tau$ (Figure 2), we note that the runtime of the algorithm NLM seems to be rather unaffected by the specified value of $\tau$. On the other hand, CG shows a large increase in runtime when $\tau$ is raised from 1e-11 to 1e-13. On the whole, the relative ranking of the different algorithms remains about the same for different levels of $\tau$. Hence, the faster algorithms when requiring a high tolerance level will also be the faster ones when only a rude approximation is needed. In particular this implies that it will not be advantageous to use the output of one algorithm as starting value for another algorithm. For this reason, we stick to the coordinatewise median as a starting value for all the algorithms we consider.

Table 3 shows the chosen tolerance levels for subsequent simulations. The largest level of $\tau$ at which each algorithm reaches its best performance (zero deviation) is chosen and then scaled down by a factor 1e3, compensating for different simulation scenarios, as the data structure might slightly influence the convergence behavior. With respect to the high increase of the runtime of the algorithm CG when $\tau$ excesses 1e-11, this algorithm's tolerance level is set to 1e-10. As method NM does not seem to be affected by the tolerance level, the choice of $\tau$ = 1e-10 is admittedly arbitrary.

**Table 3** Tolerance levels $\tau$ as used for each particular algorithm.

| NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|
| 1e-10 | 1e-17 | 1e-10 | 1e-14 | 1e-9 | 1e-8 |

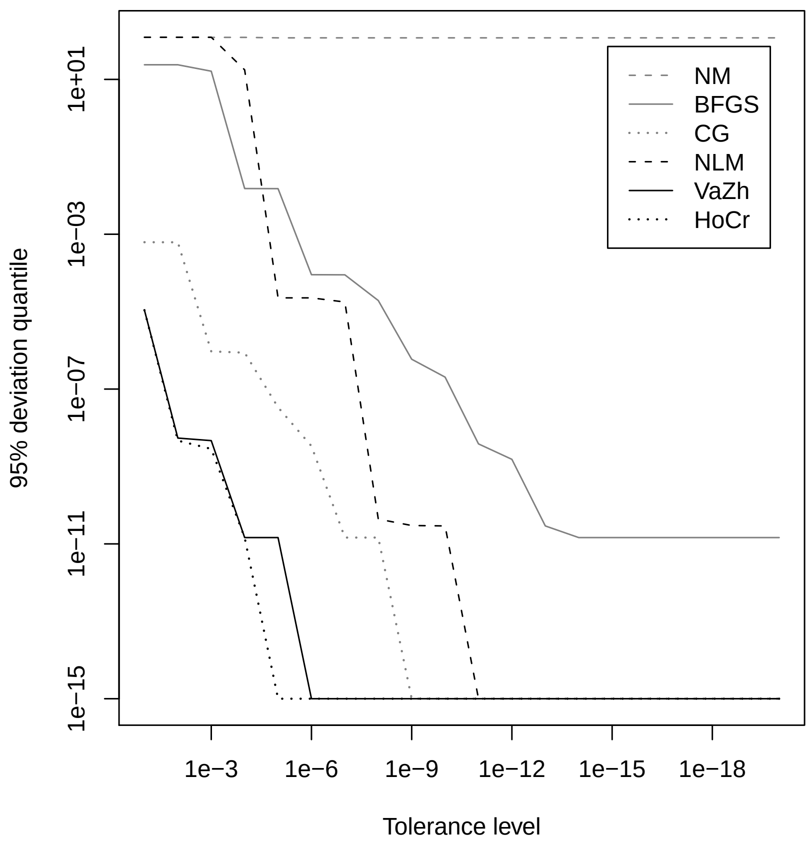

**Fig. 1** The 95% deviation quantiles of the $L_1$-median target values as a function of the tolerance level $\tau$.

By selecting the tolerance level $\tau$ as outlined above, we aim at optimizing the precision of the algorithm, as measured by the 95% quantile of the deviation. This is not implying that all considered algorithms end up with the same precision, but it does turn out that the best performing algorithms have precisions very close to each other. We then discriminate between these best performing algorithms by comparing their run times. An alternative strategy might have been to control the running times, and then compare the attained precision. We did not pursue this alternative strategy since, besides several other implementation issues, a tolerance level $\tau$ still needs to be specified, and it is not so clear how to do this.

## 3.2 Uncorrelated data

The observations are drawn randomly from $p$-variate normal ($N_p$) and log-normal ($LN_p$) distributions, respectively, with center $\mathbf{0}$ and covariance matrix $\boldsymbol{C}$. Simulations on multivariate $t$-distributed data have been computed too, but as the algorithms per-

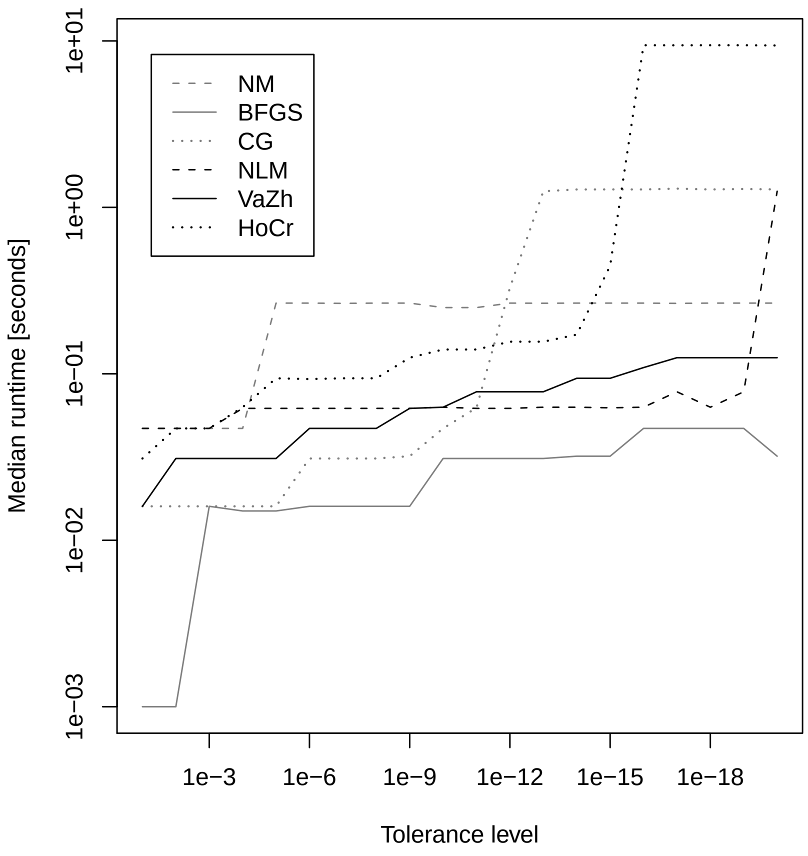

**Fig. 2** Median runtimes [seconds] of the $L_1$-median computation as a function of the tolerance level $\tau$.

form quite similar as on normally distributed data, these results are omitted here. In order to get an impression on how the algorithms are relatively affected by outliers, the percentage of outliers, $p_{out}$, varies between 0% and 40% . Table 4 shows the 95% deviation quantiles of each algorithm, for the given distributions and different percentages of outliers. For $p_{out} = 0$ (and normally distributed data) this is the same simulation setting as before when the tolerance levels have been chosen, and it is not surprising that apart from BFGS and NM all algorithms return the same precision, yielding a 95% deviation quantile of 0. As the concept of the $L_1$-median may downweight but never completely trims any observation, added outliers always slightly influence the estimation. However, increasing $p_{out}$ does not influence the algorithms' relative performance, as none of the methods can be identified as particularly sensitive to a different outlier proportion among the tested candidates. The runtime behavior in this simulation setting is shown in Table 5. It is not possible to point out one algorithm which is always the fastest while delivering the best approximation quality. On normally distributed data, BFGS is the fastest algorithm, but as seen before it is not as accurate as other methods. The runtimes of the algorithms seem to be quite independent from the added

**Table 4** 95% deviation quantiles of $L_1$-median algorithms applied to uncorrelated data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 115 | 1.46e-11 | 0 | 0 | 0 | 0 |
| | 10 % | 118 | 2.91e-11 | 0 | 0 | 0 | 0 |
| | 20 % | 133 | 1.46e-12 | 0 | 0 | 0 | 0 |
| | 30 % | 147 | 0 | 0 | 0 | 0 | 0 |
| | 40 % | 146 | 5.82e-11 | 0 | 0 | 0 | 0 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 1.66 | 1.19e-13 | 0 | 0 | 0 | 0 |
| | 10 % | 3.51 | 4.77e-13 | 0 | 0 | 0 | 0 |
| | 20 % | 5.98 | 9.09e-13 | 0 | 0 | 0 | 0 |
| | 30 % | 9.60 | 9.09e-13 | 0 | 0 | 0 | 0 |
| | 40 % | 16.20 | 1.82e-12 | 0 | 0 | 0 | 0 |

amount of outliers, and they are all in about the same range (apart from NM being always the slowest choice which obviously is caused by convergence issues). Applying the algorithms on log-normally distributed data, however, yields the best results for the NLM algorithm. In terms of computation time, NLM is the only algorithm which is not heavily influenced by the distribution type, nor by the amount of outliers added. For the algorithms VaZh, HoCr and CG, the data configuration has a notable effect since their runtimes increase up to a factor of 6 (CG) when highly contaminated data are processed. The algorithm of Nelder and Mead (NM) never converges within the 500 allowed iterations, which explains its high deviation values and longer runtimes.

**Table 5** Median runtimes [seconds] of the $L_1$-median algorithms applied to uncorrelated data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.265 | 0.032 | 0.047 | 0.063 | 0.062 | 0.094 |
| | 10 % | 0.266 | 0.046 | 0.125 | 0.063 | 0.062 | 0.078 |
| | 20 % | 0.265 | 0.032 | 0.047 | 0.063 | 0.062 | 0.078 |
| | 30 % | 0.265 | 0.031 | 0.046 | 0.063 | 0.062 | 0.078 |
| | 40 % | 0.265 | 0.031 | 0.047 | 0.063 | 0.062 | 0.078 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.266 | 0.172 | 0.047 | 0.063 | 0.062 | 0.078 |
| | 10 % | 0.266 | 0.172 | 0.281 | 0.063 | 0.062 | 0.078 |
| | 20 % | 0.265 | 0.187 | 0.109 | 0.078 | 0.078 | 0.109 |
| | 30 % | 0.265 | 0.141 | 0.211 | 0.078 | 0.110 | 0.141 |
| | 40 % | 0.265 | 0.109 | 0.297 | 0.078 | 0.218 | 0.250 |

Finally, to check the robustness of our findings with respect to the chosen performance measures, we show in Table 6 the average of the deviations over the simulated samples, as an alternative to the 95% quantile deviation we used before. In line with the findings from Table 4, we see that the NM algorithm is not competitive, and also BFGS does worse than the other considered algorithms. For the other algorithms, the average deviations are very small, and there is hardly any difference between them, confirming the conclusions made from the corresponding table containing the 95% quantile deviations. Furthermore, Table 7 presents the mean, instead of the median, runtime. Comparing Tables 7 and 5 shows that the mean and median runtime remain very close to each other. In the remainder of this paper, we only report the 95% quantile deviations and median runtimes.

**Table 6** Average deviation of the $L_1$-median target value applied to uncorrelated data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 91.1 | 3.2e-12 | 1.46e-13 | 1.46e-13 | 0 | 1.46e-13 |
| | 10 % | 93.1 | 1.75e-12 | 0 | 0 | 2.91e-13 | 0 |
| | 20 % | 114 | 1.46e-12 | 0 | 0 | 5.82e-13 | 0 |
| | 30 % | 122 | 2.33e-12 | 5.82e-13 | 5.82e-13 | 0 | 5.82e-13 |
| | 40 % | 116 | 6.98e-12 | 0 | 0 | 0 | 0 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 1.42 | 5e-14 | 2.27e-15 | 2.27e-15 | 0 | 2.27e-15 |
| | 10 % | 3.05 | 2.36e-13 | 9.09e-15 | 9.09e-15 | 1.82e-14 | 9.09e-15 |
| | 20 % | 5.51 | 3.73e-13 | 9.09e-15 | 9.09e-15 | 3.64e-14 | 9.09e-15 |
| | 30 % | 9.09 | 4.73e-13 | 1.82e-14 | 1.82e-14 | 9.09e-15 | 1.82e-14 |
| | 40 % | 15.4 | 4.18e-13 | 3.64e-14 | 3.64e-14 | 0 | 3.64e-14 |

**Table 7** Average runtimes [seconds] of the $L_1$-median algorithms applied to uncorrelated data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.262 | 0.039 | 0.062 | 0.064 | 0.062 | 0.094 |
| | 10 % | 0.262 | 0.039 | 0.146 | 0.064 | 0.061 | 0.086 |
| | 20 % | 0.260 | 0.039 | 0.090 | 0.065 | 0.062 | 0.083 |
| | 30 % | 0.262 | 0.036 | 0.100 | 0.064 | 0.062 | 0.076 |
| | 40 % | 0.261 | 0.031 | 0.105 | 0.065 | 0.061 | 0.074 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.262 | 0.177 | 0.049 | 0.068 | 0.062 | 0.076 |
| | 10 % | 0.262 | 0.177 | 0.245 | 0.070 | 0.062 | 0.086 |
| | 20 % | 0.261 | 0.185 | 0.172 | 0.072 | 0.079 | 0.106 |
| | 30 % | 0.262 | 0.143 | 0.318 | 0.072 | 0.114 | 0.142 |
| | 40 % | 0.262 | 0.102 | 0.544 | 0.074 | 0.212 | 0.252 |

## 3.3 Correlated data

In this section we investigate the effect of a correlation structure within the simulated data on the performance of the algorithms. Therefore, we use a covariance matrix $\boldsymbol{C'}$ for data generation, with elements 1 in the diagonal, and numbers $c$ as off-diagonal elements. The values of $c$ are set to 0.5, 0.9, and 0.99 among the different simulation scenarios. In addition, the effect of the data distribution and the influence of outliers is observed. Table 8 shows the precision of the algorithms. Their performance does not seem to be really influenced by the correlation level $c$, the only difference is observed for the algorithm HoCr which performs slightly worse for log-normally distributed highly correlated data.

According to Table 9, the only algorithms unaffected by the correlation level are NM and NLM. NM again did not converge which explains the constant runtime of 265 milliseconds. NLM however converges almost within the same time as when applied on uncorrelated data, which is outstanding among all tested methods. The algorithms CG, HoCr and VaZh require more computation time for highly correlated data.

## 3.4 High-dimensional data with low sample size ($p \gg n$)

Here we generate data according to $N_p(\mathbf{0}, \boldsymbol{I}_p)$ and $LN_p(\mathbf{0}, \boldsymbol{I}_p)$, respectively, for $n = 10$ and $p = 100$. As previously, contamination is added and its percentage is varied. Since the rank of the generated $p$-dimensional data is $n$, it is possible to reduce the

**Table 8** 95% deviation quantiles of $L_1$-median algorithms applied to correlated data.

| Distribution | $p_{out}$ | $r$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C}')$ | 0 % | 0.5 | 19.2 | 1.82e-12 | 0 | 0 | 0 | 0 |
| | | 0.9 | 23.3 | 1.82e-12 | 0 | 0 | 0 | 0 |
| | | 0.99 | 31.0 | 1.82e-12 | 0 | 0 | 0 | 0 |
| | 30 % | 0.5 | 33.2 | 7.28e-12 | 0 | 0 | 0 | 0 |
| | | 0.9 | 22.1 | 7.28e-12 | 0 | 0 | 0 | 0 |
| | | 0.99 | 57.8 | 7.28e-12 | 0 | 0 | 0 | 0 |
| $LN_p(\mathbf{0}, \mathbf{C}')$ | 0 % | 0.5 | 3.09 | 1.14e-13 | 0 | 0 | 0 | 0 |
| | | 0.9 | 2.83 | 1.14e-13 | 0 | 0 | 0 | 1.14e-13 |
| | | 0.99 | 4.34 | 1.14e-13 | 0 | 0 | 0 | 2.27e-13 |
| | 30 % | 0.5 | 3.62 | 9.09e-13 | 0 | 0 | 0 | 0 |
| | | 0.9 | 3.13 | 9.09e-13 | 0 | 0 | 0 | 0 |
| | | 0.99 | 3.42 | 9.09e-13 | 0 | 0 | 0 | 9.09e-13 |

**Table 9** Median runtimes [seconds] of the $L_1$-median algorithms applied to correlated data.

| Distribution | $p_{out}$ | $c$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C}')$ | 0 % | 0.5 | 0.265 | 0.078 | 0.110 | 0.063 | 0.156 | 0.219 |
| | | 0.9 | 0.265 | 0.094 | 0.296 | 0.078 | 0.281 | 0.422 |
| | | 0.99 | 0.265 | 0.125 | 0.344 | 0.078 | 0.468 | 0.688 |
| | 30 % | 0.5 | 0.265 | 0.063 | 0.297 | 0.078 | 0.156 | 0.188 |
| | | 0.9 | 0.265 | 0.078 | 0.593 | 0.078 | 0.250 | 0.328 |
| | | 0.99 | 0.265 | 0.110 | 0.718 | 0.078 | 0.422 | 0.547 |
| $LN_p(\mathbf{0}, \mathbf{C}')$ | 0 % | 0.5 | 0.266 | 0.157 | 0.141 | 0.078 | 0.156 | 0.203 |
| | | 0.9 | 0.265 | 0.156 | 0.211 | 0.078 | 0.282 | 0.359 |
| | | 0.99 | 0.266 | 0.140 | 0.375 | 0.078 | 0.453 | 0.570 |
| | 30 % | 0.5 | 0.265 | 0.094 | 0.539 | 0.078 | 0.172 | 0.203 |
| | | 0.9 | 0.265 | 0.156 | 0.609 | 0.078 | 0.265 | 0.313 |
| | | 0.99 | 0.266 | 0.141 | 1.258 | 0.078 | 0.406 | 0.500 |

dimensionality to $n$ without any loss of information. This is done by singular value decomposition (SVD) as follows (compare Serneels et al, 2005). Let $\boldsymbol{X}$ be the simulated data matrix, then $\boldsymbol{X}^t = \boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^t$ is the SVD of $\boldsymbol{X}^t$, with $\boldsymbol{U}$ an $n \times n$ orthogonal matrix, $\boldsymbol{S}$ a diagonal matrix including the $n$ singular values in its diagonal, and $\boldsymbol{V}$ a $p \times n$ orthogonal matrix. The matrix $\tilde{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{S}$ has only size $n \times n$, but it contains the same information as $\boldsymbol{X}$. Using this dimension reduction, we can apply the $L_1$-median algorithms to both $\boldsymbol{X}$ and $\tilde{\boldsymbol{X}}$, and compare the resulting estimates $\hat{\boldsymbol{\mu}}$ and $\hat{\tilde{\boldsymbol{\mu}}}$, respectively. This can be done by back-transforming the column vector $\hat{\tilde{\boldsymbol{\mu}}}$ to the original space with $\boldsymbol{V}\hat{\tilde{\boldsymbol{\mu}}}$, and comparing the results with the Euclidean distance

$$\delta = \|\hat{\boldsymbol{\mu}} - \boldsymbol{V}\hat{\tilde{\boldsymbol{\mu}}}\|. \tag{13}$$

This difference between estimations in transformed and original space is computed 100 times for each simulation setting, and the 95% distance quantile is reported in Table 10. As the algorithm NLM provided the best results in previous simulations, it is rather expected that it also performs well in here. Indeed, this is confirmed, as NLM gives the smallest distances between the estimations computed in the original and in the transformed space. The only algorithm which sticks out again due to its instability and inaccuracy is NM, which has to be stopped after 500 iterations without reaching convergence.

**Table 10** 95% distance quantiles of $L_1$-median algorithms applied to the original and the transformed data sets.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \boldsymbol{I}_p)$ | 0 % | 3.98 | 2.71e-07 | 2.93e-07 | 7.31e-12 | 8.49e-10 | 9.41e-08 |
| | 10 % | 3.81 | 2.37e-07 | 2.81e-07 | 7.41e-12 | 8.27e-10 | 1.13e-07 |
| | 20 % | 4.42 | 2.32e-07 | 3.21e-07 | 1.14e-11 | 1.04e-09 | 1.21e-07 |
| | 30 % | 4.92 | 3.54e-07 | 3.57e-07 | 1.53e-11 | 1.06e-09 | 1.66e-07 |
| | 40 % | 5.71 | 4.6e-07 | 4.1e-07 | 1.77e-11 | 1.51e-09 | 2.07e-07 |
| $LN_p(\mathbf{0}, \boldsymbol{I}_p)$ | 0 % | 0.0401 | 2.46e-09 | 2.57e-09 | 6.44e-15 | 8.41e-12 | 3.74e-10 |
| | 10 % | 0.0376 | 2.27e-09 | 2.58e-09 | 8.19e-15 | 8.07e-12 | 1.22e-09 |
| | 20 % | 0.0424 | 3.29e-09 | 2.88e-09 | 1.47e-14 | 1.17e-11 | 1.99e-09 |
| | 30 % | 0.0464 | 5.24e-09 | 3.69e-09 | 2.78e-14 | 1.79e-11 | 3.56e-09 |
| | 40 % | 0.0591 | 6.06e-09 | 6.08e-09 | 4.97e-14 | 1.41e-10 | 7.46e-09 |

## 3.5 Degenerated situations

In this subsection we study the behavior of the algorithms at two particular data structures. First we consider the case of collinear data, secondly the case where more than half of the data points are coinciding. In these settings the exact value of the minimum of the objective function (2) is known. The deviation of an algorithm is now computed relative to the exact optimum.

– If $n$ is even, and all observations are collinear, the $L_1$-median is not uniquely defined, as any point between the 2 innermost observations is a proper solution (this is in analogy to the univariate median). As such a data structure has rank one, the application of the classical median is possible yielding the same result as the $L_1$-median. Therefore the data matrix has to be projected onto the line given by two arbitrary (different) observations, which in this case coincides with the first principal component. The median is computed in this one-dimensional subspace, whereas the result is transformed back into the original space afterwards.
  Simulations on data sets generated with a covariance matrix with arbitrary values (finite and positive) for the variances, and off-diagonal elements $c_{ij} = \sqrt{(c_{ii}c_{jj})}$ $\forall i \neq j$ (perfect collinearity of all variables) and arbitrary (finite) mean vectors result in Table 11. Apart from two occurrences where VaZh gives slightly better results, the algorithms CG, NLM, VaZh and HoCr show equal performance and return an exact solution. BFGS still returns good results but seems to have numerical problems due to its slight imprecision. NM seems to deliver arbitrary results, which are caused by convergence problems. The runtime comparison in Table 12 shows that BFGS is always the fastest algorithm, followed by NLM which is as fast as in the first simulation setting. The runtimes of CG, VaZh and HoCr increased by a factor 2-4 compared to Table 5.
– If more than $n/2$ observations are concentrated in one point, say $\boldsymbol{y}$, the solution of the $L_1$-median is $\hat{\boldsymbol{\mu}} = \boldsymbol{y}$. A simulation setting as in Section 3.2 but with $n/2 + 1$ observations set to $\boldsymbol{y} = \mathbf{1}$ (vector of ones) has been carried out, expecting the resulting $L_1$-median estimation to be $\hat{\boldsymbol{\mu}} = \mathbf{1}$. All algorithms are able to find the known center exactly, thus a table showing deviation quantiles can be omitted. However, the computation times differ for some algorithms (Table 13). Although NM converges to the exact solution in this setting, it converges slowly, resulting in a considerably higher value for the runtime. Also NLM is unable to detect this degenerated situation immediately. All other algorithms stop after a single iteration

**Table 11** 95% error quantiles of $L_1$-median estimations applied to collinear data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C}))$ | 0 % | 6.42e+03 | 9.09e-13 | 0 | 0 | 0 | 0 |
| | 10 % | 6.01e+03 | 3.64e-12 | 0 | 0 | 0 | 0 |
| | 20 % | 5.66e+03 | 1.82e-13 | 0 | 0 | 0 | 0 |
| | 30 % | 5.36e+03 | 7.28e-12 | 0 | 0 | 0 | 0 |
| | 40 % | 5.27e+03 | 7.28e-12 | 7.28e-12 | 7.28e-12 | 0 | 7.28e-12 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 1.08e+04 | 1.82e-12 | 9.09e-14 | 9.09e-14 | 0 | 1.82e-12 |
| | 10 % | 1.08e+04 | 3.82e-12 | 0 | 0 | 0 | 0 |
| | 20 % | 1.12e+04 | 7.28e-12 | 0 | 0 | 0 | 0 |
| | 30 % | 1.23e+04 | 1.46e-11 | 0 | 0 | 0 | 0 |
| | 40 % | 1.40e+04 | 0 | 0 | 0 | 0 | 0 |

**Table 12** Median runtimes [seconds] of the $L_1$-median algorithms applied to collinear data.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.266 | 0.047 | 0.094 | 0.078 | 0.258 | 0.446 |
| | 10 % | 0.266 | 0.047 | 0.125 | 0.078 | 0.250 | 0.422 |
| | 20 % | 0.266 | 0.047 | 0.195 | 0.078 | 0.250 | 0.406 |
| | 30 % | 0.265 | 0.047 | 0.172 | 0.078 | 0.250 | 0.391 |
| | 40 % | 0.266 | 0.047 | 0.125 | 0.078 | 0.282 | 0.422 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.266 | 0.047 | 0.109 | 0.093 | 0.235 | 0.422 |
| | 10 % | 0.266 | 0.054 | 0.148 | 0.093 | 0.266 | 0.453 |
| | 20 % | 0.266 | 0.047 | 0.133 | 0.093 | 0.250 | 0.382 |
| | 30 % | 0.266 | 0.047 | 0.125 | 0.079 | 0.328 | 0.469 |
| | 40 % | 0.265 | 0.039 | 0.125 | 0.094 | 0.406 | 0.578 |

and thus converge so quickly, that the used method for measuring time is not able to record such short periods.
The algorithms HoCr and VaZh have to be pointed out here, as they can handle such occurrences separately: both algorithms are based on the distances of each observation to the current $L_1$-median estimation, which is calculated during each iteration. By simply checking how many observations have zero distance to the current $L_1$-median estimation, the method can detect such cases and stop the iteration. A different return code is provided in order to inform the user about such rare occurrences.

**Table 13** Median runtimes [seconds] of the $L_1$-median algorithms applied to the degenerated data structure.

| Distribution | $p_{out}$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| $N_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.265 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 10 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 20 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 30 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 40 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| $LN_p(\mathbf{0}, \mathbf{C})$ | 0 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.008 | 0.000 |
| | 10 % | 0.265 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 20 % | 0.265 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 30 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |
| | 40 % | 0.266 | 0.000 | 0.000 | 0.047 | 0.000 | 0.000 |

Although the situations examined here would rarely occur, it is important to keep in mind that there might be further data configurations which are not so easy to handle.

## 3.6 Real data examples

For testing the $L_1$-median algorithms on real data, the data sets `bhorizon`, `chorizon`, `humus`, `moss`, `bssbot` and `bsstop`, from the R-library `mvoutlier` are used. These data contain the concentration of chemical elements in the soil of certain regions of Europe. Depending on the considered data set, 600 to 800 observations are available for 30-110 variables. Overall, the data are usually right-skewed, and therefore the algorithms were additionally run on the log-transformed data. Moreover, there are many equal values (especially for `bsstop`) which are caused by detection limit problems (see Reimann et al, 2008). Table 14 gives the deviations as returned by the different algorithms. Since the tolerance levels have been adjusted before, all methods perform similar, again with the exception NM. A comparison of runtimes in Table 15 shows that NLM is constantly the fastest approach, outperforming all other algorithms in terms of computation time, while maintaining the highest accuracy.

**Table 14** Deviations of $L_1$-median algorithms applied to real data sets.

| Data set | log | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| `bhorizon` | no | 1.46e+03 | 0 | 0 | 0 | 0 | 0 |
| | yes | 1.73 | 0 | 0 | 0 | 0 | 0 |
| `chorizon` | no | 4.99e+04 | 0 | 0 | 0 | 0 | 0 |
| | yes | 95.3 | 0 | 0 | 0 | 0 | 0 |
| `humus` | no | 501 | 0 | 0 | 0 | 2.33e-10 | 0 |
| | yes | 1 | 4.55e-13 | 0 | 0 | 0 | 0 |
| `moss` | no | 317 | 0 | 0 | 0 | 0 | 0 |
| | yes | 0.225 | 2.27e-13 | 0 | 0 | 0 | 0 |
| `bssbot` | no | 80.5 | 0 | 0 | 0 | 0 | 0 |
| | yes | 32.1 | 4.55e-13 | 0 | 0 | 0 | 0 |
| `bsstop` | no | 71.5 | 0 | 0 | 0 | 0 | 0 |
| | yes | 35.5 | 0 | 0 | 0 | 0 | 0 |

## 3.7 Runtime as a function of the sample size

In this subsection we report median runtimes as a function of the sample size $n$. Since the ranking of the different algorithms with respect to their runtime does not depend much on the actual data configuration, we report results for the uncorrelated data case, as in subsection 3.2. We generate from a multivariate normal distribution, with $p = 100$, and where $\log_{10}(n)$ varies from 2 to 5. Table 16 reports the median runtimes over 100 simulated samples. We see that the NLM algorithm is the fastest for all considered sample sizes. The runtimes suggest that the computational complexity of the different algorithms is $O(n \log n)$.

**Table 15** Runtimes [seconds] of the $L_1$-median algorithms applied to real data sets.

| Data set | log | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|---|
| `bhorizon` | no | 0.047 | 0.000 | 0.062 | 0.016 | 0.063 | 0.046 |
| | yes | 0.047 | 0.031 | 0.125 | 0.016 | 0.016 | 0.016 |
| `chorizon` | no | 0.172 | 0.062 | 0.219 | 0.047 | 0.094 | 0.156 |
| | yes | 0.156 | 0.078 | 0.063 | 0.031 | 0.063 | 0.078 |
| `humus` | no | 0.031 | 0.016 | 0.016 | 0.000 | 0.047 | 0.046 |
| | yes | 0.046 | 0.016 | 0.094 | 0.016 | 0.031 | 0.016 |
| `moss` | no | 0.031 | 0.000 | 0.016 | 0.000 | 0.031 | 0.031 |
| | yes | 0.031 | 0.015 | 0.016 | 0.000 | 0.032 | 0.031 |
| `bssbot` | no | 0.109 | 0.016 | 0.047 | 0.016 | 0.078 | 0.094 |
| | yes | 0.109 | 0.031 | 0.031 | 0.016 | 0.031 | 0.047 |
| `bsstop` | no | 0.094 | 0.015 | 0.047 | 0.016 | 0.063 | 0.047 |
| | yes | 0.094 | 0.062 | 0.328 | 0.015 | 0.047 | 0.032 |

**Table 16** Median runtimes [seconds] of the $L_1$-median algorithms applied to uncorrelated data with varying sample size $n$ and fixed number of variables $p = 100$.

| $n$ | NM | BFGS | CG | NLM | VaZh | HoCr |
|---|---|---|---|---|---|---|
| 100 | 0.017 | 0.012 | 0.005 | 0.006 | 0.012 | 0.009 |
| 1000 | 0.239 | 0.117 | 0.064 | 0.071 | 0.122 | 0.147 |
| 10000 | 2.335 | 1.739 | 10.204 | 0.694 | 1.336 | 2.577 |
| 1e+05 | 23.096 | 15.179 | 141.382 | 7.121 | 13.361 | 30.646 |

## 4 Conclusions

For computing the $L_1$-median, several specific algorithms can be found in the literature. In addition, the solution can also be found by general optimization routines, which are widely available in modern software packages. A fair comparison of these approaches is only possible by a unified software implementation. This has been done in the R-library `pcaPP` (version 1.8-1), using C++ code for the implementation. The implemented algorithms have been tested for various data configurations. Optimization based on NLM leads to the smallest value of the target function in all considered settings, which makes it the first choice as an appropriate algorithm for computing the $L_1$-median. Moreover, the computation time of this approach turned out to be to a large extent unaffected by the used distribution family, the outlier proportion added, the convergence criterion, and even by high correlation values, whereas other algorithms tend to have convergence issues in some simulation settings. In the tested situations NLM never showed any problems with convergence, and hence it provides a stable, fast and reliable approach, returning results with the highest precision among the tested algorithms.

Considering the tested data sets and their dimensions, it might be surprising that the absolute runtime of the algorithms is very low. Thus, whenever a robust data center needs to be computed, the $L_1$-median is an attractive estimator as long as affine equivariance is not required. It is definitely more attractive than the component-wise median, which has the same breakdown point, but which is not orthogonal equivariant.

Concluding, we recommend the algorithm NLM, implemented as `l1median_NLM` in the R-library `pcaPP` (version 1.8-1), as this particular approach turned out to deliver results of highest precision in combination with very low computation time, widely unaffected by the underlying data structure.

**Acknowledgments**

We would like to thank Martin Mächler from the ETH-Zürich for providing different $L_1$-median algorithms in the R-library `robustX`.

## References

Bedall F, Zimmermann H (1979) As 143: The mediancentre. Applied Statistics 28:325–328

Chaudhuri P (1996) On a geometric notion of quantiles for multivariate data. J Amer Statist Assoc 91:862–872

Croux C, Ruiz-Gazen A (2005) High breakdown estimators for principal components: The projection-pursuit approach revisited. Journal of Multivariate Analysis 95:206–226

Croux C, Filzmoser P, Oliveira M (2007) Algorithms for projection-pursuit robust principal component analysis. Chemometrics and Intelligent Laboratory Systems 87:218–225

Debruyne M, Hubert M, Van Horebeek J (2010) Detecting influential observations in Kernel PCA. Computational Statistics & Data Analysis 54(12):3007–3019

Dennis J, Schnabel R (1983) Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice Hall, New Jersey

Fletcher R, Reeves C (1964) Function minimization by conjugate gradients. Computer Journal 7:148–154

Gervini D (2008) Robust functional estimation using the median and spherical principal components. Biometrika 95:587–600

Gower J (1974) As 78: The mediancentre. Journal of the Royal Statistical Society 23:466–470

Hössjer O, Croux C (1995) Generalizing univariate signed rank statistics for testing and estimating a multivariate location parameter. Nonparametric Statistics 4:293–308

Lopuhaä H, Rousseeuw P (1991) Breakdown points of affine equivariant estimators of multivariate location and covariance matrices. The Annals of Statistics 19(1):229–248

Nelder J, Mead R (1965) A simplex algorithm for function minimization. Computer Journal 7:308–313

Nocedal J, Wright S (2006) Numerical Optimization, 2nd edn. Springer-Verlag, New York

R Development Core Team (2010) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL http://www.R-project.org, ISBN 3-900051-07-0

Reimann C, Filzmoser P, Garrett R, Dutter R (2008) Statistical data analysis explained. Applied environmental statistics with R. John Wiley, Chichester

Schnabel R, Koontz J, Weiss B (1985) A modular system of algorithms for unconstrained minimization. ACM Trans Math Software 11:419–440

Serneels S, Croux C, Filzmoser P, Van Espen P (2005) Partial robust M-regression. Chemometrics and Intelligent Laboratory Systems 79:55–64

Vardi Y, Zhang CH (2000) The multivariate $l_1$-median and associated data depth. Proc National Academy of Science 97(4):1423–1426

Weber A (1909) Über den Standort der Industrien. Mohr, Tübingen

Weiszfeld E (1937) Sur le point pour lequel la somme des distances de $n$ points donnés est minimum. Tôhoku Mathematical Journal 43:355–386