Methodological and Implementation Details on the Weighted BACON Algorithms

Tobias Schoch

University of Applied Sciences Northwestern Switzerland FHNW
School of Business, Riggenbachstrasse 16, CH-4600 Olten
tobias.schoch@fhnw.ch

March 25, 2021

Abstract. Billor et al. (2000, Comput. Stat. Data Anal.) proposed the BACON algorithms for multivariate outlier detection and robust linear regression. Béguin and Hulliger (2008, Surv. Methodol.) extended the outlier detection method to weighted and incomplete data problems. Both methods are implemented in the R packages, respectively, robustX and modi. We suggest a computationally efficient implementation in the C language. Efficiency is achieved by using a weighted quantile based on the Quicksort algorithm, partial sorting in place of full sorting, reuse of computed estimates, and most importantly an up-/downdating scheme for the Cholesky and QR factorizations. The computational costs of up-/downdating are far less than recomputing the entire decomposition repeatedly.

MSC2020. 62D05, 62H12, 62J05.

1. Introduction

Outlier detection and robust regression are computationally hard problems. This is all the more true when the number of variables and observations grow rapidly. Among all candidate methods, the BACON (blocked adaptive computationally efficient outlier nominators) algorithm of Billor, Hadi, and Vellemann (2000) has favorable computational characteristics as it requires only a few model evaluation irrespective of the sample size. This makes it a superior algorithm for big data applications.

The BACON algorithms for multivariate outliers detection and robust linear regression are implemented in the R package robustX (Maechler, Stahel, Turner, Oetliker, and Schoch, 2021). The algorithms do not take the sampling weights into account. The multivariate outlier detection method of Béguin and Hulliger (2008) that is capable of dealing with sampling weights and missing values can be found in the R package modi (Hulliger and Sterchi, 2020). Both implementations are written in the R statistical software (R Development Core Team, 2020).

In methodological terms, the BACON algorithms consist of the application of series of straightforward statistical estimation methods like coordinate-wise means, covariance matrix, Mahalanobis distances, or least squares regression on subsets of the data. A naive implementation would call the estimation methods iteratively on a sequence of growing subsets of the data without bothering too much with re-using or updating existing blocks of data. This leads to an excessively large number of copy/ modify operations and (unnecessary) re-computations. Altogether, the implementation will be computationally inefficient.

In this paper, we discuss the methodological details of a computationally efficient implementation of BACON algorithms. The techniques used to achieve this are (to name a few):

- an implementation of the weighted quantile based on the C.A.R. Hoare Select (FIND, Quicksort) algorithm with Bentley–McIlroy partitioning,
- a partial sorting device (based on Quicksort),
- reuse of computed estimates,
- up-/downdating Cholesky and QR factorizations.

The computational costs of the up-/downdating schemes for the Cholesky and QR factorizations are far less than recomputing the entire decomposition repeatedly.

The functions are implemented in the `C` language with an API for the `R` statistical software. In comparison with the existing implementations in the `R` software, our implementations is better suited for very large datasets.

The remainder of the paper is organized as follows. Section 2 presents the BACON algorithms in a nutshell. The BACON algorithm for multivariate outlier detection is studied in more detail in Section 3. The BACON algorithm for robust linear regression is studied in Section 4. In Appendix A, the weighted quantile and partial sorting device are documented.

2. The BACON algorithms in a nutshell

Suppose that the data at hand are n observations on p real-valued variables, $p < n$. The data are represented as the $(n \times p)$ matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$, $\boldsymbol{x}_i \in \mathbb{R}^p$, and are known to be contaminated by outliers. But it is not known which observations are outliers and how many observations are outliers.

Let us fix some notation. Denote by $\mathscr{S} = \{1, \ldots, n\}$ the ordered set of row indices of $\boldsymbol{X}$. Fix a set S such that $S \subseteq \mathscr{S}$. We write $\boldsymbol{X}|_S$ to mean the row-wise restriction of $\boldsymbol{X}$ to the rows indexed by the elements of set S. For instance, let $S = \{1, 3\}$; then $\boldsymbol{X}|_S$ is the $(2 \times p)$ matrix that consists of the rows 1 and 3 of $\boldsymbol{X}$. The complement of S is denoted by S^c. The cardinality of a set S is denoted by $|S|$. For ease of notation, we write $\boldsymbol{X}|_S^T$ instead of $(\boldsymbol{X}|_S)^T$ for the transpose of the restricted matrix.

2.1. Multivariate outlier detection

Following Billor et al. (2000), the BACON algorithm for multivariate outlier detection consists of two algorithms (called Algorithm 2 and 3), which are applied after another.

Algorithm 2. The BACON algorithm is initialized by the computation of the center $\boldsymbol{c}$ of the data; see left panel in Fig. 1; there, we have $\boldsymbol{c} = (c_1, c_2)^T$. In order to achieve good overall robustness, the center $\boldsymbol{c}$ is computed as the component-wise median (Billor et al., 2000, see "Version 2" of Algorihm 2). Next, the distances $d_i = \|\boldsymbol{x}_i - \boldsymbol{c}\|_2$ about the center are computed for all $i = 1, \ldots, n$, where $\|\cdot\|_2$

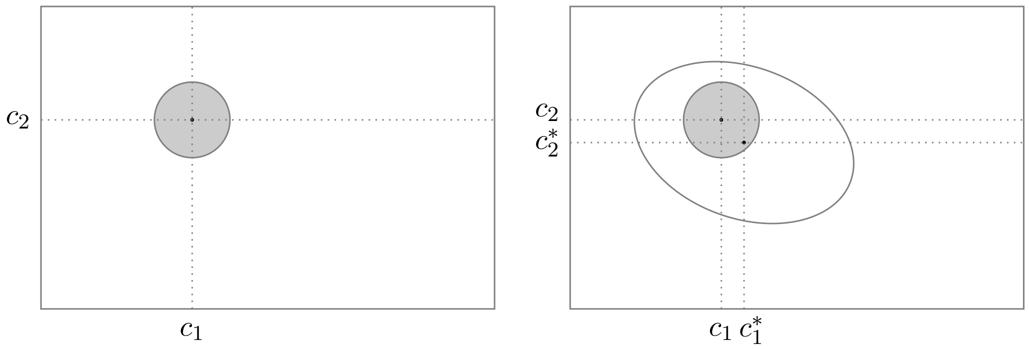

Figure 1: Schematic illustration

denotes the Euclidean norm. Then, we select the m observations with the smallest d_i's into the initial basic subset S, where $m = cp$ and $\{c \in \mathbb{N} : c < \lfloor n/p \rfloor\}$ is a tuning constant chosen by the user.

Algorithm 3.

Step 1) For $\boldsymbol{X}|_S$, we compute

- the component-wise arithmetic mean $\boldsymbol{\mu}_S$;
- the covariance/ scatter matrix $\boldsymbol{\Sigma}_S$;
- if Σ_S is singular, we keep adding observations to the subset S until Σ_S is nonsingular. The observations to be added are taken from the pool of the observations in the set $\mathscr{S} \setminus S$; in particular, we add those observations with the smallest d_i's.

Step 2) For all $i = 1, \ldots, n$, compute the Mahalanobis distances

$$d_i = \sqrt{(\boldsymbol{x}_i - \boldsymbol{\mu}_S)^T \boldsymbol{\Sigma}_S^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_S)} \tag{1}$$

and select all observations into the new subset S^* (see right panel in Fig. 1) whose Mahalanobis distances d_i are smaller than the criterion $c_{np}\chi^2_{\alpha,p}$, where $\chi^2_{\alpha,p}$ is the $1-\alpha$ quantile of the chi-square distribution with p degrees of freedom, and

$$c_{np} = 1 + \frac{p+1}{n-p} + \frac{2}{n-1-3p}. \tag{2}$$

Step 3) If $S = S^*$ we terminate the updating scheme; otherwise, we let $S \leftarrow S^*$ and jump to Step 1).

Remarks.

i) Upon termination, the set of outliers is given by $\mathscr{S} \setminus S^*$.

ii) The above algorithm generates a sequence of subsets, say, $\{S_i : i = 0, 1, \ldots\}$. The last subset in the sequence is the final subset of "outlier-free" observations. It is important to note that the subsets in the sequence are *not nested*; i.e., for any i, it is not guaranteed that $S_i \subset S_{i+1}$ (although eventually it will happen that S_{i+1} is equal to S_i; hence, the algorithm terminates).

iii) The algorithm is initialized at the center $\boldsymbol{c}$, which is computed as the component-wise median (cf. "Version 2" of Algorithm 2). As a consequence, the estimators of location and scatter are not affine equivariant; still, this proposal leads to nearly affine equivariant estimators (Billor et al., 2000). An estimator T is affine equivariant if and only if

$$T(\boldsymbol{A}\boldsymbol{X} + \boldsymbol{b}) = \boldsymbol{A}T(\boldsymbol{X}) + \boldsymbol{b},$$

for any nonsingular $(m \times n)$ matrix $\boldsymbol{A}$ and any n-vector $\boldsymbol{b}$.

iv) "Version 1" of Algorithm 2 of Billor et al. (2000) is affine equivariant by design as it takes the component-wise arithmetic means as $\boldsymbol{c}$. But this choice has a considerably lower breakdown point.

v) The breakdown point of "Version 2" of the BACON algorithm is approximately 40% (Billor et al., 2000).

vi) Béguin and Hulliger (2008) generalized the BACON algorithms for outlier detection to account for sampling weights (survey data) and missing values.

2.2. Robust linear regression

Denote by $\boldsymbol{X}$ the $(n \times p)$ design matrix with full column rank p $(p < n)$. The response variable is written as the (column) n-vector $\boldsymbol{y}$. We want to compute the least squares estimator $\boldsymbol{\beta} \in \mathbb{R}^p$

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

(Note: we will introduce the sampling weights later).

In the presence of outliers in $\boldsymbol{X}$ *and/or* $\boldsymbol{y}$, the least squares methods is (heavily) biased and/or inefficient as an estimator of the population regression parameter. Therefore, Billor et al. (2000) proposed to search for a subset S that is outlier-free and then to consider estimating $\boldsymbol{\beta}_S$, which is defined as

$$\boldsymbol{\beta}_S = (\boldsymbol{X}|_S^T\boldsymbol{X}|_S)^{-1}\boldsymbol{X}|_S^T\boldsymbol{y}|_S. \tag{3}$$

Following Billor et al. (2000), the BACON robust linear regression method consists of Algorithm 4 and 5, which are applied after another. The two algorithms are sketched subsequently.

Algorithm 4.

Step 1) Apply Algorithm 3 to the $\boldsymbol{X}$ data to obtain the subset S of outlier-free observations (having removed the column of $\boldsymbol{X}$ that contains the regression constant, if there is a constant). If $\mathrm{rank}(\boldsymbol{X}|_S) \neq p$, we keep adding observations to S until $\boldsymbol{X}|_S$ is of full rank. The observations to be added are taken from the pool of the observations in the set $\mathscr{S} \setminus S$, whose Mahalanobis distances are smallest.

Step 2) Solve (3) for $\boldsymbol{\beta}_S$, and compute the residual scale $\sigma_S = \|\boldsymbol{r}^T\boldsymbol{r}\|_2/(\|S\| - p)$, where $\boldsymbol{r} = \boldsymbol{y}|_S -$

$\boldsymbol{X}|_S\boldsymbol{\beta}_S$ is the least squares residual. Compute $\boldsymbol{t}_S = (t_1, \ldots, t_n)^T$, where

$$t_i = \begin{cases} \dfrac{y_i - \boldsymbol{x}_i^T\boldsymbol{\beta}_S}{\sigma_S\sqrt{1 - \boldsymbol{x}_i^T(\boldsymbol{X}^T|_S\boldsymbol{X}|_S)^{-1}\boldsymbol{x}_i}} & \text{if } i \in S, \\ \dfrac{y_i + \boldsymbol{x}_i^T\boldsymbol{\beta}_S}{\sigma_S\sqrt{1 + \boldsymbol{x}_i^T(\boldsymbol{X}^T|_S\boldsymbol{X}|_S)^{-1}\boldsymbol{x}_i}} & \text{otherwise.} \end{cases} \tag{4}$$

Note. On the subset S, t_i is the scaled (absolute) least squares residual, whereas on the set $\mathscr{S} \setminus S$, t_i is the scaled (absolute) prediction error.

Step 3) Let $k \leftarrow p + 1$

Step 4) Select the k observations whose t_i's are smallest (in absolute value) into the subset S. If $\operatorname{rank}(\boldsymbol{X}|_S) \neq p$, we keep adding observations from $\mathscr{S} \setminus S$ (with the smallest t_i's) to S until $\boldsymbol{X}|_S$ is of full rank. The set S is called the initial basic subset.

Step 5) If $k \leq m$, let $k \leftarrow k + 1$ and go to Step 4); otherwise terminate.

Algorithm 4 generates a sequence of subsets, say, $\{S_i : 0 = 1, \ldots\}$. It is important to note that the subsets in the sequence are *not* nested.

Algorithm 5.

Step 1) Use Algorithm 4 to select a subset S_0 of size $m = c \cdot p$, where the constant c can be chosen by the user; Billor et al. (2000) recommend a value of 4 or 5.

Step 2) For S_0, compute the t_i's in (4) and select a new subset, say, S_1 that consists of all observations whose t_i's are (in absolute value) smaller than the $\alpha/2(|S_1| + 1)$ quantile of the Student t-distribution with $|S_1| - p$ degrees of freedom, formally

$$t_{\alpha/2(|S_1|+1),\, |S_1|-p}. \tag{5}$$

Step 3) If $S_0 \neq S_1$, let $S_1 \leftarrow S_0$ and go to Step 2); otherwise terminate.

Remark. Upon termination, Algorithm 5 provides the robust estimate $\boldsymbol{\beta}_S$ of the population regression parameter $\boldsymbol{\beta}$, the regression scale estimate σ_S, and the subset of outlier-free observations.

3. Weighted BACON algorithm

In this section, we study the *weighted* BACON algorithm for multivariate outlier detection and robust estimation of the center and the covariance matrix.

3.1. Location and scatter

Let $S \subseteq \mathscr{S}$. Denote the weighted column means of $\boldsymbol{X}|_S$ (Hajek estimator) by

$$\boldsymbol{\mu}_S = \frac{1}{W_S} \sum_{i \in S} w_i \boldsymbol{x}_i, \qquad \text{where} \quad W_S = \sum_{i \in S} w_i, \tag{6}$$

and define the matrix $\boldsymbol{Z}_S$ (which is equal to $\boldsymbol{X}|_S$ centered or shifted by $\boldsymbol{\mu}_S$ and appropriately scaled)

$$\boldsymbol{Z}_S = \sqrt{\frac{\boldsymbol{w}|_S}{W_S - 1}} \circ \left(\boldsymbol{X}|_S - \mathbf{1}\boldsymbol{c}_S^T\right), \tag{7}$$

where $\mathbf{1}$ is the vector of ones (of size $|S|$), $\circ$ denotes the Hadamard product, and $\sqrt{\cdot}$ is applied element by element. Note that the Gramian matrix $\boldsymbol{Z}_S^T \boldsymbol{Z}_S$ is equal to the scatter/ covariance matrix

$$\boldsymbol{Z}_S \boldsymbol{Z}_S^T = \frac{1}{W_S - 1} \sum_{i \in S} w_i (\boldsymbol{x}_i - \boldsymbol{\mu}_S)(\boldsymbol{x}_i - \boldsymbol{\mu}_S)^T =: \boldsymbol{\Sigma}_S. \tag{8}$$

3.2. Mahalanobis distance

The scatter matrix $\boldsymbol{\Sigma}_S$ is required to be nonsingular, for otherwise we cannot compute the Mahalanobis distances in (1). There are several ways to check whether $\boldsymbol{\Sigma}_S$ is nonsingular. We prefer a method that is computationally cheap for the following reason. If $\boldsymbol{\Sigma}_S$ appears to be singular, we stop the computations on the current subset. Then, we keep adding observations to the set S until $\boldsymbol{\Sigma}_S$ is nonsingular. Because the computational costs associated with growing the set S are so small, it is not economical putting too much effort into a sophisticated method to check whether the scatter matrix is singular.

We adopt a two-stage approach.

(1) First, we count the number of positive elements on the diagonal of $\boldsymbol{\Sigma}_S$ (in floating-point arithmetic terms),

$$\widehat{r}_{\text{pd}} = \sum_{i=1}^{p} \mathbb{1}\big\{(s_{ii}) > \epsilon\big\}, \qquad (s_{ij}) \equiv \Sigma_S, \tag{9}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function, and ϵ is the machine epsilon (double precision). If $\widehat{r}_{\text{pd}} \neq p$, the computations are stopped and we switch to the process of enlarging the subset S until $\widehat{r}_{pd} = p$.

This approach is very effective as it catches the most common case of nonsingularity (non positive-definiteness) while its computational costs are negligible. To see this, suppose a subset S such that one column (variable) of $\boldsymbol{X}|_S$ is constant; hence, the variance is zero (e.g. grouped data), which implies that $\boldsymbol{\Sigma}_S$ is singular.

(2) In the second step, we compute the factorization

$$\boldsymbol{\Sigma}_S = \boldsymbol{L}_S \boldsymbol{L}_S^T. \tag{10}$$

If $\boldsymbol{\Sigma}_S$ is positive definite, the factorization in (10) is the (unique) Cholesky decomposition, where $\boldsymbol{L}$ is a lower triangular matrix with positive diagonal elements. If, however, $\boldsymbol{\Sigma}_S$ is positive *semi-*

definite it still has a decomposition of this form but the diagonal elements of $\boldsymbol{L}$ can be zero; see e.g. Golub and van Loan (1996, Chap. 4.2.8). Now, our approach is the following.

a) We compute the Cholesky decomposition in (10) using the `LAPACK: dpotrf` subroutine (Anderson et al., 1999).

b) If $\boldsymbol{\Sigma}_S$ is indeed positive semi-definite, the Cholesky decomposition can (or will) break down because a zero (or negative) pivot is encountered at some stage of the factorization. The subroutine `dpotrf` has an error flag (see argument `INFO`) that indicates when the factorization could not be completed because a leading minor of the matrix is not positive definite. If this flag has been raised, $\boldsymbol{\Sigma}_S$ is regarded as singular and we switch to the process of enlarging the subset S until $\boldsymbol{\Sigma}_S$ is nonsingular.

c) Relying on the error flag of `dpotrf` alone is too optimistic. Therefore, we also compute an estimate of the number of positive diagonal elements of $\boldsymbol{L}_S$,

$$\widehat{r} = \sum_{i=1}^{p} \mathbb{1}\big\{(l_{ii}) > \delta\big\}, \qquad (l_{ij}) \equiv \boldsymbol{L}_S, \tag{11}$$

where δ is a numerical constant. We pick a rather conservative choice for δ, e.g., $\delta = \epsilon^{1/4}$, where ϵ is the machine epsilon (double precision). If $\widehat{r} \neq p$, $\boldsymbol{\Sigma}_S$ is regarded as singular and we switch to the process of enlarging the subset S until $\boldsymbol{\Sigma}_S$ is nonsingular.

Remarks.

i) Our two-stage approach is not "fully waterproof" but it is computationally inexpensive.

ii) In place of the two-stage approach, we could determine the numerical rank of $\boldsymbol{Z}_S$ by the singular value decomposition (SVD). That is, the numerical rank $\widehat{r}$ is computed as the largest integer in $(0, \ldots, p)$ for which $\sigma_r \geq n\delta\sigma_1$, where δ is a tolerance criterion (e.g. $\delta = 1 \cdot 10^{-16}$) and $\sigma_1 \geq \cdots \geq \sigma_p$ are the singular values (Golub and van Loan, 1996, Chap. 2.5.5). Alternatively, we could use a rank-revealing Cholesky factorization with complete column pivoting (`LAPACK: dpstrf`, Anderson et al., 1999) of $\boldsymbol{\Sigma}_S$ to determine its numerical rank. However, both approaches are computationally quite expensive. Another approach would be to check whether $\boldsymbol{\Sigma}_S$ is positive definite by checking if all of its eigenvalues of are positive (in exact arithmetic). In floating-point arithmetic, we compute the eigenvalues (`LAPACK: dsyev`, Anderson et al., 1999) and then proceed as in the SVD-based. However, this approach is computationally still quite expensive.

If $\boldsymbol{\Sigma}_S$ is nonsingular, we solve the triangular system of linear equations

$$\boldsymbol{L}_S \boldsymbol{A}_S = \boldsymbol{Z}_S \tag{12}$$

for the $(n \times p)$ matrix $\boldsymbol{A}_S$ by forward substitution (`BLAS: dtrsm`, Blackford et al., 2002), where $\boldsymbol{L}_S$ and $\boldsymbol{Z}_S$ are defined in, respectively, (10) and (7). The Mahalanobis distance in (1) can be efficiently computed (for all $i = 1, \dots, n$) by

$$d_i = \sqrt{\sum_{j=1}^{p} (a_{ij})^2}, \qquad (a_{ij}) \equiv \boldsymbol{A}_S. \tag{13}$$

3.3. Algorithms

The following display shows pseudo-code of a weighted variant of Algorithm 2 of Billor et al. (2000).

Algorithm 2.

```
Require: $\boldsymbol{X}, \boldsymbol{w}, m$
$\boldsymbol{\zeta} \leftarrow$ WEIGHTED_MEDIAN$(\boldsymbol{X}, \boldsymbol{w})$        ▷ component-wise weighted median
$\boldsymbol{d} \leftarrow (d_1, \dots, d_n)^T$, where $d_i = \|\boldsymbol{x}_i - \boldsymbol{\zeta}\|_2$
$S \leftarrow$ SELECT_SUBSET$(\boldsymbol{d}, m)$        ▷ select the set with the $m$ smallest $d_i$'s
while $m < n$ do
    $\boldsymbol{\mu}_S \leftarrow$ WEIGHTED_MEAN $(\boldsymbol{X}|_S, \boldsymbol{w}|_S)$        ▷ Eq. (6)
    $\boldsymbol{\Sigma}_S \leftarrow$ WEIGHTED_SCATTER $(\boldsymbol{X}|_S, \boldsymbol{w}|_S, \boldsymbol{\mu}_S)$        ▷ Eqs. (7) and (8)
    if $\widehat{r}_{\mathrm{pd}} = p$ then        ▷ Eq. (9)
        $\boldsymbol{L}_S \leftarrow$ CHOLESKY_DECOMPOSITION $(\boldsymbol{\Sigma}_S)$        ▷ Eq. (10)
        if $\widehat{r} = p$ then        ▷ Eq. (11)
            break
        end if
    end if
    $m \leftarrow m + 1$        ▷ add obs. to the subset
    $S \leftarrow S \cup$ INDEX$(\boldsymbol{d}[m])$        ▷ INDEX returns the indices
end while
return $S, m$        ▷ return initial basic subset and its size
```

Remarks.

i) WEIGHTED_MEDIAN$(\boldsymbol{X}, \boldsymbol{w})$ computes the weighted median for each column of $\boldsymbol{X}$. The weighted median is implemented as a weighted Select (FIND, Quickselect) algorithm; see Appendix A.

ii) SELECT_SUBSET$(\boldsymbol{d}, m)$ partially sorts the elements of $\boldsymbol{d}$ such that the first m elements are in their final (sorted) position. The indices of the first m elements are selected into the subset, which is returned; see Appendix A for more details.

iii) In the `while` loop, we keep adding observations to the subset until the scatter matrix $\boldsymbol{\Sigma}_S$ is nonsingular.

The following display shows pseudo-code of a weighted variant of Algorithm 3 of Billor et al. (2000).

Algorithm 3.

```
Require: $\boldsymbol{X}$, $\boldsymbol{w}$, $S$, $m$ from ALGORITHM 2      ▷ initial basic subset and its size
 1: $S_1 \leftarrow \{\}$                                                  ▷ initialize $S_1$ as the empty set
 2: while $m < n$ do
 3:     $\boldsymbol{\mu}_S \leftarrow$ WEIGHTEED_MEAN $(\boldsymbol{X}|_S,\ \boldsymbol{w}|_S)$          ▷ Eq. (6)
 4:     $\boldsymbol{\Sigma}_S \leftarrow$ WEIGHTED_SCATTER $(\boldsymbol{X}|_S,\ \boldsymbol{w}|_S)$      ▷ Eqs. (7) and (8)
 5:     $\boldsymbol{L}_S \leftarrow$ CHOLESKY_DECOMPOSITION$(\boldsymbol{\Sigma}_S)$                     ▷ Eq. (10)
 6:     $\boldsymbol{d}_S = (d_1, \dots, d_n)^T \leftarrow$ MAHALANOBIS_DISTANCE$\big(\boldsymbol{X}, \boldsymbol{L}, S\big)$
 7:     $S_1 \leftarrow$ INDEX $\big(\boldsymbol{d}_S < c_{np} \cdot \chi^2_{p,\alpha}\big)$                 ▷ new subset
 8:     $m \leftarrow |S_1|$
 9:     if $S = S_1$ then
10:         break
11:     end if
12:     $S \leftarrow S_1$
13: end while
14: return $\mu_S$, $\Sigma_S$, $S$, $m$
```

Remarks.

i) The return values of Algorithm 3 are the final subset S, its size m, the weighted mean $\boldsymbol{\mu}_S$, and the weighted scatter/covariance matrix $\boldsymbol{\Sigma}_S$ on the subset S.

ii) The function MAHALANOBIS_DISTANCE at Line 6 computes the Mahalanobis distances for all $i \in \mathscr{S}$; it solves (12) and then computes the d_i's defined in (13).

iii) The chi-square criterion $c_{np} \cdot \chi^2_{p,\alpha}$ at Line 7 is defined in (2).

4. Weighted BACON regression algorithm

Denote by $\boldsymbol{X}$ the $(n \times p)$ design matrix with full column rank p $(p < n)$. The response variable is written as the (column) n-vector $\boldsymbol{y}$. Fix S such that $S \subseteq \mathscr{S}$ and $|S| \geq p$. Consider the least squares (LS) estimator $\boldsymbol{\beta}_S \in \mathbb{R}^p$ which solves the normal equations

$$\boldsymbol{X}|_S^T\, \boldsymbol{X}|_S\, \boldsymbol{\beta}_S = \boldsymbol{X}|_S^T\, \boldsymbol{y}|_S. \tag{14}$$

Note. The weighted least squares estimator obtains by replacing $\widetilde{\boldsymbol{X}}|_S$ and $\widetilde{\boldsymbol{y}}|_S$ in (14) with, respectively, $\widetilde{\boldsymbol{X}}|_S = (\sqrt{\boldsymbol{w}} \circ \boldsymbol{X})|_S$ and $\widetilde{\boldsymbol{y}}|_S = (\sqrt{\boldsymbol{w}} \circ \boldsymbol{y})|_S$, where $\sqrt{\cdot}$ is applied element by element, and $\circ$ denotes the Hadamard product.

The solution of the normal equations in (14) is known to be numerically unstable (Golub and van Loan, 1996, Chap. 5.3). Therefore, we consider solving the LS problem by the QR factorization, which is stable but computationally rather expensive. Suppose that $\boldsymbol{X}|_S$ has full column rank. Define the "thin" QR factorization of $\boldsymbol{X}|_S$ (Golub and van Loan, 1996, Chap. 5.3)

$$\boldsymbol{X}|_S = \boldsymbol{Q}\boldsymbol{R} = \left(\boldsymbol{Q}_S^1,\ \boldsymbol{Q}_S^2\right) \begin{pmatrix} \boldsymbol{R}_S^1 \\ \boldsymbol{0}_S \end{pmatrix} = \boldsymbol{Q}_S^1 \boldsymbol{R}_S^1, \tag{15}$$

where $\boldsymbol{R}_S^1$ is an $(p \times p)$ upper triangular matrix and $\boldsymbol{Q}_S^1$ is an $(|S| \times p)$ orthogonal matrix; the matrices $\boldsymbol{0}_S$ and $\boldsymbol{Q}_S^2$ are of conformable size but of no further interest. The parameter $\boldsymbol{\beta}_S$ solves the triangular system

$$\boldsymbol{R}_S^1 \boldsymbol{\beta}_S = (\boldsymbol{Q}_S^1)^T \boldsymbol{y}|_S. \tag{16}$$

A key characteristic of the BACON algorithm for regression is that the subset S is enlarged over several steps. To see this, let the design matrix $\boldsymbol{X}$ be of dimension $n = 1\,000$ with $p = 4$ variables. In the first step (see Section 2), Algorithm 3 is called on $\boldsymbol{X}$. We suppose that the resulting initial subset is of size 700. In step 2, we apply Algorithm 4 to select $k \leftarrow p + 1$ observations with the smallest distances. Then, we keep growing $k \leftarrow k + 1$ as long as $k \leq m$, where $m = cp$ and c is typically chosen to be 4 or 5. Let's take $c = 5$. Then, we observe the following sequence of subset sizes

```
700, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
```

until we can start with Algorithm 5. The computation of the QR factorization for each instance along this sequence is computationally quite expensive. Fortunately, an updating scheme for the QR factorization is available such that we do not have to re-compute the entire factorization over and over. The computational costs of the updating scheme are far less than recomputing the entire decomposition.

4.1. Up- and dating schemes

We consider up- and downdating separately. For ease of reading, we study the un-weighted regression problem and point out what needs to be modified for the weighted problem.

4.1.1. Updating

Consider the subset S and the QR factorization of $\boldsymbol{X}|_S$ in (15) $\boldsymbol{X}|_S = \boldsymbol{Q}_S^1 \boldsymbol{R}_S^1$. Suppose that the subset S is enlarged by one element. To be specific, we shall assume that the kth element is added to the subset; hence, $S_+ = S \cup \{k\}$. The design matrix associated with the enlarged subset obtains by appending the kth row of $\boldsymbol{X}$ to $\boldsymbol{X}|_S$,

$$\boldsymbol{X}|_{S_+} = \begin{bmatrix} \boldsymbol{X}|_S \\ \boldsymbol{x}_k^T \end{bmatrix}.$$

Let $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_p$ denote Givens rotation matrices (i.e., planar rotation matrices); see e.g. Golub and van Loan (1996, Chap. 5.1.8). Premultiplication by a Givens rotation matrix amounts to a counterclockwise rotation. In particular, the rotation matrices can be determined such that

$$\boldsymbol{G}_1^T \cdots \boldsymbol{G}_p^T \boldsymbol{H} = \boldsymbol{R}_S^1 \tag{17}$$

is an upper triangular matrix; $\boldsymbol{H}$ is an upper Hessenberg matrix. It then follows that (Golub and van

Loan, 1996, Chap. 12.5.3) the QR factorization of $\boldsymbol{X}|_{S_+}$ is $\boldsymbol{X}|_{S_+} = \boldsymbol{Q}^1_{S_+}\boldsymbol{R}^1_{S_+}$, where

$$\boldsymbol{Q}^1_{S_+} = \mathrm{diag}\big(1, \boldsymbol{Q}^1_S\big)\boldsymbol{G}_1 \cdots \boldsymbol{G}_p. \tag{18}$$

In other words, the identities (17) and (18) describe a scheme for updating the matrices $\boldsymbol{R}^1_S$ and $\boldsymbol{Q}^1_S$ to get $\boldsymbol{R}^1_{S_+}$ and $\boldsymbol{Q}^1_{S_+}$. There exists a similar method to downdate the QR factorization (i.e., removing a row from $\boldsymbol{X}|_S$). The downdating scheme is more intricate as it can break down when the matrix becomes indefinite. We shall discuss this later.

Updating $\boldsymbol{R}^1_S$ is straightforward and inexpensive (order $p^2/2$ flops). In contrast, updating $\boldsymbol{Q}^1_S$ is more expensive (order n^2 flops). Therefore, we take a different approach. Our approach is based on the observation that $\boldsymbol{R}^1_S = \boldsymbol{L}^T_S$, where $\boldsymbol{L}_S$ is a lower triangular matrix, i.e. the Cholesky factor of the Gramian matrix $\boldsymbol{X}|^T_S\boldsymbol{X}|_S$. So, we initialize the regression estimator by the QR factorization, and then we switch to a Cholesky-based regression approach,

$$\boldsymbol{L}_S\,\boldsymbol{L}^T_S\,\boldsymbol{\beta}_S = \boldsymbol{X}|^T_S\,\boldsymbol{y}|_S \qquad \Longleftrightarrow \qquad \boldsymbol{L}_S\,\boldsymbol{u}_S = \boldsymbol{X}|^T_S\,\boldsymbol{y}|_S,$$

where $\boldsymbol{u}_S = \boldsymbol{L}^T_S\,\boldsymbol{\beta}_S$. For the Cholesky-based approach, we solve

$$\boldsymbol{\beta}_S \leftarrow \text{FORWARD\_SOLVE}\Big(\boldsymbol{L}_S,\ \text{FORWARD\_SOLVE}\big(\boldsymbol{L}_S, (X^T\boldsymbol{y})|_S\big)\Big). \tag{19}$$

The Cholesky regression approach is computationally less expensive than the QR aproach. Its flop counts is of order $p^2(n + p/3)$, whereas the QR algorithm requires $2p^2(n - p/3)$ flops; see e.g. Golub and van Loan (1996, Chap. 5.3).

For the Cholesky-based approach, the updating scheme is as follows (let $S_+ = S \cup \{k\}$). First, we compute, the rank-one update of $\boldsymbol{X}|^T_S\,\boldsymbol{y}|_S$,

$$\boldsymbol{X}|^T_{S_+}\,\boldsymbol{y}_{S_+} = \boldsymbol{X}|^T_S\,\boldsymbol{y}|_S + y_k\boldsymbol{x}^T_k. \tag{20}$$

For the weighted regression-problem, the r.h.s. has to be pre-multiplied by w_k. Second, the Cholesky factor $\boldsymbol{L}_S$ is updated by the following function (Stewart, 1998, p. 340).

```
function CHOL_UPDATE($(\boldsymbol{L}, \boldsymbol{x})$)
    for $i = 1, \dots, p$ do
        SETUP_ROTATION $(\boldsymbol{L}_S[i,i], \boldsymbol{x}[i], c, s)$
        APPLY_ROTATION $(\boldsymbol{L}_S[i, i+1:p], \boldsymbol{x}[i+1:p], c, s)$
    end for
end function
```

where $\boldsymbol{L}_S[i,j]$ denotes the element on the ith row and in the jth column of $\boldsymbol{L}_S$. The functions SETUP_ROTATION and APPLY_ROTATION are defined as follows (Stewart, 1998, Algorithms 1.6 and 1.7).

```
function SETUP_ROTATION$(a, b, c, s)$
    $\tau \leftarrow |a| + |b|$
    if $\tau \le \epsilon$ then
```

```
 4:         $c \leftarrow 1$;   $s \leftarrow 0$
 5:         **return**
 6:     **end if**
 7:     $\nu \leftarrow \tau\sqrt{(a/\tau)^2 + (b/\tau)^2}$
 8:     $c \leftarrow a/\nu$;   $s \leftarrow b/\nu$
 9:     $a \leftarrow \nu$;   $b \leftarrow 0$
10: **end function**
```

```
**function** APPLY_ROTATION $(c, s, \boldsymbol{x}, \boldsymbol{y})$
    $\boldsymbol{t} \leftarrow c\boldsymbol{x} + s\boldsymbol{y}$
    $\boldsymbol{y} \leftarrow c\boldsymbol{y} - s\boldsymbol{x}$
    $\boldsymbol{x} \leftarrow \boldsymbol{t}$
**end function**
```

Remarks.

i) The scaling factor τ in function SETUP_ROTATION is introduced to avoid overflows and make underflows harmless; see Stewart (1998, p. 273) and Golub and van Loan (1996, Chap. 5.1.8).

ii) The C library math.h provides (since standard C99) the dedicated function hypot(x,y) for the computation of $\sqrt{x^2 + y^2}$ (see Line 7 of SETUP_ROTATION) without undue overflow or underflow at intermediate stages of the computation.

iii) The complexity of function CHOL_UPDATE is of order $p^2/2$ flops (Stewart, 1998, p. 340).

4.1.2. Downdating scheme

The downdating scheme is more intricate as it can break down when the matrix becomes indefinite. Let $S_- = S \setminus \{k\}$. The rank-one downdate of $\boldsymbol{X}|_S^T\,\boldsymbol{y}|_S$ is unproblematic and is given by

$$\boldsymbol{X}|_{S_-}^T\,\boldsymbol{y}_{S_-} = \boldsymbol{X}|_S^T\,\boldsymbol{y}|_S - y_k\boldsymbol{x}_k^T. \tag{21}$$

There exist three candidate algorithms for downdating the Cholesky factor $\boldsymbol{L}$ (Stewart, 1998, p. 355): Saunder‘s method, the methods of mixed rotation, and the methods of hyperbolic rotations. We use the method of mixed rotations, an implementation of which is the following algorithm (Stewart, 1998, Algorithm 3.9).

```
**function** CHOL_DOWNDATE$(\boldsymbol{L}, \boldsymbol{x})$
    **for** $i = 1, \ldots, p$ **do**
        $a \leftarrow \boldsymbol{L}[i,i]^2 - \boldsymbol{x}[i]^2$
        **if** $a < \epsilon$ **then**
            **return** Error
        **else**
            $b \leftarrow \sqrt{a}$
```

```
 8:         end if
 9:         $c \leftarrow b/\boldsymbol{L}[i,i]$
10:         $s \leftarrow \boldsymbol{x}[i]/\boldsymbol{L}[i,i]$
11:         $\boldsymbol{L}[i,i] \leftarrow b$
12:         $\boldsymbol{L}[i,i+1:p] \leftarrow \big(\boldsymbol{L}[i,i+1:p] - s\boldsymbol{x}[i+1:p]\big)/c$
13:         $\boldsymbol{x}[i+1:p] \leftarrow c\boldsymbol{x}[i+1:p] - s\boldsymbol{L}[i,i+1:p]$
14:     end for
15: end function
```

Remarks.

1. The constant ϵ (see Line 5 in CHOL_DOWNDATE) is taken to be the machine double epsilon.

2. The function CHOL_DOWNDATE returns an ERROR if downdating is not feasible (see line 5). This happens when the matrix $\boldsymbol{L}^T\boldsymbol{L} - \boldsymbol{x}\boldsymbol{x}^T$ associated with downdating is not positive definite.

3. It might be thought that the appearance of a small c leads to numerical instability (see Line 12). But this is not the case as Stewart (1998, p. 346) shows, unless the problem is itself ill-conditioned.

4. Stewart (1998, p. 352) shows that the downdating scheme used in CHOL_DOWNDATE has some nice numerical properties; in particular, it is relationally stable (whereas the method of hyperbolic rotations is not).

5. The order of flops count of the functions CHOL_DOWNDATE and CHOL_UPDATE is the same (Stewart, 1998, p. 346).

4.1.3. Application of the up- and downdating schemes

The functions CHOL_UPDATE and CHOL_DOWNDATE compute an update of the Cholesky factor when one row of the design matrix is added or removed. Let S_0 and S_1 be subsets. The following function (where the weights array has been suppressed for ease of reading) takes care of all up-/ and downdates that result when we transition from set S_0 to set S_1. It returns up-/downdates of $\boldsymbol{L}_S$ and $\boldsymbol{X}_S^T\boldsymbol{y}_S$.

```
function UPDATE $\big(\boldsymbol{L}_{S_0}, \boldsymbol{X}, \boldsymbol{y}, S_0, S_1\big)$
    $U \leftarrow S_0 \setminus S_1 \neq \{\}$                                              ▷ identify updates
    $D \leftarrow S_1 \setminus S_0 \neq \{\}$                                              ▷ identify downdates
    for $u \in U$ do
        $\boldsymbol{L}_{S_1} \leftarrow$ CHOL_UPDATE $\big(\boldsymbol{L}_{S_0}, \boldsymbol{X}|_{S_0 \setminus S_1}\big)$
        $(\boldsymbol{X}^T\boldsymbol{y})|_{S_1} \leftarrow (\boldsymbol{X}^T\boldsymbol{y})|_{S_0} + (\boldsymbol{X}^T\boldsymbol{y})|_{S_0 \setminus S_1}$                  ▷ Eq. (20)
    end for
    for $u \in U$ do
        $\boldsymbol{L}_{S_1} \leftarrow$ CHOL_DOWNDATE $\big(\boldsymbol{L}_{S_0}, \boldsymbol{X}|_{S_1 \setminus S_0}\big)$     ▷ returns ERROR if downdating breaks
```

```
            if ERROR then
                return ERROR
            end if
            $(\boldsymbol{X}^T\boldsymbol{y})|_{S_1} \leftarrow (\boldsymbol{X}^T\boldsymbol{y})|_{S_0} - (\boldsymbol{X}^T\boldsymbol{y})|_{S_1\setminus S_0}$        ▷ Eq. (21)
        end for
        return $\boldsymbol{L}_{S_1},\ (\boldsymbol{X}^T\boldsymbol{y})|_{S_1}$
    end function
```

Remark. The "mechanics" underlying the function UPDATE are trivial. But it is important that the updates are computed in the first place, followed by the downdates. Otherwise we would experience too many breakdowns of the downdating algorithm.

4.2. Residuals, "hat" matrix, and t_i's

Define the least squares residuals by

$$\boldsymbol{r}(\boldsymbol{\beta}_S) = \big(r_1(\boldsymbol{\beta}_S), \ldots, r_n(\boldsymbol{\beta}_S)\big)^T = \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_S \tag{22}$$

for all $i = 1, \ldots, n$ and let

$$\sigma_S = \frac{\|\, \boldsymbol{r}|_S(\boldsymbol{\beta}_S)\, \|_2}{\sqrt{|S| - p}} \tag{23}$$

denote the estimate of the residual scale (on the restriction). For the weighted regression, we have

$$\sigma_S = \frac{\|\, \widetilde{\boldsymbol{r}}|_S(\boldsymbol{\beta}_S)\, \|_2}{\sqrt{\sum_{i\in S} w_i - p}}, \tag{24}$$

where $\widetilde{\boldsymbol{r}}(\boldsymbol{\beta}_S) = \widetilde{\boldsymbol{y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\beta}_S$.

The "hat" matrix of the LS estimate, i.e., the orthogonal projection matrix onto the column space of $\boldsymbol{X}|_S$, is given by $\boldsymbol{H}_S = \boldsymbol{Q}_{S1}\boldsymbol{Q}_{S1}^T$. The diagonal elements of $\boldsymbol{H}_S$ are called leverages. The *extension* of the projection matrix onto column space of the entire matrix $\boldsymbol{X}$ is given by

$$\boldsymbol{H} = \boldsymbol{A}\boldsymbol{A}^T \qquad \text{with} \qquad \boldsymbol{A} \equiv (a_{ij}) = \boldsymbol{X}\boldsymbol{R}_{S1}^{-1},$$

and the "extended" leverages for all $1, \ldots, n$ observations are computed as

$$\boldsymbol{h} = (h_1, \ldots, h_n)^T = \mathrm{diag}(\boldsymbol{H}) = \sum_{j=1}^{p}(a_{ij})^2. \tag{25}$$

For the weighted regression, the weighted "hat" matrix is defined as (Li and Valliant, 2009)

$$\boldsymbol{h}_w = \mathrm{diag}\Big\{\boldsymbol{X}\big(\widetilde{\boldsymbol{X}}|_S^T\,\widetilde{\boldsymbol{X}}|_S\big)^{-1}\boldsymbol{X}^T\boldsymbol{W}\Big\}, \tag{26}$$

where $\boldsymbol{W} = \operatorname{diag}(\boldsymbol{w})$. It can be computed efficiently by

$$\boldsymbol{h}_w = \boldsymbol{h}_* \circ \boldsymbol{w}, \tag{27}$$

where $\boldsymbol{h}_*$ obtains from (25) with $(a_{ij}) \equiv \boldsymbol{A} = \boldsymbol{X}\widetilde{\boldsymbol{R}}_{S1}^{-1}$, where $\widetilde{\boldsymbol{R}}_{S1}$ is the $\boldsymbol{R}_1$ matrix of the "thin" QR factorization of $\widetilde{\boldsymbol{X}}|_S$.

The distances t_i of Billor et al. (2000, p. 288) – see also Eq. (4) – are computed for all $i = 1, \ldots, n$ by

$$t_i(\boldsymbol{\beta}_S) = \begin{cases} \dfrac{|r_i(\boldsymbol{\beta}_S)|}{\sigma_S\sqrt{1-h_i}} & \text{if } i \in S, \\[2ex] \dfrac{|r_i(\boldsymbol{\beta}_S)|}{\sigma_S\sqrt{1+h_i}} & \text{otherwise,} \end{cases} \tag{28}$$

where $r_i(\boldsymbol{\beta}_S)$ is defined in (22) and the h_i's are defined in (25). The following function computes the t_i's.

```
1:  function COMPUTE_TI($\boldsymbol{L}, \boldsymbol{X}|_S, \boldsymbol{X}\boldsymbol{y}, S, p$)
2:      $\boldsymbol{\beta}_S \leftarrow$ FORWARD_SOLVE $\big(\boldsymbol{L}_S$, FORWARD_SOLVE$(\boldsymbol{L}_S, (X^T\boldsymbol{y})|_S)\big)$    ▷ BLAS: dtrsm, Eq. (19)
3:      $\boldsymbol{r} \leftarrow \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_S$    ▷ LAPACK: dgemv
4:      $\sigma_S \leftarrow \big\|\boldsymbol{r}|_S^T(\boldsymbol{\beta}_S)\boldsymbol{r}|_S(\boldsymbol{\beta}_S)\big\|_2/\sqrt{|S|-p}$    ▷ Eq. (23)
5:      $\boldsymbol{L}_S^{-1} \leftarrow$ INVERT_TRIANGULAR_MATRIX($\boldsymbol{L}_S$)    ▷ LAPACK: dtrtri
6:      $\boldsymbol{A} \leftarrow \boldsymbol{L}_S^{-T}\boldsymbol{X}$    ▷ BLAS: dtrmm
7:      $\boldsymbol{h} \leftarrow \sum_{j=1,\ldots,p}(a_{ij}^2)$,    where $(a_{ij}) \equiv \boldsymbol{A}$    ▷ Eq. (25)
8:      $\boldsymbol{t} \leftarrow \big(t_1(\boldsymbol{\beta}_S), \ldots, t_n(\boldsymbol{\beta}_S)\big)^T$    ▷ Eq. (28)
9:      return $\boldsymbol{t}$
10: end function
```

Remark. For the weighted regression, Equations (23) and (25) referred to in the Lines 4 and 7 of COMPUTE_TI must be replaced by, respectively, (24) and (27).

4.3. Algorithms

The following display shows pseudo-code of a weighted variant of Algorithm 4 of Billor et al. (2000).

Algorithm 4.

Require: $\boldsymbol{d}_{\text{sort}}$

```
$\boldsymbol{t} \leftarrow$ COMPUTE_TI $\big(\boldsymbol{L}_{S_0}, \boldsymbol{X}|_{S_0}, \boldsymbol{X}, \boldsymbol{y}, S_0, p\big)$
$m \leftarrow p + 1$
$S_1 \leftarrow$ SELECT_SUBSET($\boldsymbol{d}_{\text{sort}}, m$)
while $|S_1| \leq c \cdot p$ do
```

```
    $\boldsymbol{L}_{S_1},\ (\boldsymbol{X}^T\boldsymbol{y})|_{S_1} \leftarrow$ UPDATE $\left(\boldsymbol{L}_{S_0}, (\boldsymbol{X}^T\boldsymbol{y})|_{S_0}, S_0, S_1\right)$    ▷ update Cholesky factor
    if rank$(\boldsymbol{L}_{S_1}) \neq p$ then    ▷ check for rank deficiency
        while $|S_1| < c \cdot p$ do
            $m \leftarrow m + 1$    ▷ add obs. to $S_1$
            $S_1 \leftarrow S_1 \cup$ INDEX $(\boldsymbol{d}_{\text{sort}}[m])$    ▷ INDEX returns the index
            $\boldsymbol{L}_{S_1},\ (\boldsymbol{X}^T\boldsymbol{y})|_{S_1} \leftarrow$ UPDATE $\left(\boldsymbol{L}_{S_0}, (\boldsymbol{X}^T\boldsymbol{y})|_{S_0}, S_0, S_1\right)$
            if rank$(\boldsymbol{L}_{S_1}) = p$ then
                break    ▷ stop adding obs.
            end if
        end while
    end if
    $\boldsymbol{t} \leftarrow$ COMPUTE_TI $\left(\boldsymbol{L}_{S_1}, \boldsymbol{X}|_{S_0}, \boldsymbol{X}, \boldsymbol{y}, S_1, p\right)$
    $S_1 \leftarrow$ SELECT_SUBSET$(\boldsymbol{t}, m)$    ▷ update the set
    $S_0 \leftarrow S_1; \quad m \leftarrow m + 1$    ▷ prepare the next iteration
end while
return $S, m$
```

Remarks.

1) For ease of reading, ALGORITHM 4 is displayed without the sampling weights.

2) The constant c (supplied by the user) determines the iterations of the while loop.

The following display shows pseudo-code of a weighted variant of Algorithm 5 of Billor et al. (2000).

Algorithm 5.

```
Require: $m$ and $S$ from ALGORITHM 4
$i \leftarrow 1; \quad S_1 \leftarrow \{\}$
while $i \leq$ maxiter do
    $(\boldsymbol{\beta}_S,\ \boldsymbol{L}_S^T) \leftarrow$ REGRESSION $(\boldsymbol{X}|_S, \boldsymbol{y}|_S)$    ▷ QR-based least squares, Eq. (16)
    $\boldsymbol{t} \leftarrow$ COMPUTE_TI $\left(\boldsymbol{L}_S, \boldsymbol{X}|_S, \boldsymbol{X}, \boldsymbol{y}, S, p\right)$
    $S_1 \leftarrow$ SELECT_SUBSET_WHERE $\left(\boldsymbol{t} < t_{\alpha/2(|S_1|+1),\,|S_1|-p}\right)$    ▷ Eq. (5)
    if $S_1 = S$ then
        break
    end if
    $i \leftarrow i + 1; \quad S \leftarrow S_1$    ▷ prepare the next iteration
end while
```

Remarks.

1) On return, ALGORITHM 5 yields a robust estimate of $\boldsymbol{\beta}$ and σ; and it returns the set S of outlier-free observations.

2) The REGRESSION function (see Line 3) is based on the QR factorization see (16).

3) For ease of reading, ALGORITHM 5 is displayed without the sampling weights. For the weighted regression problem, (i) the REGRESSION function in Line 3 is called with the arguments $\widetilde{X}|_S$ and $\widetilde{y}|_S$ in place of $X|_S$ and $y|_S$; and (ii) COMPUTE_TI must be adapted for the weights.

Appendix

A. Weighted quantile

There exists a large number of different definitions for unweighted sample quantiles. Hyndman and Fan (1996) discuss nine different definitions. We focus on their second definition, which corresponds to `type 2` in the `stats::quantile` function of the `R` statistical software. This definition averages over discontinuities of the inverse empirical distribution function.

Consider a sample of size n. Let $\boldsymbol{x}$ be an n-vector of real values, and denote by $x_{(i)}$ the ith order statistic of $\boldsymbol{x}$ (with array indexing: $1..n$). The `type 2` of the pth sample quantile can be written as

$$Q(p) = \begin{cases} x_{(1)} & \text{if } p = 0, \\ \frac{1}{2}\big(x_{(i)} + x_{(i+1)}\big) & \text{if } 0 < p < 1 \quad \text{and} \quad \text{frac}(np) = 0, \\ x_{(i+1)} & \text{if } 0 < p < 1 \quad \text{and} \quad \text{frac}(np) \neq 0, \\ x_{(n)} & \text{if } p = 1, \end{cases}$$

where $i = \lfloor pn \rfloor$ and $\text{frac}(x) = x - \lfloor x \rfloor$ denotes the fractional part of x.

Let $\boldsymbol{w}$ denote an n-vector of positive weights. Let $w_{(i)}$ denote the weight associated with the order statistic $x_{(i)}$. A weighted estimator of the pth population quantile is given by

$$Q_w(p) = \begin{cases} x_{(1)} & \text{if } w_{(1)} < pW, \\ \frac{1}{2}\big(x_{(i)} + x_{(i+1)}\big) & \text{if } \sum_{j=1}^{i} w_{(j)} = pW, \\ x_{(i+1)} & \text{if } \sum_{j=1}^{i} w_{(j)} < pW < \sum_{j=1}^{i+1} w_{(k)}, \end{cases}$$

where W is the total weight $W = \sum_{i=1}^{n} w_i$.

The function to compute $Q_w(p)$ is WQUANTILE, which is based on a weighted variant of C.A.R. Hoare's Quicksort/ Select (FIND) algorithm. Select differs from Quicksort in that it does not do a full sort. Instead it sorts only the partition of the data where the value to be selected lies. Quicksort/ Select has some desirable feature (Sedgewick, 1997, p. 303).

i) It is an in-place sorting device;

ii) Quicksort requires only time proportional to $n \log n$ for sorting an array of size n. Because Select does not do a full sort its time complexity is linear in n.

The drawbacks of Quicksort/ Select are (Sedgewick, 1997, p. 303).

i) The sort need not be stable (i.e. the order of equal elements is not preserved).

ii) It may take up to an order of n^2 operations in the worst case.

Gurwitz (1990) compared several implementations of the weighted median (partial heapsort, linear-time fast median, and Quicksort/ Select). He found that Quicksort/ Select was considerably faster than the other methods. This may come at some surprise since the linear-time fast median has (in theory) the best worst-case run time. However, the overhead associated with finding the median in subsamples slows the linear-time fast median down.

Some further remarks are in order.

- On arrays with many identical elements, Quicksort with the classical Lomuto or Hoare partitioning scheme may perform rather poorly. It can be substantially improved by using the 3-way partitioning scheme of Bentley and McIlroy (1993).
- For very small arrays, insertion sort is used because it has less overhead than Quicksort; see e.g. (Sedgewick, 1997, p. 316).
- The Quicksort algorithm is "easy to describe, and also easy to get wrong" (Bentley and McIlroy, 1993, p. 1252). In the words of Sedgewick (1997, p. 303) Quicksort "is fragile in the sense that a simple mistake in the implementation can go unnoticed and cause it to perform badly". Therefore, we follow the implementation of Bentley and McIlroy (1993) closely.

All functions use C style zero array indexing; $\boldsymbol{a}$ denotes the array of data and $\boldsymbol{w}$ is the array of weights (of the same dimension); $p \in [0, 1]$ determines the quantile of interest.

```
 1: function WQUANTILE($\boldsymbol{a}$, $\boldsymbol{w}$, $p$)
 2:     $n \leftarrow$LENGTH($\boldsymbol{a}$)
 3:     if p = 0 then
 4:         WSELECT0 ($\boldsymbol{a}$, $\boldsymbol{w}$, 0)                 ▷ select the smallest value
 5:         $q \leftarrow \boldsymbol{a}[0]$
 6:     else if p = 1 then
 7:         WSELECT0 ($\boldsymbol{a}$, $\boldsymbol{w}$, $n-1$)             ▷ select the largest value
 8:         $q \leftarrow \boldsymbol{a}[n-1]$
 9:     else
10:         WQUANT0 ($\boldsymbol{a}$, $\boldsymbol{w}$, 0, $n-1$, $p$, $q$)      ▷ compute weighted quantile
11:     end if
12:     return $q$
13: end function
```

Remarks.

i) The function WSELECT0 (see below) selects the kth largest element in the array (k is the last argument in the function call). The function does not return anything; instead, it sorts/ selects the kth element into is final sorted position. What remains to be done is the extraction of the respective element from array $\boldsymbol{a}$ (see lines 5 and 8).

ii) The function WQUANT0 is the workhorse function and is defined as follows. On exit, the function returns the result in argument q.

```
1: function WQUANT0($\boldsymbol{x}$, $\boldsymbol{w}$, $lo$, $hi$, $p$, $q$)
```

```
    if $lo \leq hi$ then
        $q \leftarrow \boldsymbol{x}[0]$                                    ▷ case: $n = 1$
        return
    end if
    if $hi - lo = 1$ then                                            ▷ case: $n = 2$
        if $(1 - p)\boldsymbol{w}[lo] = p\boldsymbol{w}[hi]$ then
            $q \leftarrow (\boldsymbol{x}[lo] + \boldsymbol{x}[hi])/2$
            return
        else if $(p - 1)\boldsymbol{w}[lo] > p\boldsymbol{w}[hi]$ then
            $q \leftarrow \boldsymbol{x}[lo]$
            return
        else
            $q \leftarrow \boldsymbol{x}[hi]$
            return
        end if
    end if
    if $hi - lo + 1 \leq$ _n_quickselect then
        $q \leftarrow$ INSERTIONSELECT($\boldsymbol{x}$, $\boldsymbol{w}$, $lo$, $hi$, $p$)       ▷ insertion sort
    end if
    $S \leftarrow \sum_{k=lo}^{hi} \boldsymbol{w}[k]$                                ▷ total weight
    $i, j \leftarrow 0$                                              ▷ initialize sentinels
    PARTITION_3WAY($\boldsymbol{x}$, $\boldsymbol{w}$, $lo$, $hi$, $i$, $j$)
    $S_{lo} \leftarrow \sum_{k=lo}^{j} \boldsymbol{w}[k], \quad S_{hi} \leftarrow \sum_{k=i}^{hi} \boldsymbol{w}[k]$      ▷ total weight by partition
    if $S_{lo} < pS$ AND $S_{hi} < (1 - p)S$ then                   ▷ termination criterion
        $q \leftarrow \boldsymbol{x}[j + 1]$
        return
    else
        if $(1 - p)S_{lo} > pS_{hi}$ then
            $\boldsymbol{w}[j + 1] \leftarrow S - S_{lo}$
            WQUANT0($\boldsymbol{x}, \boldsymbol{w}, lo, j + 1, p$)                 ▷ recursion: lower part
        else
            $\boldsymbol{w}[i - 1] \leftarrow S - S_{hi}$
            WQUANT0($\boldsymbol{x}, \boldsymbol{w}, i - 1, hi, p$)                 ▷ recursion: upper part
        end if
    end if
end function
```

Remarks.

i) The lines 6–17 implement the computation of the weighted quantile, which coincides with the type 2 quantile in Hyndman and Fan (1996) if all weights are equal.

ii) If the array has `_n_quickselect` elements or less, insertion sort is used; see lines 18–20.

iii) The function PARTITION_3WAY implements the Bentley–McIlroy partitioning scheme; see Bentley and McIlroy (1993) or Program 7.5 in Sedgewick (1997, p. 326). It takes the two sentinels, i and j as arguments and modifies them while partitioning. The two sentinels are required in the program (see line 24 and beyond) in order to compute the weight totals associcated with the partitions (this is a speciality of the weighted algorithm and is not part of the original Bentley–McIlroy implementation). The function PARTITION_3WAY calls the function CHOOSE_PIVOT [not shown] which computes the pivotal element by the median-of-three rule (see e.g. Sedgewick, 1997, Chap. 7.5) if the array has less than `_n_ninther` elements; otherwise the pivot is determined by Tukey's ninther (Bentley and McIlroy, 1993, cf.).

iv) From the lines 31 and 34 we see that the tail recursion only takes place on one partition. This is a key characteristic of the Select algorithm. In contrast, Quicksort uses tail recursion on both partitions simultaneously.

```
function WSELECT0($\boldsymbol{a}, \boldsymbol{w}, lo, hi, k$)
    if $hi \leq lo$ then
        return
    end if
    $i, j \leftarrow 0$                                          ▷ initialize sentinels
    PARTITION_3WAY($\boldsymbol{x}$, $\boldsymbol{w}$, $lo$, $hi$, $i$, $j$)
    if $k \leq j$ then
        WSELECT0($\boldsymbol{a}, \boldsymbol{w}, lo, j, k$)                 ▷ recursion: lower part
    else if $k \geq i$ then
        WSELECT0($\boldsymbol{a}, \boldsymbol{w}, i, hi, k$)                 ▷ recursion: upper part
    end if
end function
```

Remark. The function WSELECT0 is a one-to-one implementation of the Bentley–McIlroy type Quicksort/ Select algorithm except that it also selects/ sorts the array of weights along its way.

References

ANDERSON, E., Z. BAI, C. BISCHOF, L. S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. D. CROZ, A. GREENHAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN (1999): *LAPACK Users' Guide*, Philadelphia: Society for Industrial and Applied Mathematics (SIAM), 3rd ed.

BÉGUIN, C. AND B. HULLIGER (2008): "The BACON-EEM Algorithm for Multivariate Outlier Detection in Incomplete Survey Data," *Survey Methodology*, Vol. 34, No. 1, 91–103.

BENTLEY, J. AND D. MCILROY (1993): "Engineering a Sort Function," *Software - Practice and Experience*, 23, 1249–1265.

BILLOR, N., A. S. HADI, AND P. F. VELLEMANN (2000): "BACON: Blocked Adaptive Computationally-efficient Outlier Nominators," *Computational Statistics and Data Analysis*, 34, 279–298.

BLACKFORD, L. S., A. PETITET, R. POZO, K. REMINGTON, R. C. WHALEY, J. DEMMEL, J. DONGARRA, I. DUFF, S. HAMMARLING, G. HENRY, M. HEROUX, L. KAUFMAN, AND A. LUMSDAINE (2002): "An updated set of basic linear algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, 28, 135–151.

GOLUB, G. H. AND C. F. VAN LOAN (1996): *Matrix Computations*, London: The Johns Hopkins University Press, 3rd ed.

GURWITZ, C. (1990): "Weighted median algorithms for L_1 approximation," *BIT Numerical Mathematics*, 30, 301–310.

HULLIGER, B. AND M. STERCHI (2020): *modi: Multivariate Outlier Detection and Imputation for Incomplete Survey Data*, R package version 0.1-0.

HYNDMAN, R. J. AND Y. FAN (1996): "Sample Quantiles in Statistical Packages," *The American Statistician*, 50, 361–365.

LI, J. AND R. VALLIANT (2009): "Survey weighted hat matrix and leverages," *Survey Methodology*, 35, 15–24.

MAECHLER, M., W. A. STAHEL, R. TURNER, U. OETLIKER, AND T. SCHOCH (2021): *robustX: 'eXtra' / 'eXperimental' Functionality for Robust Statistics*, R package version 1.2-5.

R DEVELOPMENT CORE TEAM (2020): *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0.

SEDGEWICK, R. (1997): *Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, and Searching*, USA: Addison-Wesley Longman Publishing Co., Inc., 3rd ed.

STEWART, G. W. (1998): *Matrix Algorithms: Volume 1, Basic Decompositions*, vol. 1, Philadelphia: SIAM Society for Industrial and Applied Mathematics.