Probabilistic algorithms for computing the LTS estimate.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Probabilistic algorithms for computing the LTS estimate

Martin Jenč

Department of Applied Mathematics
Supervisor: Ing. Karel Klouda, Ph.D.

May 14, 2019

Acknowledgements

THANKS to everybody

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 14, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Jenč, Martin. *Probabilistic algorithms for computing the LTS estimate*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Metoda nejmenších usekaných čverců je robustní verzí známé metdoy nejmenších čverců, jednou ze základních metod regresní analýzy, používané k odhadování koeficientů lineárního regresního modelu. Výpočet odhadadu pomocí metody nejmenších usekaných čtverců je znám jako NP-težký a proto jsou v praxi nejčatěji používány pouze subotimalní pravděpodobnostní algoritmy. Mimo popisu těchto algoritmů navhrneme několik způsobů jak je zkobinovat za účelem dosažení lepších výslekdů.

Klíčová slova nejmenší usekané čtverce, LTS odhad, linearní regrese, robustní statistika, nejmenší čverce, chybná pozorovaní

Abstract

The least trimmed squares method is a robust version of the method of least squares which is an essential tool of regression analysis used to find an estimate of coefficients in the linear regression model. Computing the least trimmed squared estimate is known to be NP-hard, and hence only suboptimal probabilistic algorithms are usually used in practice. Besides describing those algorithms, we propose a few ways of combining those algorithms to obtain better performance.

Keywords least trimmed squares, LTS estimate , linear regression, robust statistics, ordinary least squares, outliers

Contents

List of Figures

List of Tables

Introduction

Least trimmed squares (LTS) is one of many modifications very well known method ordinary least squares (OLS). Both of these methods are tools of regression analysis, which is a group of processes used to estimate the dependence of variables. Specifically in the case when we try to estimate dependence one variable on one or multiple others. The regression analysis uses the regression models, and in the case of OLS and LTS methods, that model is the linear regression model.

To OLS estimate produces reliable results, many strong assumptions about the data have to be fulfilled. It is assumed that data is generated specific way as well as that the data does not contain measurement errors called outliers. Those assumptions are in practice hardly fulfilled, because outliers in the data are very common. The OLS estimate is in such cases unreliable.

Problems of classical statistic methods try to solve robust statistic. Its methods are usually emulations of classical statistic methods but try to provide good performance even if data contains a large number of outliers. That means such methods does not rely so much on assumptions which are difficult to achieve in practice. Because the OLS method is one of the essential tools of linear regression analysis, multiple its alternatives have been designed to fulfill the assumptions of a robust estimator.

One of the methods is LTS. Its idea is simple, but unlike the OLS method, the exact formula for calculation is not known, so that only to date known exact algorithms for computing the LTS estimate are combinatorial. For this reason beside exact algorithms, several probabilistic algorithms have been proposed.

This work is divided into three chapters. In the first chapter, we introduce theory required to understand linear regression model, OLS and LTS. We mention properties of both methods and also describe filed of its usage.

In the second chapter, we cover algorithms for calculating the OLS estimate. It is necessary because most of the algorithm used to compute the LTS estimate relies on those algorithms. Next, we describe all currently used

algorithms for calculating LTS estimate. They consist of several probabilistic and also few exact ones. In this chapter, we also show that those algorithms can be easily combined to obtain higher speed and performance.

In the last chapter, we describe our experimental results. At first, we cover data generator which is used for our experiments and which can provide data affected by various types of outliers. Next, we provide information about our implementation of all algorithms for chapter two. Finally, we present our results for specific data.

Chapter 1

Least trimmed squares

In this chapter, we introduce one of the most common regression analysis models which is known as the linear regression model. It aims to model the relationship between one variable which is called *dependent* and one or more variables which are called *explanatory*. The relationship is based on a model function with parameters which are not known in advance and are to be estimated from data. We also describe one of the most common methods for finding those parameters in this model, namely the ordinary least squares method. It is important to note that all vectors in this text are considered as column vectors. On the other hand, we denote a row vector as a transposed vector.

1.1 Linear regression model

Definition 1. The *linear regression model* is given by

$$y = \boldsymbol{x}^T \boldsymbol{w} + \varepsilon, \tag{1.1}$$

where $y \in \mathbb{R}$ is a random variable which is called *dependent variable* and $\boldsymbol{x}^T = (x_1, x_2, \ldots, x_p)$ is a vector of *explanatory variables*. Usually we call x_i a regressor. Finally $\varepsilon \in \mathbb{R}$ is a random variable called *noise* or *error*. The vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_p)$ is a vector of parameters called *regression coefficients*.

In regression analysis we aim to estimate the $\boldsymbol{w}$ using n measurements of y and $\boldsymbol{x}$. We can write this in matrix form

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{\varepsilon} \tag{1.2}$$

where

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \dots & x_{np} \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}, \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

This means that we can think of rows of matrix $\boldsymbol{X}$ as columns vectors $\boldsymbol{x_i}$ written into the row.

It is assumed that errors are independent and identically distributed so that $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$.

Note 2. It is usual refer to given $\boldsymbol{X}$ and $\boldsymbol{y}$ as to *data set* and to y_i with corresponding $\boldsymbol{x_i}$ as to ith *data sample* or *observation*.

1.1.1 Prediction with the linear regression model

The linear regression model contains the vector $\boldsymbol{w}$ of regression coefficients which are unknown and which we need to estimate in order to be able to use the model for predictions. Let us assume that we already have estimated regression coefficients as $\hat{\boldsymbol{w}}$. Then the predicted values of y are given by

$$\hat{y} = \hat{\boldsymbol{w}}^T \boldsymbol{x}. \tag{1.3}$$

The true value of y is given by

$$y = \boldsymbol{w}^{*T} \boldsymbol{x} + \varepsilon \tag{1.4}$$

where $\boldsymbol{w}^*$ represents actual regression coefficients which we aim to estimate.

Because we assume linear dependence between dependent variable y and explanatory variables $\boldsymbol{x}$ which we assume to be non-random, then what makes y random variable is a random variable ε. Because we assume that $\mathbb{E}(\varepsilon) = 0$ we can see that

$$\mathbb{E}(y) = \boldsymbol{x}^T \boldsymbol{w} + \mathbb{E}(\varepsilon) = \boldsymbol{x}^T \boldsymbol{w} \tag{1.5}$$

so $\hat{y}$ is a point estimation of the expected value of y.

Intercept

In real world situations it is not usual that $\mathbb{E}(\varepsilon) = 0$. Consider this trivial example.

Example 3. Let us consider that y represents price of the room and x represents the number of the windows in such a room. If this room does not have windows thus $x = 0$ and $\mathbb{E}(\varepsilon) = 0$ then $y = wx + \varepsilon$ equals zero. But it is very unlikely that room without windows is free.

Because of that, it is very common to include one constant regressor $x_1 = 1$. The corresponding coefficient w_1 of $\boldsymbol{w}$ is called an *intercept*. We refer this model as a *model with an intercept*. The intercept then corresponds to expected value of y when all regressors are zero and prevent the problem from Example 3. This means that intercept can be assumed as a shift so that it corresponds to $\mathbb{E}(\varepsilon) = \mu$. With regards to this fact we can still assume that in the model with intercept $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. In this work, we consider the model with the intercept. This means that we consider $\boldsymbol{x} = (x_1, x_2 \dots x_p)$ where constant $x_1 = 1$ represents intercept.

Note 4. Sometimes in the model with an intercept, the explanatory variable $\boldsymbol{x}$ is marked as $\boldsymbol{x} \in \mathbb{R}^{p+1}, \boldsymbol{x} = (x_0, x_1 \dots x_p)$, which means that actual observation $\boldsymbol{x} \in \mathbb{R}^p$ and the intercept $x_0 = 1$ is explicitly marked.

1.2 Ordinary least squares

We want to estimate $\boldsymbol{w}$ so that an error of the model on the whole data set is the least possible. This error is measured by a *loss function* $L : \mathbb{R}^2 \to \mathbb{R}$, which in case of the *ordinary least squares* (OLS) method is quadratic loss function $L(y, \hat{y}) := (y - \hat{y})^2$. So the idea is to find $\hat{\boldsymbol{w}}$ so that it minimizes the sum of squared *residuals*

$$r_i(\boldsymbol{w}) = y_i - \hat{y}_i = y_i - \boldsymbol{x_i}^T \boldsymbol{w}, \ i = 1, 2, \dots, n. \tag{1.6}$$

This is commonly know as residual sum of squares RSS

$$RSS(\boldsymbol{w}) = \sum_{i=1}^{n} r_i^2(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2. \tag{1.7}$$

Definition 5. The RSS as the function of $\boldsymbol{w}$ is an *objective function* for OLS

$$\mathrm{OF}^{(OLS, \boldsymbol{X}, \boldsymbol{y})}(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}^T \boldsymbol{x_i})^2 = (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w})^T (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w}). \tag{1.8}$$

The point of the minimum of this function

$$\hat{\boldsymbol{w}}^{(\boldsymbol{OLS}, \boldsymbol{n})} = \arg\min_{\boldsymbol{w} \in \mathbb{R}^p} \mathrm{OF}^{(OLS, \boldsymbol{X}, \boldsymbol{y})}(\boldsymbol{w}) \tag{1.9}$$

is a the ordinary least squares estimate of regression coefficients.

To find the minimum of this function, we first need to find the gradient by calculating all partial derivatives

$$\frac{\partial \mathrm{OF}^{(OLS, \boldsymbol{X}, \boldsymbol{y})}}{\partial w_j} = \sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T \boldsymbol{x_i})(-x_{ij}), \quad j \in \{1, 2, \dots, p\}. \tag{1.10}$$

By this we obtain the gradient

$$\nabla \mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})} = -\sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T\boldsymbol{x_i})\boldsymbol{x_i}. \tag{1.11}$$

Putting gradient equal to zero we get the so called *normal equations*

$$-\sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T\boldsymbol{x_i})\boldsymbol{x_i} = 0 \tag{1.12}$$

that can be rewritten in a matrix form as

$$\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = 0. \tag{1.13}$$

Let us now construct the hessian matrix using second-order partial derivatives:

$$\frac{\partial^2 \mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}}{\partial w_h \partial w_j} = \sum_{i=1}^{n} 2(-x_{ik})(-x_{ij}), \quad h \in \{1, 2, \ldots, p\}. \tag{1.14}$$

We get

$$\boldsymbol{H}_{\mathbf{OF}^{(\boldsymbol{OLS},\boldsymbol{X},\boldsymbol{y})}} = 2\boldsymbol{X}^T\boldsymbol{X}. \tag{1.15}$$

We can see that hessian $\boldsymbol{H}_{\mathbf{OF}^{(\boldsymbol{OLS},\boldsymbol{X},\boldsymbol{y})}}$ is always positive semi-definite because for all $\boldsymbol{s} \in \mathbb{R}^p$

$$\boldsymbol{s}^T(2\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{s} = 2(\boldsymbol{X}\boldsymbol{s})^T(\boldsymbol{X}\boldsymbol{s}) = 2\left\|\boldsymbol{X}\boldsymbol{s}\right\|^2 \tag{1.16}$$

It is easy to prove that twice differentiable function is convex if and only if the hessian of such function is positive semi-definite. Moreover, any local minimum of the convex function is also the global one. Hence the solution of (1.13) gives us the global minimum.

Assuming that $\boldsymbol{X}^T\boldsymbol{X}$ is a regular matrix, its inverse exists, and the solution can be explicitly written as

$$\hat{\boldsymbol{w}}^{(OLS,n)} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \tag{1.17}$$

where (OLS, n) denotes that $\hat{\boldsymbol{w}}^{(OLS,n)}$ is the estimate of true regression coefficients given by the OLS method for n observations.

Moreover, we can see that if $\boldsymbol{X}^T\boldsymbol{X}$ is a regular matrix, then the hessian $\boldsymbol{H}_{\mathbf{OF}^{(\boldsymbol{OLS},\boldsymbol{X},\boldsymbol{y})}}$ is positive definite and $\mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$ is strictly convex and $\hat{\boldsymbol{w}}^{(OLS,n)}$ is the unique strict global minimum.

1.2.1 Properties of the OLS estimate

Gauss-Markov theorem [1] tells us that if particular assumptions about the regression model are fulfilled then the OLS estimate is unbiased and efficient. Gauss-Markov theorem states that OLS is the best linear unbiased estimator (BLUE). Being an efficient estimate means that any other linear unbiased estimate has the same or higher variance. The most important conditions are:

- Expected value of errors is zero.
- Errors are independently distributed and uncorrelated thus $cov(\varepsilon_i, \varepsilon_j),\ i, j = 1, 2 \ldots, n,\quad i \neq j$
- All errors have same finite variance. This is known as *homoscedasticity*.

There are also other theorems which describe properties of OLS under specific conditions, but they are out if the scope of this work.

1.3 Robust statistics

Standard statistics methods rely on multiple assumptions and fail if those assumptions are not met. The goal of the robust statistics is to produce acceptable results even when the data are from some unconventional distributions or if data contains outliers or errors which are not normally distributed.

Such assumptions about the OLS method are described in Section 1.2.1. Before we explain what happens if those conditions are not met or met only partially let us describe one of the most common reasons why assumptions are false.

1.3.1 Outliers

We stated many assumptions that are required for the OLS method to produce a acceptable estimate of $\hat{\boldsymbol{w}}$. Unfortunately, in real conditions, these assumptions are often false so that the ordinary least squares do not guarantee to return reasonable results. One of the most common reasons for the assumptions being not met are observations called outliers.

Outliers are for instance erroneous measurements such as transmission errors or noise. Another common reason for outliers is that nowadays the data are automatically processed by computers. Sometimes we are also given data which are heterogeneous in the sense that they contain data from multiple regression models. In some sense outliers are inevitable. One would say that we should be able to eliminate them by precise examination, repair or removal of such data. That is possible in some cases, but often the data we are dealing with are too big and highly dimensional to check.

The robust methods are sometimes not only useful to create models that are not being unduly affected by the presence of outliers but also capable of identifying data which seems to be outliers.

We use terminology from [2] to describe certain types of outliers. Let us have observation $(y_i, \boldsymbol{x_i})$. If the observation is not outlying in any direction we call it *regular observation* . If it is outlying in direction of the explanatory variable $\boldsymbol{x_i}$ we call it *leverage point*. We have two types of leverage points. If $\boldsymbol{x_i}$ is outlying but $(y_i, \boldsymbol{x_i})$ follows the liner pattern we call it a *good leverage point*. If it does not follow such a pattern we call it *bad leverage point*. Finally if $(y_i, \boldsymbol{x_i})$ is outlying only in direction of y_i, we call it a *vertical outlier*.

1.3.2 Measuring robustness

There are a couple of tools to measure the robustness of an estimate. One of the most popular one is called *breakdown point*. Others are *empirical influence function* and *influence function and sensitivity curve*. Here we describe only breakdown point right now. More on robustness measures can be found in [3].

Definition 6. Let T be a statistics, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ be an n-element random sample and $T_n(\boldsymbol{x})$ value of this statistics. The breakdown point of T at sample $\boldsymbol{x}$ is defined using sample $\overline{\boldsymbol{x}}^{(k)}$, that arose by replacing k points from the original sample $\boldsymbol{x}$ with random values x_i. Then the *breakdown point* is

$$\text{bdpoint}(T, \boldsymbol{x_n}) = \frac{1}{n} \min S_{T,\boldsymbol{x_n}}, \tag{1.18}$$

where

$$S_{T,\boldsymbol{x_n},D} = \left\{ k \in \{1, 2, \ldots, n\} : \sup_{\overline{\boldsymbol{x}}^{(k)}} \left\| T_n(\boldsymbol{x}) - T_n(\overline{\boldsymbol{x}}^{(k)}) \right\| = \infty \right\}. \tag{1.19}$$

This definition is says that the breakdown point is the function of the minimal number of observations needed to be changed so that the estimator gives arbitrarily biased results.

Intuitively, a reasonable breakdown point should not be higher than 0.5 [4] ; if more than 50% of the data is exchanged, the model of exchanged data should override the model of the original data.

In the case of the OLS estimator, one outlier is enough to increase the value of $\hat{\boldsymbol{w}}^{(OLS,n)}$ to any desired value [5] thus

$$\text{bdpoint}(\hat{\boldsymbol{w}}^{(OLS,n)}, \boldsymbol{x_n}) = \frac{1}{n}. \tag{1.20}$$

Figure 1.1 gives us an idea of how one outlier may change the hyperplane given by the OLS estimator of regression coefficients.

For an increasing number of the data samples n the breakdown point of $\hat{\boldsymbol{w}}^{(OLS,n)}$ tends to zero. We can see that ordinary least squares estimator is

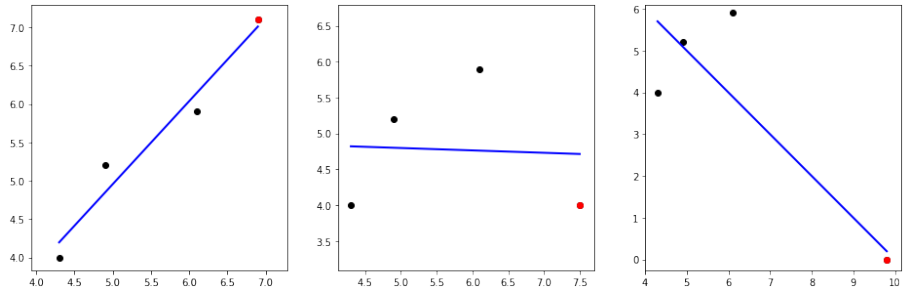


Figure 1.1: Change of the regression hyperplane given by coefficients estimated with OLS method when one of the four observations (highlighted with red color) starts to deviate from the linear pattern.

not resistant to outliers at all. Due to this fact, multiple robust estimators alternatives to the OLS have been proposed.

1.4 Least trimmed squares

The least trimmed squares (LTS) estimator is a robust version of the OLS estimator. In this section, we give a definition and show that its breakdown point is variable and can go up to the maximum possible value of breakdown point, thus 0.5.

Definition 7. Let us have $\boldsymbol{X} \in \mathbb{R}^{n,p}$, $\boldsymbol{y} \in \mathbb{R}^{n,1}$, $\boldsymbol{w} \in \mathbb{R}^p$ and h, $n/2 \leq h \leq n$. The objective function of LTS for data $\boldsymbol{X}$ and $\boldsymbol{y}$ is

$$\mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}) = \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) \tag{1.21}$$

where $r_{i:n}^2(\boldsymbol{w})$ denotes the ith smallest squared residuum at $\boldsymbol{w}$, i.e.

$$r_{1:n}^2(\boldsymbol{w}) \leq r_{2:n}^2(\boldsymbol{w}) \leq \ldots \leq r_{n:n}^2(\boldsymbol{w}). \tag{1.22}$$

Even though that objective function of LTS seems similar to the OLS objective function, finding the minimum is far more complex because the order of the least squared residuals depends on $\boldsymbol{w}$. Moreover, $r_{i:n}^2(\boldsymbol{w})$ residuum is not uniquely determined if more squared residuals have same value. This makes finding the LTS estimate non-convex optimization problem and, in fact, finding the global minimum is NP-hard [6].

1.4.1 Discrete objective function

The LTS objective function from Definition 7 is not differentiable and not convex, so we are unable to use the same approach as with the OLS objective

function. Let us transform this objective function to a discrete version which is easier to use by algorithms to minimize it.

Let us assume for now that we know the vector $\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})}$ of estimated regression coefficients minimizing the LTS objective function. Let π be the permutation of $\hat{n} = \{1, 2, \ldots, n\}$ such that

$$r_{i:n}(\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})}) = r_{\pi(j)}(\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})}), \quad j \in \hat{n}. \tag{1.23}$$

Put

$$Q^{(n,h)} = \left\{ \boldsymbol{m} \in \mathbb{R}^n \mid m_i \in \{0,1\}, i \in \hat{n}, \sum_{i=1}^{n} m_i = h \right\}, \tag{1.24}$$

which is simply the set of all vectors $\boldsymbol{m} \in \mathbb{R}^n$ which contain h ones and $n-h$ zeros. Let $\boldsymbol{m}^{(LTS)} \in Q^{(n,h)}$ such that $m_j^{(LTS)} = 1$ when $\pi(j) \leq h$ and $m_j^{(LTS)} = 0$ otherwise. Then

$$\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^p} \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}\in\mathbb{R}^p} \sum_{i=1}^{n} m_i^{(LTS)} r_i^2(\boldsymbol{w}). \tag{1.25}$$

This means that if we know the vector $\boldsymbol{m}_{LTS}$ than we can compute the LTS estimate as the OLS estimate with $\boldsymbol{X}$ and $\boldsymbol{Y}$ multiplied by the diagonal matrix $\boldsymbol{M}_{LTS} = diag(\boldsymbol{m}^{(LTS)})$:

$$\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})} = (\boldsymbol{X}^T \boldsymbol{M}_{LTS} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{M}_{LTS} \boldsymbol{y}. \tag{1.26}$$

In other words, finding the minimum of the LTS objective function can be done by finding the OLS estimates (1.26) for all vectors $\boldsymbol{m} \in Q^{(n,h)}$. Thus if we denote $\boldsymbol{X}_{\boldsymbol{M}} = \boldsymbol{M}\boldsymbol{X}$ and $\boldsymbol{y}_{\boldsymbol{M}} = \boldsymbol{M}\boldsymbol{y}$ which corresponds to the h-element subsets of $\boldsymbol{X}$ and $\boldsymbol{Y}$, then as described in [7],

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} \mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}) = \min_{\boldsymbol{w}\in\mathbb{R}^p} \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) \tag{1.27}$$

$$= \min_{\boldsymbol{w}\in\mathbb{R}^p, \boldsymbol{m}\in Q^{(n,h)}} \sum_{i=1}^{n} m_i r_i^2(\boldsymbol{w}) \tag{1.28}$$

$$= \min_{\boldsymbol{m}\in Q^{(n,h)}} \Big(\min_{\boldsymbol{w}\in\mathbb{R}^p} \mathrm{OF}^{(OLS,\boldsymbol{M}\boldsymbol{X},\boldsymbol{M}\boldsymbol{y})}(\boldsymbol{w}) \Big) \tag{1.29}$$

$$= \min_{\boldsymbol{m}\in Q^{(n,h)}} \Big(\min_{\boldsymbol{w}\in\mathbb{R}^p} \|\boldsymbol{M}\boldsymbol{y} - \boldsymbol{M}\boldsymbol{X}\boldsymbol{w}\|^2 \Big). \tag{1.30}$$

Substituting $\boldsymbol{w}$ with the OLS estimate as in (1.25) we get

$$\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})} = \min_{\boldsymbol{m}\in Q^{(n,h)}} \Big(\Big\| \boldsymbol{M}\boldsymbol{y} - \boldsymbol{M}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{y} \Big\|^2 \Big)$$

We get the discrete objective function with domain $\boldsymbol{m} \in Q^{(n,h)}$

$$\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) = \left\| \boldsymbol{M}\boldsymbol{y} - \boldsymbol{M}\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{y} \right\|^2. \tag{1.31}$$

Minimizing this OF could by done straightforwardly by iterating over the $Q^{(n,h)}$ set. Unfortunately, this set has cardinality equal to $\binom{n}{h}$, which is huge, so this approach is infeasible for bigger data sets. Multiple algorithms were proposed to overcome this problem. Majority of them are probabilistic algorithms, but besides those, some exact algorithms were proposed.

Finally, let us point out some fact about the number h of non-trimmed residuals and how it makes least trimmed squares robust. The LTS reaches maximum breakdown point 0.5 at $h = [(n/2] + [(p + 1)/2]$ [5]. This means that up to 50% of the data can be outliers. In practice, the portion of outliers is usually lower; if an upper bound on the percentage of outliers is known, h should be set to match this percentage.

Chapter 2

Algorithms

In the previous chapter, we have covered the theoretical background required to implement algorithms that are presented in this chapter. We have introduced the discrete version of the LTS objective function whose minimum is equivalent to the continuous one. Finding the minimum of this function requires to find the particular h-element subset and then calculate the estimate of the corresponding regression coefficients $\boldsymbol{w}$. To achieve this, we need to examine all h-element subsets. The exhaustive approach fails due to the exponential size of $Q^{(n,h)}$.

2.0.1 First attempts

First known algorithms were based on iterative removal data samples whose residuum had the highest value based on the OLS estimate on the whole dataset. Such attempts are known to be flawed [8] because the initial OLS fit can be already profoundly affected by outliers and the algorithm may remove data samples which represent the actual model.

Other algorithms were based purely on a random approach. One such algorithm is the Random solution algorithm [9] which randomly selects k h-element subsets and subsequently compute the OLS estimate on each of them and chooses the estimate with a minimum value of the objective function. Such approach is straightforward, but the probability of selecting at least one h-element out of k subsets which does not contain outliers thus has a chance of producing good result tends to zero for an increasing number of data samples n as we describe in detail in Section 2.2.2.

Another very similar algorithm called Resampling algorithm introduced in [10]. It selects vectors from $Q^{(n,p+1)}$ instead of $Q^{(n,h)}$. This minor tweak has a higher chance to succeed. Mainly because the probability of selecting k h-element subsets of size $p+1$ gives the nonzero probability of selecting at least one h-element subset not containing outliers (see Section 2.2.2 for more details). Besides that, the number of vectors in this set is significantly lower

than in $Q^{(n,h)}$ (at least if h is conservatively chosen so that $h = [(n/2] + [(p + 1)/2])$.

2.0.2 Strong and weak necessary conditions

Generating all possible h-element subsets is computationally exhaustive and relying on randomly selecting "good" h-element subsets does not lead to reliable results. So what are our options? In [11] two criteria called a *weak necessary condition* and a *strong necessary condition* are introduced. They state necessary properties which some h-element subset must satisfy to be a subset which leads to the global optimum of the LTS objective function. Let us introduce those two necessary conditions. For that, it is convenient not only to label a h-element subset of *non-trimmed* observations but also the complementary subset of trimmed observations. We refer to this complementary subset as to *trimmed* subset.

Definition 8. A h-element subset corresponding to $\boldsymbol{m} \in Q^{(n,h)}$ satisfies the *strong necessary condition* if for any vector $\boldsymbol{m_{swap}}$ which differs from $\boldsymbol{m}$ only by swapping one 1 with one 0, we get $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) \leq \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_{swap}})$. In words, the value of the discrete LTS objective function cannot be reduced by swapping one non-trimmed observation with one trimmed observation.

Based on this fact an algorithm can be created. We'll discuss it in detail in Section 2.3.

Definition 9. An h-element subset corresponding to $\boldsymbol{m} \in Q^{(n,h)}$ satisfies the *weak necessary condition* if $r_i^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})})$ for all trimmed observation is greater or equal to the greatest non-trimmed squared residuum $r_j^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})})$.

Again, based on this criteria an algorithm can be created. Interesting consequence which we use later gives us the following lemma.

Lemma 10. The strong necessary condition is not satisfied unless the weak necessary condition is satisfied. Thus, if a strong condition is satisfied then weak is too.

Proof. Let us assume that we have some $\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})}$ with $\boldsymbol{m} \in Q^{(n,h)}$ for which the strong necessary condition is satisfied, but the weak necessary condition is not. That means there exists $\boldsymbol{x_i}$ with y_i from the non-trimmed subset and $\boldsymbol{x_j}$ and y_j from the trimmed subset such that

$$r_j^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})}) < r_i^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})}).$$

Thus

$$\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) > \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) + r_j^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})}) - r_i^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,MX,My})}) \quad (2.1)$$

Now we need to show that $\boldsymbol{m_{swap}}$ vector that is created by swapping jth observation from the trimmed subset with ith observation from the non-trimmed subset leads to

$$\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) + r_j^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS},\boldsymbol{MX},\boldsymbol{My})}) - r_i^2(\hat{\boldsymbol{w}}^{(\boldsymbol{OLS},\boldsymbol{MX},\boldsymbol{My})}) \geq \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_{swap}}). \tag{2.2}$$

That is true because the value of $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_{swap}})$ is the minimum on the given subset of observations. That is, of course, a contradiction with our assumption which says that the strong necessary condition is satisfied. □

2.1 Computing OLS

In this section, we describe a few of many methods that can be used to obtain $\hat{\boldsymbol{w}}^{(OLS,n)}$. Those methods are parts of the algorithms used to calculate the LTS estimate.

In the following algorithms, we describe its time complexity. Because the matrix multiplication is the fundamental part of all the following algorithms, it is, therefore, appropriate to mention a few facts about the time complexity of the matrix multiplication.

There are multiple algorithms for the multiplication of $n \times n$ matrices, for example:

- Naive algorithm; its time complexity is $\mathcal{O}(n^3)$.
- Strassen algorithm; its time complexity is $\mathcal{O}(n^{2.8074})$ [12].
- Coppersmith-Winograd; its time complexity is $\mathcal{O}(n^{2.375477})$ [13]

In practice, however, the naive algorithm is usually used. Even though some of those algorithms have been efficiently implemented and are known to be numerically stable (primarily variations of Strassen algorithm), their error bound is weaker than in the case of the naive algorithm [14]. For this reason, they are not used even when they might improve performance.

Moreover, currently widely used linear algebra software libraries such as LAPACK uses Basic Linear Algebra Subprograms (BLAS) low-level routines which implement naive matrix-matrix multiplication [15].

For that reason we assume in this work that the matrix multiplication is $\mathcal{O}(n^3)$. For not square matrices $\boldsymbol{A} \in \mathbb{R}^{m\times n}, \boldsymbol{B} \in \mathbb{R}^{n\times p}$ it is then $\mathcal{O}(mnp)$.

2.1.1 Computation using a matrix inversion

Calculating the OLS estimate using the matrix inversion can be done as follows:

1. Compute $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{y}$.

2. Find the inversion of $\boldsymbol{A}$.

3. Multiply $\boldsymbol{A}^{-1}\boldsymbol{B}$.

Observation 11. Time complexity of computing the OLS estimate on $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$ using the matrix inversion is $O(p^2 n)$.

Proof. First, we compute the $\boldsymbol{A} = \boldsymbol{X^T X}$ and $\boldsymbol{B} = \boldsymbol{X^T y}$. That gives us $\mathcal{O}(p^2 n + pn)$. Next, we compute inversion $\boldsymbol{C} = \boldsymbol{A}^{-1}$ which gives us $\mathcal{O}(p^3)$. Finally, we compute $\hat{\boldsymbol{w}}^{(OLS,n)} = \boldsymbol{CB}$ which is $\mathcal{O}(p^2)$. Altogether, we get $\mathcal{O}(p^2 n + pn + p^3 + p^2)$; $p^2 n$ and p^3 asymptotically dominates over the rest so

$$\mathcal{O}(p^2 n + pn + p^3 + p^2) \sim \mathcal{O}(p^2 n + p^3). \tag{2.3}$$

Moreover, if we assume that $n \geq p$, we get $\mathcal{O}(p^2 n + p^3) \sim \mathcal{O}(p^2 n)$. □

2.1.2 Computation using the Cholesky decomposition

Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix. Then it is possible to find lower triangular matrix L so that

$$\boldsymbol{A} = \boldsymbol{LL^T} \tag{2.4}$$

We call this decomposition *Cholesky factorization* (sometimes also *Cholesky decomposition*)

If we look at our problem of finding a solution of

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y}, \tag{2.5}$$

we can easily rewrite it as

$$\boldsymbol{Z}\boldsymbol{w} = \boldsymbol{b} \tag{2.6}$$

where $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{X}$ is a symmetric positive definite matrix and $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y}$. Because $\boldsymbol{Z}$ can be factorized as $\boldsymbol{LL^T}$ the solution can be found easily by substitution

$$\boldsymbol{Ld} = \boldsymbol{b} \tag{2.7}$$

$$\boldsymbol{L}^T\boldsymbol{w} = \boldsymbol{d}, \tag{2.8}$$

where $\boldsymbol{d}$ and $\boldsymbol{w}$ can be obtained by forward and backward substitution. Solution of $\boldsymbol{w}$ then represents $\hat{\boldsymbol{w}}^{(OLS,n)}$ estimate.

So the algorithm for solving OLS using Cholesky factorization goes as follows:

1. Compute $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{y}$.

2. Compute Cholesky factorization $\boldsymbol{Z} = \boldsymbol{LL}^T$ where $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{X}$.

3. Solve lower triangular system $\boldsymbol{Ld} = \boldsymbol{b}$ where $\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y}$ for $\boldsymbol{d}$ using forward substitution.

4. Solve upper triangular system $\boldsymbol{L}^T\boldsymbol{w} = \boldsymbol{d}$ for $\boldsymbol{w}$.

Observation 12. The time complexity of solving OLS using Cholesky factorization is $\mathcal{O}(p^2 n)$

Proof. First step requires two matrix multiplication $\boldsymbol{X}^T\boldsymbol{X}$ which is $\mathcal{O}(np^2)$ and $\boldsymbol{X}^T\boldsymbol{y}$ which is $\mathcal{O}(np)$. Second step represents computing Cholesky factorization. This can be done in $\mathcal{O}(\frac{1}{3}p^3)$ [16]. In the third and fourth step, we are solving triangular linear systems; both of them require $\mathcal{O}(\frac{1}{2}p^2)$.

Putting all the steps together we get

$$\mathcal{O}(np^2 + np + \frac{1}{3}p^3 + p^2) \tag{2.9}$$

and because we assume $n \geq p$ then the multiplication $\boldsymbol{X}^T\boldsymbol{X}$ asymptotically dominates over the rest so we get $\mathcal{O}(p^2 n)$. □

Computing the OLS estimate using the Cholesky factorization is more numerically stable than using matrix inversion and the time complexity is asymptotically similar.

2.1.3 Computation using the QR decomposition

Let us now look on a similar method of computing the OLS estimate which does not require multiplying $\boldsymbol{X}^T\boldsymbol{X}$.

Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be square matrix. Then exists matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ so that

$$\boldsymbol{A} = \boldsymbol{QR}, \tag{2.10}$$

where $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. This decomposition is known as *QR decomposition*.

If matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is not square, then decomposition can be found as

$$\boldsymbol{A} = \boldsymbol{QR} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{R_1}\\ \boldsymbol{0}\end{bmatrix} = \begin{bmatrix}\boldsymbol{Q_1} & \boldsymbol{Q_2}\end{bmatrix}\begin{bmatrix}\boldsymbol{R_1}\\ \boldsymbol{0}\end{bmatrix} = \boldsymbol{Q_1}\boldsymbol{R_1} \tag{2.11}$$

Where $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, $\boldsymbol{R_1} \in \mathbb{R}^{n\times n}$ is upper an triangular matrix, $\boldsymbol{0} \in \mathbb{R}^{(m-n)\times n}$ is a zero matrix and $\boldsymbol{Q_1} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{Q_2} \in \mathbb{R}^{(m-n)\times n}$ are matrices with orthogonal columns.

Given a QR decomposition of $\boldsymbol{X}$

$$\boldsymbol{X} = \boldsymbol{QR} \tag{2.12}$$

we get

$$\boldsymbol{X}^T\boldsymbol{X} = (\boldsymbol{QR})^T\boldsymbol{QR} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{QR} \tag{2.13}$$

and because $\boldsymbol{Q}$ is orthogonal, $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$ and so

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{R}^T\boldsymbol{R}. \tag{2.14}$$

Because $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ is not a square matrix then $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{0} \end{bmatrix}$ and

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{R_1}^T\boldsymbol{R_1}, \tag{2.15}$$

where $\boldsymbol{R_1^T}$ is a lower triangular matrix.

Solution to

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \tag{2.16}$$

is then given by

$$\boldsymbol{R_1}^T\boldsymbol{R_1}\boldsymbol{w} = \boldsymbol{R_1}^T\boldsymbol{Q_1}^T\boldsymbol{y}. \tag{2.17}$$

If we assume $\boldsymbol{X}$ have full column rank, $\boldsymbol{R_1}$ must be invertible and this equation can be simplified to

$$\boldsymbol{R_1}\boldsymbol{w} = \boldsymbol{b_1} \tag{2.18}$$

where $\boldsymbol{b_1} = \boldsymbol{Q_1}^T\boldsymbol{y}$. Because $\boldsymbol{R_1}$ is upper an triangular matrix, finding the is trivial using the backward substitution. Resulting $\boldsymbol{w}$ is then the OLS estimate $\hat{\boldsymbol{w}}^{(OLS,n)}$.

Note 13. We can see that Cholesky factorization $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$ is closely connected to the QR decomposition $\boldsymbol{X} = \boldsymbol{Q_1}\boldsymbol{R_1}$. Indeed, putting

$$\boldsymbol{L} = \boldsymbol{R_1^T} \tag{2.19}$$

we can get Cholesky decomposition directly from the QR decomposition.

The algorithm for solving the OLS using QR decomposition can go as follows:

1. Calculate a QR decomposition $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}^T = \boldsymbol{Q_1}\boldsymbol{R_1}^T$.
2. Calculate $\boldsymbol{b_1} = \boldsymbol{Q_1}^T\boldsymbol{y}$.
3. Solve upper triangular system $\boldsymbol{R_1}\boldsymbol{w} = \boldsymbol{b_1}$.

The QR factorization can be calculated in multiple ways. The most basic method is applying the Gram-Schmidt process to columns of the matrix $\boldsymbol{X}$. This approach is not numerically stable so in practice it is not used as much as two following methods: *Householder transformations* and *Givens rotations*. The time complexity of both algorithms is similar. QR decomposition of a matrix $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ is [17]

$$\mathcal{O}(2p^2n - \frac{2}{3}p^3) \sim O(p^2n). \tag{2.20}$$

On the other hand, using Givens rotation for QR decomposition of the same matrix is

$$\mathcal{O}(3np^2 - p^3) \sim O(p^2 n). \tag{2.21}$$

Although Householder transformations are about 50% faster, both have asymptotically equal time complexity. Moreover Givens rotations are more numerically stable. We speak about Givens rotations in detail in Section 2.1.3 where we also make proof of the (2.21). Moreover, Givens rotations and are suitable for sparse matrices. We use this property of Givens rotations in Section 2.4.4.

For now, let us look at the time complexity of solving the OLS using Householder transformations.

Observation 14. The time complexity of finding the OLS estimate using the QR decomposition by Householder transformations is $\mathcal{O}(2p^2 n - \frac{2}{3}p^3) \sim \mathcal{O}(p^2 n)$.

Proof. In the first step, we calculate a QR decomposition. Householder transformations can be done in $\mathcal{O}(2np^2 - \frac{2}{3}p^3)$. The second step consists of the matrix multiplication $\boldsymbol{Q_1}^T \boldsymbol{y}$ which is $\mathcal{O}(np)$. In the last step we solve upper triangular linear system which is $\mathcal{O}(\frac{1}{2}p^2)$. Putting all steps together we get

$$\mathcal{O}(2np^2 - \frac{2}{3}p^3 + np + \frac{1}{2}p^2) \tag{2.22}$$

We can see that $p^2 n$ asymptotically dominates (this is the same as in the case of Givens rotations). □

The QR decomposition is considered as a standard way of computing OLS estimate because of its high numerical stability. Let us mention that a little slower, but a more stable method of computing the OLS estimate exists, and that is the one using the singular value decomposition (SVD). On the other hand, the QR decomposition is sufficiently stable for most cases so describing SVD decomposition is out of the scope of this work.

Givens Rotation

In this section, we describe Givens rotations in detail. Moreover, we also show how to update QR decomposition when adding or deleting a from the matrix $\boldsymbol{X}$. We use this in the algorithm described in Section 2.4.4.

As described in [18], we compute the QR factorization of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ so that we apply an orthogonal transformation using a matrix $\boldsymbol{Q}^T$ as

$$\boldsymbol{Q}^T \boldsymbol{A} = \boldsymbol{R} \tag{2.23}$$

where $\boldsymbol{Q}$ is a product of orthogonal matrices. These matrices have the following matrix as sub-matrix:

$$\boldsymbol{Q}_{\varphi} = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \tag{2.24}$$

This matrix is indeed orthogonal since

$$\boldsymbol{Q}_{\varphi}\boldsymbol{Q}_{\varphi}^{T} = \boldsymbol{Q}_{\varphi}^{T}\boldsymbol{Q}_{\varphi} = \boldsymbol{I}. \tag{2.25}$$

Moreover, if we multiply this orthogonal matrix with some vector $\boldsymbol{x} \in \mathbb{R}^2$, as a result $\boldsymbol{Q}\boldsymbol{x}$ we get a vector of the same length as $\boldsymbol{x}$ which is rotated clockwise by the angle φ. When we say that vector has the same length we mean that L^2 norm of such vector stays the same. This is an important property of all orthogonal matrices (operations that preserve L^2 norm are known as *unitary transformations*). Let us verify this claim. Let us have an orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ and any vector $\boldsymbol{x} \in \mathbb{R}^m$ then

$$\|\boldsymbol{Q}\boldsymbol{x}\|^2 = (\boldsymbol{Q}\boldsymbol{x})^T(\boldsymbol{Q}\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} = \|\boldsymbol{x}\|^2. \tag{2.26}$$

So as we said, the idea is to create a series of orthogonal matrices which gradually rotates two-element column sub-vectors of $\boldsymbol{A}$, so that we obtain zeros under diagonal. One such method - *Givens rotation* uses *Givens matrices* that erase one element under diagonal at a time.

The example of (2.24) is not random. Let us look at the orthogonal matrix $\boldsymbol{Q}_{\varphi}$ one more time and let us multiply this matrix with some vector.

$$\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ z \end{bmatrix}. \tag{2.27}$$

To obtain the zeroing effect, we need to rotate this vector so that it is parallel to $(1, 0)^T$. So we only need to find φ so that $\cos(\varphi)a + \sin(\varphi)b = r$. As we know, rotation with $\boldsymbol{Q}_{\varphi}$ preserves L^2 norm. Hence if we want $z = 0$, then r must be equal to L^2 norm of the vector $(a, b)^T$ i.e. $r = \sqrt{a^2 + b^2}$. This leads to the solution

$$\cos(\varphi) = \frac{a}{\sqrt{a^2 + b^2}} \tag{2.28}$$

and

$$\sin(\varphi) = \frac{b}{\sqrt{a^2 + b^2}}. \tag{2.29}$$

In practice, the algorithm of computing $\cos(\varphi)$ and $\sin(\varphi)$ is slightly different because we want to prevent arithmetic overflow. This algorithm is described by the following pseudocode:

Algorithm 1: Rotate

Input: a, b
Output: cos, sin
$sin \leftarrow \emptyset$;
$cos \leftarrow \emptyset$;
if $b == 0$ **then**
  $sin \leftarrow 0$;
  $cos \leftarrow 1$;
else if $abs(b) \geq abs(a)$ **then**
  $cotg \leftarrow \frac{a}{b}$;
  $sin \leftarrow \frac{1}{\sqrt{1+(cotg)^2}}$;
  $cos \leftarrow sin\ cotg$;
else
  $tan \leftarrow \frac{b}{b}$;
  $cos \leftarrow \frac{1}{\sqrt{1+(tan)^2}}$;
  $sin \leftarrow cos\ tan$;
end
return $cos\ ,\ sin$;

We can scale the same idea to higher dimensions. Let us denote matrix $\boldsymbol{Q}_{\varphi}(i,j) \in \mathbb{R}^{m \times m}$ defined as

$$\boldsymbol{Q}_{\varphi}(i,j) = \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & & i & & j & \end{array} \\ \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix} \end{array}$$

where $c = \cos(\varphi)$ and $s = \sin(\varphi)$ for some φ. That means this matrix is orthogonal. Now if we have some column vector $\boldsymbol{x} \in \mathbb{R}^{m \times 1}$ and we calculate c and s by (2.28) and (2.29) for $a := x_i$ and $b := x_j$. Finally if we multiply $\boldsymbol{Q}_{\varphi}(i,j)\boldsymbol{x} = \boldsymbol{p}$ we can see that $\boldsymbol{p}$ is the same vector as $\boldsymbol{x}$ except for p_i and p_j so that:

$$\boldsymbol{Q}_{\varphi}(i,j)\boldsymbol{x} = \boldsymbol{Q}_{\varphi}(i,j)\begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \\ x_m \end{bmatrix} = \boldsymbol{p} = \begin{bmatrix} x_1 \\ \vdots \\ r \\ \vdots \\ 0 \\ \vdots \\ x_m \end{bmatrix}$$

where $r = \sqrt{x_i^2 + x_j^2}$. If we have matrix $\boldsymbol{A} \in \mathbb{R}^{n \times p}$ instead of only one column $\boldsymbol{x}$, it works the same way. We need to create matrix $\boldsymbol{Q}_{\varphi}(i,j)$ for each a_{ij} under the diagonal in order to create upper triangular matrix. Usually we are zeroing columns so that we start with a_{12} and continue with $a_{13} \dots a_{1_n}$. Then we start with the second column with element a_{23} and so on. That means we need to create in total $e = \frac{p^2-p}{2} + np - p^2$ matrices $\boldsymbol{Q}_{\varphi}(i,j)$. We can denote this sequence of matrices as $\boldsymbol{Q}_{\varphi_1}, \boldsymbol{Q}_{\varphi_2}, \dots \boldsymbol{Q}_{\varphi_e}$. The QR decomposition then looks like

$$\boldsymbol{Q}_{\varphi_e} \dots \boldsymbol{Q}_{\varphi_2} \boldsymbol{Q}_{\varphi_1} \boldsymbol{A} = \boldsymbol{R} \tag{2.30}$$

where $\boldsymbol{Q}_{\varphi_e} \dots \boldsymbol{Q}_{\varphi_2} \boldsymbol{Q}_{\varphi_1}$ is actually $\boldsymbol{Q}^T$ and $\boldsymbol{Q}$ is obtained by

$$\boldsymbol{Q} = \boldsymbol{Q^T}_{\varphi_1} \boldsymbol{Q^T}_{\varphi_2} \dots \boldsymbol{Q^T}_{\varphi_e}. \tag{2.31}$$

We can see that for the each non-zero under diagonal element of the matrix $\boldsymbol{A}$, we need one additional matrix $\boldsymbol{Q}_{\varphi_i}$, and with such matrix, we are multiply only two rows. Moreover, the rows are getting shorter after each finished column. Hence the total number of operations is at most

$$\sum_{i=1}^{n} \sum_{j=i+1}^{p} 6(n-i+1) \approx 6np^2 - 3np^2 - 3p^3 + 2p^3 = 3np^2 - p^3. \tag{2.32}$$

2.2 FAST-LTS

In this section, we introduce the FAST-LTS algorithm from [19]. It is, as well as other algorithms we introduce an iterative algorithm. We discuss all main components of the algorithm starting with its core idea called a concentration step which authors call a *C-step*.

2.2.1 C-step

We show that from an existing LTS estimate $\hat{\boldsymbol{w}}$ we can construct a new LTS estimate $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$ so that value the objective function at $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$ is less or equal to

the value at $\hat{\boldsymbol{w}}$. Based on this property, the algorithm creates a sequence of LTS estimates leading to better results.

Theorem 15. Consider $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$. Let us also have $\boldsymbol{w_0} = (m_1^{(0)}, \ldots, m_n^{(0)}) \in \mathbb{R}^p$ and $\boldsymbol{m_0} \in Q^{(n,h)}$. Let us put $L_0 = \sum_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0})$. Let $\pi : \hat{n} \to \hat{n}$ be permutation of $\hat{n}$ such that

$$|r_{\pi(1)}(\boldsymbol{w_0})| \leq \ldots \leq |r_{\pi(n)}(\boldsymbol{w_0})| \tag{2.33}$$

and mark $\boldsymbol{m_1} \in Q^{(n,h)}$ such that $m_i^{(1)} = 1$ for $i \in \{\pi(1), \pi(2), ...\pi(h)\}$ and $m_i^{(1)} = 0$ otherwise. This means that $\boldsymbol{m_1}$ corresponds to h-element subset with smallest squared residuals $r_i^2(\boldsymbol{w_0})$.

Finally let $\boldsymbol{w_1}$ be the least squares fit on the $\boldsymbol{m_1}$ h-element subset of observations and $L_1 = \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1})$, then

$$L_1 \leq L_0. \tag{2.34}$$

Proof. Because $\boldsymbol{m_1}$ represents h observations with the smallest squared residuals $r_i^2(\boldsymbol{w_0})$ at point $\boldsymbol{w_0}$, then $\sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0}) \leq \sum_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0}) = L_0$. When we take into account that the OLS estimate minimizes the sum of squared residuals for the $\boldsymbol{m_1}$ subset of observations, then

$$L_1 = \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1}) \leq \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0}). \tag{2.35}$$

Together we get

$$L_1 = \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1}) \leq \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0}) \leq \sum_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0}) = L_0 \tag{2.36}$$

□

Let us denote $\hat{\boldsymbol{w}}_{\boldsymbol{old}}^{(\boldsymbol{LTS,h,n})} = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_{old}})$ and $L_{old} = \sum_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_{old}})$ for some vector $\boldsymbol{m_{old}}$. Then based on the previous theorem, using some $\boldsymbol{m_{old}}$ with corresponding $\hat{\boldsymbol{w}}_{\boldsymbol{old}}^{(\boldsymbol{LTS,h,n})}$ we can construct $\boldsymbol{m_{new}}$ with corresponding $\hat{\boldsymbol{w}}_{\boldsymbol{new}}^{(\boldsymbol{LTS,h,n})}$ such that $L_{new} \leq L_{old}$.

Applying the theorem repetitively leads to the iterative sequence of $L_1 \leq L_2 \leq \ldots$. One step, called the *C-step*, of this process is described by the

following pseudocode.

Algorithm 2: C-step

Input: dataset consiting of $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ and $\boldsymbol{y} \in \mathbb{R}^{n \times 1}$, $\hat{\boldsymbol{w}}_{\boldsymbol{old}} \in \mathbb{R}^{p \times 1}$
Output: $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$, $\boldsymbol{m}_{\boldsymbol{new}}$

```
R ← ∅;
for i ← 1 to n do
    R ← R ∪ {|y_i − ŵ_old x_i^T|};
end
m_new ← select set of h smallest absolute residuals from R;
ŵ_new ← OLS fit the on m_new subset;
return ŵ_new , m_new;
```

C-step algorithm is visualized on Figure 2.1 where we start with h-element subset $\boldsymbol{m_1}$ and corresponding $\hat{\boldsymbol{w}}_{\boldsymbol{1}}$ OLS fit on the $\boldsymbol{m_1}$ subset and $L_1 = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_1})$. By sorting absolute the residuals and selecting h smallest we obtain $\boldsymbol{m_2}$ h-element subset. Its value $\mathrm{OF}^{(OLS,\boldsymbol{M_2X},\boldsymbol{M_2y})}(\hat{\boldsymbol{w}}_{\boldsymbol{1}})$ is highlighted with red dot. Finally we calculate OLS fit on $\boldsymbol{m_2}$ and obtain $\hat{\boldsymbol{w}}_{\boldsymbol{2}}$ estimate.

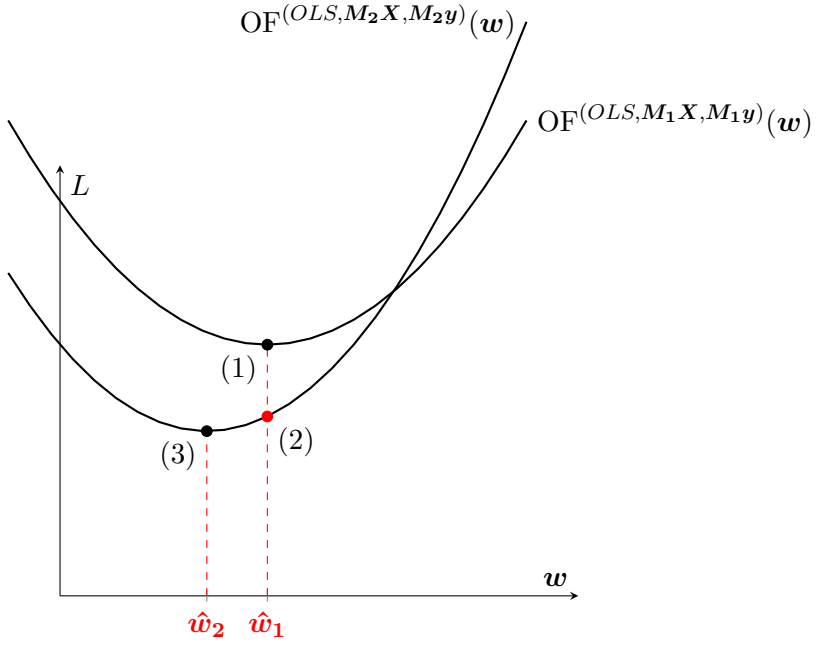


Figure 2.1: Illustration of the C-step algorithm. (1) represents the value of $\mathrm{OF}^{(OLS,\boldsymbol{M_1X},\boldsymbol{M_1y})}(\hat{\boldsymbol{w}}_{\boldsymbol{1}})$ (which is equal to the value $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_1})$). (2) represents the value of $\mathrm{OF}^{(OLS,\boldsymbol{M_2X},\boldsymbol{M_2y})}(\hat{\boldsymbol{w}}_{\boldsymbol{1}})$ and (3) represents the value of $\mathrm{OF}^{(OLS,\boldsymbol{M_2X},\boldsymbol{M_2y})}(\hat{\boldsymbol{w}}_{\boldsymbol{2}})$ (which is equal to the value $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_2})$).

Observation 16. The time complexity of the C-step 2 algorithm is asymptotically similar to the time complexity of the computation of the OLS fit,

namely $O(p^2 n)$.

Proof. In the C-step we must compute n absolute residuals. Computation of one absolute residual consists of matrix multiplication of shapes $1 \times p$ and $p \times 1$ that gives us $\mathcal{O}(p)$. Hence, the time of computing n residuals is $\mathcal{O}(np)$. Next, we must select a set of h smallest residuals which can be done in $\mathcal{O}(n)$ using a modification of the QuickSelect algorithm [20]. Finally, we must compute the $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$ OLS estimate on an h-element subset of the data. Because h is proportional to n, we can say that this is $\mathcal{O}(p^2 n + p^3)$ which is asymptotically dominant over the previous steps which are $\mathcal{O}(np + n)$. Because we assume $n \geq p$, we get $\mathcal{O}(p^2 n + p^3) \sim \mathcal{O}(p^2 n)$. □

As we stated above, repeating C-step leads to a sequence of $\hat{\boldsymbol{w}}_{\boldsymbol{1}}, \hat{\boldsymbol{w}}_{\boldsymbol{2}} \ldots$ on subsets $\boldsymbol{m}_{\boldsymbol{1}}, \boldsymbol{m}_{\boldsymbol{2}} \ldots$ with corresponding sum of squared residuals $L_1 \geq L_2 \geq \ldots$ One could ask if this sequence converges, so that $L_i == L_{i+1}$. Answer to this question is presented by the following theorem.

Theorem 17. The sequence of the estimates $\hat{\boldsymbol{w}}_{\boldsymbol{1}}, \hat{\boldsymbol{w}}_{\boldsymbol{2}} \ldots$ obtained by the C-step becomes constant after at most $k = \binom{n}{h}$, i.e. $\hat{\boldsymbol{w}}_{\boldsymbol{k}} = \hat{\boldsymbol{w}_{\boldsymbol{k+1}}}$.

Proof. $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ is uniquely given by the h-element subset $\boldsymbol{m}_{\boldsymbol{i}} \in Q^{(n,h)}$ and since $Q^{(n,h)}$ is finite, namely its size is $\binom{n}{h}$, the sequence becomes constant at the latest after this number of steps. □

The theorem gives us a clue to create algorithm described by the following pseudocode.

Algorithm 3: Repeat-C-step

Input: dataset consiting of $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ and $\boldsymbol{y} \in \mathbb{R}^{n \times 1}$, $\hat{\boldsymbol{w}}_{\boldsymbol{old}} \in \mathbb{R}^{p \times 1}$, $\boldsymbol{m_0}$
Output: $\hat{\boldsymbol{w}}_{\boldsymbol{final}}$, $\boldsymbol{m}_{\boldsymbol{final}}$

```
ŵ_new ← ∅;
m_new ← ∅;
L_new ← ∞;
while True do
    L_old ← OF_D^LTS(ŵ_old);
    ŵ_new , m_new ← C-step(X , y , ŵ_old);
    L_new ← OF_D^LTS(ŵ_new);
    if L_old == L_new then
        break
    end
    ŵ_old ← ŵ_new
end
return ŵ_new, m_new;
```

It is important to note that although the maximum number of steps of this algorithm is $\binom{n}{h}$, in practice, it is most often under 20 steps as can be seen on Figure 2.2.

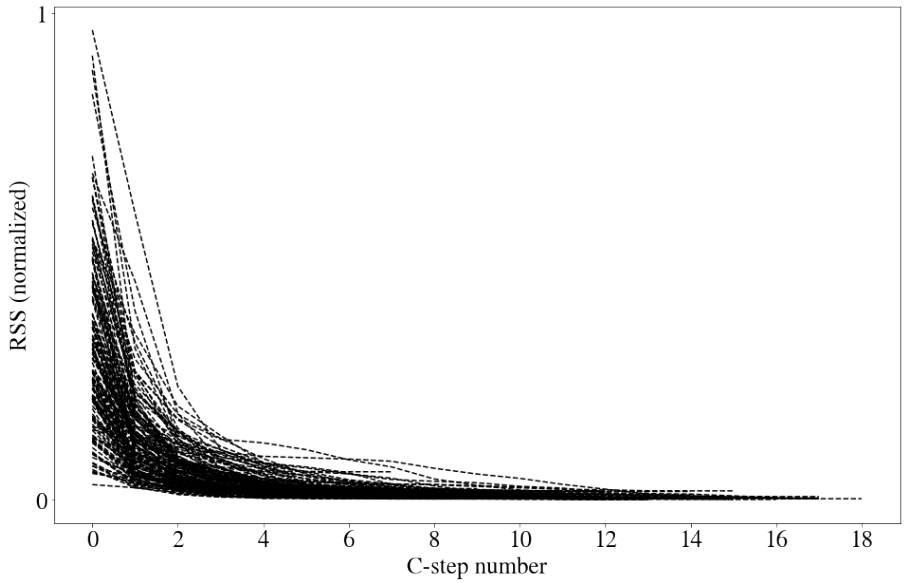


Figure 2.2: Value of the residual sum of squares (normalized) based on the number of the step of C-step algorithm for 100 different starting subsets. Dataset D3 was used with configuration $n = 500, p = 20$ and 30% of the the outliers (see Section 3 for more details about the dataset).

Now we can describe the final algorithm from [19]: Choose a lot of initial subsets $\boldsymbol{m_1}$ and for each of them apply the Repeat-C-step algorithm. From all resulting subsets with the corresponding $\hat{\boldsymbol{w}}$ estimates choose the one with the least value of $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}})$.

Before we can construct final the algorithm, we must decide how to choose the initial subset $\boldsymbol{m_1}$ and how many of them means "*a lot*".

2.2.2 Choosing an initial m_1 subset

It is important to note, that when we choose $\boldsymbol{m_1}$ subset such that it contains outliers, then iteration of C-steps usually does not converge to a good results, so we should focus on methods with non zero probability of selecting $\boldsymbol{m_1}$ such that it does not contain outliers. There are many possibilities of how to create an initial $\boldsymbol{m_1}$ subset. Let us start with the most trivial one.

Random selection

Most basic way of creating the $\boldsymbol{m_1}$ subset is simply to choose random $\boldsymbol{m_1} \in Q^{(n,h)}$. The following observation shows that it is not the best way.

Observation 18. Assume that the n-element dataset contains outliers whose number is proportional to ϵ/n with $\epsilon > 0$. Let $\boldsymbol{m_{1_1}}, \ldots, \boldsymbol{m_{1_k}}, \boldsymbol{m_{1_i}} \in Q^{(n,h)}$

be k randomly selected h-element subset of data. Then the probability that at least one of these h-element subsets does not contain an outlier tends to 0 as n goes to infinity.

Proof. It follows from observation above and the fact that $h > n/2$ that

$$\begin{aligned} P(\textit{one random data sample not an outliers}) &= (1-\epsilon) \\ P(\textit{one random } h \textit{ element subset without outliers}) &= (1-\epsilon)^h \\ P(\textit{one subset with at least one outlier}) &= 1-(1-\epsilon)^h \\ P(k \textit{ subsets with at least one outlier in each}) &= (1-(1-\epsilon)^h)^k \\ P(k \textit{ subsets with at least one subset without outliers}) &= 1-(1-(1-\epsilon)^h)^k \end{aligned}$$

Because $n \to \infty$, then $(1-\epsilon)^h \to 0$, $1-(1-\epsilon)^h \to 1$, $(1-(1-\epsilon)^h)^k \to 1$, and $1-(1-(1-\epsilon)^h)^k \to 0$ □

That means that we should consider other options for selecting initial $\boldsymbol{m_1}$ subset. Authors of the algorithm came with the following solution.

P-subset selection

Let us choose a vector $\boldsymbol{c} \in Q^{(n,p)}$ and compute the rank of the matrix $\boldsymbol{X}_C = \boldsymbol{C}\boldsymbol{X}$, where $C = diag(\boldsymbol{c})$. If $rank(\boldsymbol{X}_C) < p$ add randomly selected rows of $\boldsymbol{X}$ to $\boldsymbol{X}_C$ without repetition until $rank(\boldsymbol{X}_C) = p$. Next let us denote $\hat{\boldsymbol{w}}_{\boldsymbol{0}} = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{c})$. Let $\pi : \hat{n} \to \hat{n}$ be the permutation of $\hat{n}$ such that $|r_{\pi(1)}(\boldsymbol{c})| \leq \ldots \leq |r_{\pi(n)}(\boldsymbol{c})|$.

Finally, let $\boldsymbol{m_1} \in Q^{(n,h)}$ be initial h-element subset such that $m_i^{(1)} = 1$ for $i \in \{\pi(1), \pi(2), ...\pi(h)\}$ and $m_i^{(1)} = 0$ otherwise.

Observation 19. Assume that the n-element dataset contains outliers whose number is proportional to ϵ/n with $\epsilon > 0$. Let $\boldsymbol{c_{1_1}}, \ldots, \boldsymbol{c_{1_k}}, \boldsymbol{c_{1_i}} \in Q^{(n,p)}$ be k randomly selected p-element subset of data. Then the probability that at least one of these p-element subsets does not contain an outlier tends to

$$1-(1-(1-\epsilon)^h)^k > 0. \tag{2.37}$$

Proof. Similarly as in previous observation. □

The last missing piece of the algorithm is determining the number k of initial subsets $\boldsymbol{m_1}$, which maximize the probability to at least one of them leads to a sequence of estimates ending up in the global minimum. Simply put, the more, the better. So before we answer this question accurately, let us discuss some key observations about the algorithm.

2.2.3 Speed-up of the algorithm

In this section, we describe essential observations which help us to formulate the final algorithm. In two subsections we describe how to optimize the current algorithm.

Selective iteration

The most computationally demanding part of one C-step is computation of the OLS fit for the subset $\boldsymbol{m_i}$ and then the calculation of n absolute residuals. As we stated above, convergence is usually achieved under 20 steps. So for fast algorithm run, we would like to repeat C-step as little as possible and at the same time do not lose the performance of the algorithm.

Since the convergence of the Repeat-C-step algorithm is very fast, it turns out that we can distinguish between starts that leads to good solutions and those which does not after few C-steps iterations. Based on empiric observation, we can distinguish good or bad solution already after two or three iterations of C-steps based on the values $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{3}})$ and $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{4}})$ respectively (see Figure 2.2).

So even though authors do not specify the size of k explicitly, they propose that after a few C-steps we can choose (say 10) best solutions among all $\boldsymbol{m_1}$ starting subsets and continue with iterating the C-steps only using the best solutions. Authors refer to this process as to the *selective iteration*.

Nested extension

C-step computation is usually very fast for small n. It gets slow for very high n, say $n > 10^3$, because we need to compute the OLS estimate for the $\boldsymbol{m_i}$ subset of size h which proportional to n and then calculate n absolute residuals.

Authors came up with a solution they call a *nested extension*. Let k be number initial subsets $\boldsymbol{m_1}$. The we can describe the nested extension as follows.

- If n is greater than a given limit l, we create subset L of data $|L| = l$, and divide this subset into s disjoint sets $P_1, P_2, \ldots, P_s\,, |P_i| = \frac{l}{s}\,, P_i \cap P_j = \emptyset\,, \bigcup_{i=1}^{s} P_i = L$.
- For every P_i we set the number of starts $k_{P_i} = \frac{k}{l}$.
- Next in every P_i we create k_{P_i} number of initial $\boldsymbol{m}_{P_{i_1}}$ subsets and apply C-steps few times for each of them.
- Choose 10 best results from each subsets and merge them together. We get family of sets F_{merged} containing 10 best $\boldsymbol{m}_{P_{i_3}}$ subsets from each P_i.

- On each subset from F_{merged} family of subsets we again apply 2 C-steps twice and then choose 10 best results.
- Finally we use these 10 best subsets and apply Repeat-C-step algorithm.
- As a result we choose the best of those 10 results.

2.2.4 Putting all together

We have described all major parts of the algorithm FAST-LTS. One last thing we need to mention is that even though C-steps iteration usually converges under 20 steps, it is appropriate to introduce two parameters i_{max} and r which limits the number of C-steps iterations in some rare cases when convergence is too slow. Parameter i_{max} denotes the maximum number of iterations in the final Repeat-C-step iteration. Parameter r denotes the threshold for the stopping criterion because of the rounding errors we use $|\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) - \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})| \leq r$ instead of $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})$.

2.3 Feasible solution

In this section we introduce feasible solution algorithm from [8]. It is based on the strong necessary condition described at Definition 8. The basic idea can be described as follows.

Let us consider that we have some $\boldsymbol{m} \in Q^{(n,h)}$, then we can denote $O_m = \{i \in \{1, 2, \ldots, n\}; w_i = 1\}$ and $Z_m = \{j \in \{1, 2, \ldots, n\}; w_j = 0\}$ thus sets of indexes of positions where is 0 respectively 1 in vector $\boldsymbol{m}$. We can think about it as indexes of observations in h set and g set respectively. Then we can mark $\boldsymbol{m}^{(i,j)}$ as a vector which is constructed by swap of its ith and jth element where $i \in O$ and $j \in Z$. Such vector corresponds to vector $\boldsymbol{m}_{\boldsymbol{swap}}$ which we marked at Definition 8.

With this in mind, let us mark

$$\Delta S_{i,j}^{(\boldsymbol{m})} = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)}) - \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) \tag{2.38}$$

thus a change of the LTS objective function by swapping one observation from not trimmed subset with another from trimmed subset. To calculate this we can obviously first calculate the $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ and the $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)})$ and finally subtract both results. Although it is a option, it it computationally exhaustive. So the question is if there is easier way of calculating $\Delta S_{i,j}^{(\boldsymbol{m})}$. The answer is positive and we describe it now.

Let us mark $M = diag(\boldsymbol{m})$ and $M^{(i,j)} = diag(\boldsymbol{m}^{(i,j)})$ and also $\boldsymbol{Z_M} = (\boldsymbol{X}^T \boldsymbol{M}^T \boldsymbol{X})$ For now let us assume that we also have ${\boldsymbol{Z_M}}^{-1}$ and $\hat{\boldsymbol{w}} = {\boldsymbol{Z_M}}^{-1} \boldsymbol{X}^T \boldsymbol{M} \boldsymbol{y}$. We now want to calculate $\Delta S_{i,j}^{(\boldsymbol{m})}$. Let us mark vector of residuals $\boldsymbol{r}^{(\boldsymbol{m})} =$

$\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{w}}$ and also $d_{r,s} = \boldsymbol{x_r}{\boldsymbol{Z_M}}^{-1}\boldsymbol{x_s}$ then by equation introduced in [21] we get

$$\Delta S_{i,j}^{(\boldsymbol{m})} = \frac{(r_j^{(\boldsymbol{m})})^2(1-d_{i,i}) - (r_i^{(\boldsymbol{m})})^2(1+d_{j,j}) + 2r_i^{(\boldsymbol{m})}r_j^{(\boldsymbol{m})}d_{j,j}}{(1-d_{i,i})(1+d_{j,j}) + d_{i,j}^2}. \tag{2.39}$$

Let us now describe the core of the algorithm. It is similar to the FAST-LTS algorithm in the sense of iterative refinement of a h-element subset. So let us assume that we have some vector $\boldsymbol{m} \in Q^{(n,h)}$. No we compute $\Delta S_{i,j}^{(\boldsymbol{m})}$ for all $i \in O$ and $j \in Z$. This may lead to several different outcomes:

1. all $S_{i,j}^{(\boldsymbol{m})}$ are non-negative
2. one $S_{i,j}^{(\boldsymbol{m})}$ is negative
3. multiple $S_{i,j}^{(\boldsymbol{m})}$ are negative

In the first case, all $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)})$ are higher or the same as the $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ so none swap will lead to an improvement. That also means that strong necessary condition is satisfied and the algorithm ends.

In the second and third case, the strong necessary condition is not satisfied, and we can make the swap. In the second case, it is easy which one to choose because we have only one. In the third case, we have a couple of options again:

1. use the first swap that leads to the improvement
2. from all possible swaps choose one that has highest improvement value $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)})$
3. use the first swap that has improvement higher than some given threshold

In terms of complexity, all three options give the same results, because to find a feasible solution you need to evaluate all pair swaps. In practice, the third option is the winner because it leads to the least amount of iterations. On the other hand, as we said this does not improve the complexity of the algorithm, so from now on let us assume that we use the case number two. So if there negative $S_{i,j}^{(\boldsymbol{m})}$, we choose the one with the lowest value, make the swap and repeat the process.

The algorithm ends when there is no possible improvement, i.e., when all $S_{i,j}^{(\boldsymbol{m})}$ are non-negative. The number of iterations needed till algorithm stops is usually quite low, but for practical usage, it is still convenient to use some parameter i_{max} after which algorithm stops without finding h-element subset satisfying the strong necessary condition. One step of this algorithm

we call optimal swap additive algorithm (OSAA) is described by the following pseudocode.

Algorithm 4: OSAA

Input: $\boldsymbol{Z_M}^{-1} \in \mathbb{R}^{p\times p}$, $\boldsymbol{r^{(m)}} \in \mathbb{R}^{n\times 1}$, O_m, Z_m, $\boldsymbol{X} \in \mathbb{R}^{n\times p}$
Output: $\boldsymbol{\hat{w}_{new}}$, m_{new}

$S \leftarrow 0$;
$i_{swap} \leftarrow \emptyset$;
$j_{swap} \leftarrow \emptyset$;
for $m_i \in O_m$ **do**
  for $m_j \in Z_m$ **do**
    $r_i^{(m)} = \boldsymbol{r}_{m_i}^{(\boldsymbol{m})}$;
    $r_j^{(m)} = \boldsymbol{r}_{m_j}^{(\boldsymbol{m})}$;
    $d_{i,i} = \boldsymbol{x_{m_i}} \boldsymbol{Z_M}^{-1} \boldsymbol{x_{m_i}^T}$;
    $d_{i,j} = \boldsymbol{x_{m_i}} \boldsymbol{Z_M}^{-1} \boldsymbol{x_{m_j}^T}$;
    $d_{j,j} = \boldsymbol{x_{m_j}} \boldsymbol{Z_M}^{-1} \boldsymbol{x_{m_j}^T}$;
    $S_{tmp} = calculate\ \Delta S_{i,j}^{(\boldsymbol{m})}\ by\ (2.39)$;
    if $S_{tmp} < S$ **then**
      $S \leftarrow S_{tmp}$;
      $i_{swap} \leftarrow m_i$;
      $j_{swap} \leftarrow m_j$;
    end
  end
end
return i_{swap}, j_{swap}, S;

Observation 20. The time complexity of the OSAA 4 is $\mathcal{O}(n^2p^2)$

Proof. All $d_{i,i}$ and $d_{j,j}$ can be calculated before the for loops. To calculate $d_{i,i}$ resp. $d_{j,j}$ we need co multiply vector $\in \mathbb{R}^p$ with matrix $\in \mathbb{R}^{p\times p}$ and vector $\in \mathbb{R}^p$ that is $\mathcal{O}(p^2)$. For all $d_{i,i}$ this have to be done h times and for $d_{j,j}$ $n-h$ times. So all together it is $\mathcal{O}(np^2)$.

The two loops go through all pairs; thus it is $\mathcal{O}(n^2)$. $d_{i,j}$ can be calculated in the loop, and it can be done in $\mathcal{O}(p^2)$.

If we put everything together we get $\mathcal{O}(np^2 + n^2p^2) \sim \mathcal{O}(n^2p^2)$. □

One run of this iteration process leads to some local optimum, i.e., set satisfying the strong necessary condition. In [8] they refer to to this set as to *feasible set*. This because it is not global optima the algorithm needs to be run multiple times say t times. A h subset with the smallest value of the objective function is chosen as the final solution.

Discussion about how to find the initial h-element subset resp. initial $\boldsymbol{m}$ was already discussed when describing FAST-LTS algorithm. More importantly, as we already suggested h-element subset satisfying weak necessary

condition do not need to satisfy strong necessary condition so passing such a h-element subset as input to this algorithm is another option and we discuss it in detail later. We now describe a feasible solution algorithm (FSA) with pseudocode, and we assume that we already have some function that generates for us h-element subsets, e.g., random one.

Algorithm 5: FSA

Input: $\boldsymbol{X} \in \mathbb{R}^{n \times p}, \boldsymbol{y} \in \mathbb{R}^{n \times 1}, i_{max}, t$
Output: $\hat{\boldsymbol{w}}_{\boldsymbol{final}}, \boldsymbol{m}_{\boldsymbol{final}}$

$\hat{\boldsymbol{w}}_{\boldsymbol{final}} \leftarrow \emptyset$;
$\boldsymbol{m}_{\boldsymbol{final}} \leftarrow \emptyset$;
$R \leftarrow \emptyset$;
for $k \leftarrow 0$ ***to*** t **do**
  $\boldsymbol{m} \leftarrow generate\_intial\_subset()$; // e.g. random $\boldsymbol{m} \in Q^{(n,h)}$
  $l \leftarrow 0$;
  while $True$ **do**
    $\boldsymbol{M} \leftarrow diag(\boldsymbol{m})$;
    $\boldsymbol{Z_M} = (\boldsymbol{X}^T \boldsymbol{M}^T \boldsymbol{X})$;
    ${\boldsymbol{Z_M}}^{-1} = calculate\ inversion\ of\ \boldsymbol{Z_M}$;
    $\hat{\boldsymbol{w}} \leftarrow \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}$;
    $\boldsymbol{r}^{(\boldsymbol{m})} = \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{w}}$;
    $S_{i,j}, i, j = OSAA(\ {\boldsymbol{Z_M}}^{-1}, \boldsymbol{r}^{(\boldsymbol{m})}, O_m, Z_m, \boldsymbol{X})$;
    if $S_{i,j} \geq 0$ ***or*** $l \geq i_{max}$ **then**
      $\boldsymbol{m}_{\boldsymbol{final}} \leftarrow \boldsymbol{m}$;
      $\hat{\boldsymbol{w}} \leftarrow \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}$;
      $R \leftarrow R \cup \{\boldsymbol{m}_{\boldsymbol{final}}, \hat{\boldsymbol{w}}\}$;
      break;
    end
    else
      $\boldsymbol{m} \leftarrow \boldsymbol{m}^{(i,j)}$;
    end
  end
end
$\boldsymbol{m}_{\boldsymbol{final}}, \hat{\boldsymbol{w}}_{\boldsymbol{final}} \leftarrow$ select best from R based on smallest value of $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}$;
return $\hat{\boldsymbol{w}}_{\boldsymbol{final}}$, $\boldsymbol{m}_{\boldsymbol{final}}$;

Observation 21. In the main loop beside running OSAA which time complexity is $\mathcal{O}(n^2p^2)$ we need to recalculate $\hat{\boldsymbol{w}}$ using the matrix inversion which time complexity is $\mathcal{O}(p^2n)$ (see Observation 11). Main loop of the FSA runs up to l iterations for each start t. So the time complexity of whole algorithm is $\mathcal{O}(lt(n^2p^2 + p^2n))$. Because l and t are usually quite low, we can see that the FSA time complexity is dominated by the OSAA.

In this section, we've described the FSA algorithm. In the next section, we

introduce a very similar algorithm, but which have higher numerical stability and also contains various optimizations for higher performance.

2.4 OEA, MOEA, MMEA

In the FSA algorithm, we assumed that after each cycle of OSAA 4 we need to recalculate inversion $(\boldsymbol{X}^T\boldsymbol{X})$ together with $\hat{\boldsymbol{w}}$. In this section, we introduce a different approach described in [5] so that we can update it instead of recalculating. Moreover, these ideas lead to bounding condition of FSA which not only improves the speed but also gives options to construct different algorithms.

In Section 2.3 we introduce additive formula (2.39) which corresponds to $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)}) - \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$. Let us now try to obtain similar formula but with the focus on how the individual elements in our current algorithm change namely the inversion $\boldsymbol{Z}^{-\mathbf{1}}$ and $\hat{\boldsymbol{w}}$. We also split the idea of swap the i, j into the insertion of j and remove of i. Thus we describe how individual elements are affected after adding one row and also removing one row. Moreover, during this derivation, we obtain two different approaches to calculate it. Both are important, and we recapitulate them after.

2.4.1 Multiplicative formula

$\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{X}$ Let us denote $\boldsymbol{A} = (\boldsymbol{X}, \boldsymbol{y})$, a matrix $\boldsymbol{X}$ expanded by one column of corresponding dependent variables and and $\tilde{\boldsymbol{Z}} = \boldsymbol{A}^T\boldsymbol{A}$. Then

$$\tilde{\boldsymbol{Z}} = \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y} \end{bmatrix}. \tag{2.40}$$

Notice that $\boldsymbol{Z}$ is symmetric square matrix $\in \mathbb{R}^{p\times p}$, moreover we suppose that Z is regular. $\boldsymbol{X}^T\boldsymbol{y}$ is p dimensional column vector, $\boldsymbol{y}^T\boldsymbol{X}$ is p dimensional row vector and $\boldsymbol{y}^T\boldsymbol{y}$ is a scalar. OLS estimate $\hat{\boldsymbol{w}}$ is then given by

$$\hat{\boldsymbol{w}}^{(\boldsymbol{OLS},\boldsymbol{n})} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2.41}$$

and the RSS by

$$RSS = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}}. \tag{2.42}$$

Also, note that all necessary matrix multiplications are already in the matrix $\tilde{\boldsymbol{Z}}$.

Let us show that RSS can be expressed as ratio of determinants $\det(\tilde{\boldsymbol{Z}})$ and $\det(\boldsymbol{Z})$ (using determinant rule for block matrices) so that

$$
\begin{aligned}
RSS &= \frac{\det(\tilde{\boldsymbol{Z}})}{\det(\boldsymbol{Z})} \\
&= \frac{\det\begin{pmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y} \end{pmatrix}}{\det(\boldsymbol{M})} \\
&= \frac{\det(\boldsymbol{Z})\det\left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{Z}^{-1}\boldsymbol{y}\right)}{\det(\boldsymbol{Z})} \\
&= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}}.
\end{aligned}
\tag{2.43}
$$

If we assume that $RSS > 0$ then $\tilde{\boldsymbol{Z}}^{-1}$ can be expressed as

$$
\tilde{\boldsymbol{Z}}^{-1} = \begin{bmatrix} \boldsymbol{Z}^{-1} + \dfrac{\hat{\boldsymbol{w}}\hat{\boldsymbol{w}}}{RSS} & -\dfrac{\hat{\boldsymbol{w}}^T}{RSS} \\ -\dfrac{\hat{\boldsymbol{w}}^{\boldsymbol{T}}}{RSS} & \dfrac{1}{RSS} \end{bmatrix}. \tag{2.44}
$$

For following equations it is important to notice that for any two row vectors $\boldsymbol{c_i} = (\boldsymbol{x_i}, y_i)$ and $\boldsymbol{c_j} = (\boldsymbol{x_j}, y_j)$, $\boldsymbol{c_i}, \boldsymbol{c_j} \in \mathbb{R}^{1\times p+1}$ is

$$
\boldsymbol{c_i}\tilde{\boldsymbol{Z}}^{-1}\boldsymbol{c_i^T} = \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^2}{RSS} + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} \tag{2.45}
$$

and

$$
\boldsymbol{c_j}\tilde{\boldsymbol{Z}}^{-1}\boldsymbol{c_j^T} = \frac{(y_j - \boldsymbol{x_j}\hat{\boldsymbol{w}})(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})}{RSS} + \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} \tag{2.46}
$$

Including the observation

Using above let us express how the determinant $\det(\boldsymbol{Z})$ and the inverse of the $vecZ^{-1}$ changes when observation $\boldsymbol{c_i} = (\boldsymbol{x_i}, y_i)$ is added to the matrix $\boldsymbol{A}$. First let us notice that if we add this row to $\boldsymbol{A}$, then $\boldsymbol{Z}$ changes as

$$
\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & x_{i_1} \\ x_{21} & x_{22} & \dots & x_{2n} & x_{i_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \dots & x_{pn} & x_{i_p} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \\ x_{i_1} & x_{i_2} & \dots & x_{i_p} \end{bmatrix} = \boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{x_i^T}\boldsymbol{x_i} = \boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i}, \tag{2.47}
$$

so determinant with appended row changes as

$$
\det(\boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i}) = det(\boldsymbol{Z})(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}). \tag{2.48}
$$

Finally the inversion $\boldsymbol{Z}^{-1}$ can be obtained using Sherman-Morrison formula [22] so that

$$(\boldsymbol{Z} + \boldsymbol{x_i^T x_i})^{-1} = \boldsymbol{Z}^{-1} - \frac{\boldsymbol{Z}^{-1}\boldsymbol{x_i^T x_i Z}^{-1}}{1 + \boldsymbol{x_i Z}^{-1}\boldsymbol{x_i^T}} \tag{2.49}$$

It is now convenient to denote

$$b = \frac{-1}{(1 + \boldsymbol{x_i Z}^{-1}\boldsymbol{x_i^T})}, b \in \mathbb{R}, \tag{2.50}$$

and

$$\boldsymbol{u} = \boldsymbol{Z}^{-1}\boldsymbol{x_i^T}, \boldsymbol{u} \in \mathbb{R}^{p\times 1}. \tag{2.51}$$

Then (2.49) can be written as

$$(\boldsymbol{Z} + \boldsymbol{x_i^T x_i})^{-1} = \boldsymbol{Z}^{-1} + b\boldsymbol{u}\boldsymbol{u}^{\boldsymbol{T}}. \tag{2.52}$$

Given that last missing piece to express updated $\hat{\boldsymbol{w}}$ is appended $\boldsymbol{X}^T\boldsymbol{y}$ by one row, which can be simply expressed by same idea as (2.47) so that updated $\hat{\boldsymbol{w}}$ which we denote as $\overline{\hat{\boldsymbol{w}}}$ is

$$\overline{\hat{\boldsymbol{w}}} = (\boldsymbol{Z}^{-1} + b\boldsymbol{u}\boldsymbol{u}^{\boldsymbol{T}})(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} + y_i\boldsymbol{x_i^T}). \tag{2.53}$$

This can be simplified so that we get

$$\overline{\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} - (y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})b\boldsymbol{u}. \tag{2.54}$$

mention the mistake in original paper? w + (y - xw)bu ??

Least but not last we want to express updated RSS which we denote as $\overline{RSS}$. This can be done easily from (2.43) and (2.48) as

$$\overline{RSS} = RSS + \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^{\boldsymbol{2}}}{(1 + \boldsymbol{x_i Z}^{-1}\boldsymbol{x_i^T})}. \tag{2.55}$$

It is convenient to mark

$$\gamma^+(\boldsymbol{c_i}) = \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^{\boldsymbol{2}}}{(1 + \boldsymbol{x_i Z}^{-1}\boldsymbol{x_i^T})}, \tag{2.56}$$

so that

$$\overline{RSS} = RSS + \gamma^+(\boldsymbol{c_i}) \tag{2.57}$$

We can see that $\gamma^+(\boldsymbol{c_i})$ measures how RSS increase after we append our dataset with observation $\boldsymbol{c_i}$, thus $\gamma^+(\boldsymbol{c_i}) \geq 0$

Excluding the observation

Because we want to express both increment and decrement change in our dataset, let us now focus how RSS, $\boldsymbol{Z^{-1}}$ and $\hat{\boldsymbol{w}}$ changes after we exclude one observation.

Consider that we already included one observation $\boldsymbol{c_i}$ in our dataset and mark $\overline{\boldsymbol{Z}} = \boldsymbol{Z} + \boldsymbol{x_i x_i^T}$. If we exclude one observation $\boldsymbol{c_j} = (\boldsymbol{x_j}, y_j) \in \mathbb{R}^{p+1\times 1}$ from already updated matrix $\boldsymbol{A}$ then the determinant $\det(\overline{\boldsymbol{Z}})$ changes as

$$\det(\overline{\boldsymbol{Z}} - \boldsymbol{x_j^T x_j}) = det(\overline{\boldsymbol{Z}})(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}) \tag{2.58}$$

and the inversion changes (again, according to Sherman-Morrison formula) as

$$(\overline{\boldsymbol{Z}} - \boldsymbol{x_j^T x_j})^{-1} = \overline{\boldsymbol{Z}}^{-1} + \frac{\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T x_j}\overline{\boldsymbol{Z}}^{-1}}{1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}}. \tag{2.59}$$

Once again, it is convenient to denote

$$\bar{b} = \frac{-1}{(1\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})}, \bar{b} \in \mathbb{R}, \tag{2.60}$$

and

$$\overline{\boldsymbol{u}} = \overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}, \overline{\boldsymbol{u}} \in \mathbb{R}^{p\times 1}, \tag{2.61}$$

so that we can write

$$(\overline{\boldsymbol{Z}} + \boldsymbol{x_j^T x_j})^{-1} = \boldsymbol{Z}^{-1} - \bar{b}\overline{\boldsymbol{u}}\overline{\boldsymbol{u}}^{\boldsymbol{T}}. \tag{2.62}$$

Using the same approach as before we can denote express down-dated estimate which we denote as $\overline{\overline{\hat{\boldsymbol{w}}}}$

$$\overline{\overline{\hat{\boldsymbol{w}}}} = +\overline{\hat{\boldsymbol{w}}}(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})\bar{b}\overline{\boldsymbol{u}}. \tag{2.63}$$

Finally lets also express updated $\overline{RSS}$ which we denote as $\overline{\overline{RSS}}$. This can be done easily from (2.43) and (2.48) and (2.62) as

$$\overline{\overline{RSS}} = \overline{RSS} - \frac{(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})^{\boldsymbol{2}}}{(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})}. \tag{2.64}$$

Let us mark

$$\gamma^{-}(\boldsymbol{c_j}) = \frac{(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})^{\boldsymbol{2}}}{(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})}, \tag{2.65}$$

so that

$$\overline{\overline{RSS}} = \overline{RSS} - \gamma^{-}(\boldsymbol{c_j}). \tag{2.66}$$

Swapping two observations

Let us now express the equation for including and excluding observation at once. First, notice that from (2.62) we can express

$$\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}) = \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}) - \frac{(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2}{1+\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T})}, \tag{2.67}$$

so that

$$\det(\boldsymbol{Z}+\boldsymbol{x_i^T}\boldsymbol{x_i}-\boldsymbol{x_j^T}\boldsymbol{x_j}) = \tag{2.68}$$
$$\det(\boldsymbol{Z})(1+\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}-\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}+(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2-\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})$$

Finally we can express the $\overline{\overline{RSS}}$ as

$$\overline{\overline{RSS}} = RSS\rho(\boldsymbol{c_i},\boldsymbol{c_j}), \tag{2.69}$$

where

$$\rho(\boldsymbol{c_i},\boldsymbol{c_j}) = \frac{(1+\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}+\frac{e_j^2}{RSS})(1-\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}-\frac{e_i^2}{RSS})+(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}+\frac{e_ie_j}{RSS})^2}{1+\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}-\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}+(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2-\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}}, \tag{2.70}$$

where $e_i = y_i - x_i\hat{\boldsymbol{w}}$ and $e_j = y_j - x_j\hat{\boldsymbol{w}}$.

We can see that this formula is similar to the (2.39) but here the $\rho(\boldsymbol{c_i},\boldsymbol{c_j})$ represents multiplicative increment. Moreover $0 < \rho(\boldsymbol{c_i},\boldsymbol{c_j})$ and if $0 < \rho(\boldsymbol{c_i},\boldsymbol{c_j}) < 1$ then the swap leads to improvement in terms of feasible solution.

We are now able to modify the FSA so that we do not need to recompute $\hat{\boldsymbol{w}}$ and inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ but we can update it. Authors call this algorithm optimal exchange algorithm (OEA)

2.4.2 The OEA and its properties

We can apply theory from the previous section to the FSA. Sets O_m and Z_m (see Section 2.3) contains indexes of observations to exclude and include respectively. The matrix $\boldsymbol{Z_M}$ can be used instead of the matrix $\boldsymbol{Z}$ and $\boldsymbol{X_M}$ with $\boldsymbol{y_M}$ to calculate $\hat{\boldsymbol{w}}$.

In terms of the FSA 5 we first need to change the OSAA. There are only minor tweaks. First, we need to pass one more argument, and that is RSS. Second, We want to find minimal $\rho(\boldsymbol{c_i},\boldsymbol{c_j})$ calculated by (2.70) so that $0 < \rho(\boldsymbol{c_i},\boldsymbol{c_j}) < 1$. Other parts of the algorithm remains the same. Time complexity thus remains the same.

This algorithm produces $\rho(\boldsymbol{c_i}$, and indexes $i_{swap} \in O_m$ and $j_{swap} \in Z_m$ of observations we want to swap. Given that, we can update the RSS by (2.69), $\boldsymbol{Z_M^{-1}}$ by (2.52) and (2.62) and finally update $\hat{\boldsymbol{w}}$ by (2.54) and (2.63).

Let us now talk about the time complexity of such a solution. As we said, the time complexity of modifies OSAA is $\mathcal{O}(n^2p^2)$. Thus there is no asymptotic improvement. On the other hand, there are some significant constants outside the OSAA so let us look at how they improve.

When an improvement is found, thus when $0 < \rho(\boldsymbol{c_i}, \boldsymbol{c_j}) < 1$, then we update RSS, which is constant. Next we update inversion by (2.52) which is $\mathcal{O}(4p^2)$ and (2.62) which is also $\mathcal{O}(4p^2)$. Finally we need to update $\hat{\boldsymbol{w}}$ by (2.54) and (2.63) which are both $\mathcal{O}(p^2 + p)$.

Observation 22. Time complexity of updating all the quantities is $\mathcal{O}(8p^2 + 2p^2 + p) \sim \mathcal{O}(p^2)$. That is quite an improvement if we compare it to time complexity $\mathcal{O}(p^2 n)$ of updating those quantities in the FSA (see Observation 21).

Note that right now it actually does not matter if we use additive formula (2.39) with the stopping criterion $\Delta S_{i,j}^{(\boldsymbol{m})} \geq 0$ — thus unmodified OSAA — or multiplicative formula (2.70) with the stopping criterion $\rho(\boldsymbol{c_i}, \boldsymbol{c_j}) \geq 1$. Both results we can update RSS and other quantities can be used to calculate both formulas.

However, the advantage of the multiplicative formula (2.70) is that we can use the following bounding condition to improve the performance of the modified OSAA.

Bounding condition improvement

The $\rho(\boldsymbol{c_i}, \boldsymbol{c_j})$ is expressed as a ratio. We can see that in the numerator we have

$$(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} - \frac{e_i^2}{RSS}) + (\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} + \frac{e_i e_j}{RSS})^2, \quad (2.71)$$

and because $\frac{e_i e_j}{RSS})^2 \geq 0$ then whole numerator is greater or equal to

$$(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} - \frac{e_i^2}{RSS}). \quad (2.72)$$

On the other hand, we can see that denominator is

$$1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} + (\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2 - \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}, \quad (2.73)$$

and because $(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2$ and $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$ are actually inner products of $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ ($\boldsymbol{Z}^{-1}$ is positive definite) thus

$$(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2 \leq \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j} \quad (2.74)$$

using the Cauchy-Schwarz inequality. This means that denominator is less or equal to

$$1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}. \quad (2.75)$$

Given that we can denote $\rho_b(\boldsymbol{c_i}, \boldsymbol{c_j})$ as

$$\rho_b(\boldsymbol{c_i}, \boldsymbol{c_j}) = \frac{(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} - \frac{e_i^2}{RSS})}{1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}} \quad \leq \rho(\boldsymbol{c_i}, \boldsymbol{c_j}). \tag{2.76}$$

The actual speed improvement is then given by that we do not need to compute $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$ in each of $h(n-h)$ pairs swap comparison. As we know this quantity cannot be computed outside the loop; thus it is the reason for such a high time complexity. The modified OSAA can be further modified as follows.

First we set $\rho_{min} := 1$. Then for each pair we only compute $\rho_b(\boldsymbol{c_i}, \boldsymbol{c_j})$ and if it is greater of equal to ρ_{min} we can continue to next pair without computing $\rho(\boldsymbol{c_i}, \boldsymbol{c_j})$. It is very useful because all quantities necessary for calculating $\rho_b(\boldsymbol{c_i}, \boldsymbol{c_j})$ can be precalculated outside of the loop.

If $\rho_b(\boldsymbol{c_i}, \boldsymbol{c_j})$ is less that ρ_{min} we actually compute $\rho(\boldsymbol{c_i}, \boldsymbol{c_j})$ and set $\rho_{min} := \rho(\boldsymbol{c_i}, \boldsymbol{c_j})$.

That means in the double loop which time complexity is $\mathcal{O}(n^2)$ we do not always need to calculate $\rho(\boldsymbol{c_i}, \boldsymbol{c_j})$ which time complexity is $\mathcal{O}(n^2)$. This does not improve the asymptotic time complexity, but (as can be seen in the table) it can improve the speed of the algorithm. create experiment

Finally, let us note that in [5] authors call this algorithm with bounding condition as *modified optimal exchange algorithm* (MOEA).

2.4.3 Minimum-maximum exchange algorithm

The minimum-maximum exchange algorithm (MMEA) is very similar to the FSA respectively to its modified version the OEA. The main difference is the greediness of this algorithm. What we mean by that is that this algorithm does not find the optimal swap, but rather first find the $\boldsymbol{c_j}$ which increase the RSS by the minimum value and include this observation. Next, it finds $\boldsymbol{c_j}$ such that in decrease RSS by maximum and exclude this observation.

The minimum increase can be found by calculating $\gamma^+(\boldsymbol{c_i})$ by (2.56) for each trimmed observation in Z_m.

Next, this observation is included, so we get $h+1$ untrimmed observations. We update $\boldsymbol{Z_M^{-1}}$ to $\overline{\boldsymbol{Z_M}}^{-1}$ by (2.52) and and $\hat{\boldsymbol{w}}$ to $\overline{\hat{\boldsymbol{w}}}$ by (2.54).

Then we can find maximum $\gamma^-(\boldsymbol{c_j})$ by (2.65) among $O_m^{(i)}$ where $m^{(i)}$ denotes included observation c_i.

Next, we can update $\overline{\boldsymbol{Z_M}}^{-1}$ to the $\overline{\overline{\boldsymbol{Z_M}}}^{-1}$ by (2.62) and $\overline{\hat{\boldsymbol{w}}}$ to $\overline{\overline{\hat{\boldsymbol{w}}}}$ by (2.63).

Finally, we can update RSS to $\overline{\overline{RSS}}$ by (2.57) and (2.66). We can repeat this process until $\gamma^-(\boldsymbol{c_j}) > \gamma^+(\boldsymbol{c_i})$.

Observation 23. One step of the algorithm MMEA has time complexity $O(p^2n)$. So the whole algorithm has time complexity $\mathcal{O}(tlp^2n)$ where t is number of the starts and l is maximum number of iterations of each start.

Proof. Because we do not iterate through all pairs but only over $n-h$ followed by $h+1$ subsets the loops takes only $\mathcal{O}(n)$ time. In the loops, we are computing $\gamma^-(\boldsymbol{c_j})$ and $\gamma^+(\boldsymbol{c_i})$ both takes $\mathcal{O}(p^2)$ time. Outside the loops, all the each of the quantities we are updating take $\mathcal{O}(p^2)$ time. That means the one loop of the whole algorithm last $\mathcal{O}(p^2n)$. As in the case of the FSA, if we introduce parameters t and l, then the time complexity of the algorithm is $\mathcal{O}(tlp^2n)$ □

2.4.4 Different method of computation

In the last section we introduced a way of calculating the OEA so that we update $\hat{\boldsymbol{w}}$ and inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$. This, however, requires to compute inversion at the start of the algorithm (this is also the case for the FSA, MOEA and MMEA) As we know, calculating inversion is not practical due to low numerical stability. In practice, we usually use QR decomposition. In this section, we describe how we can modify OEA to use the QR decomposition (the same idea can also be applied to the FSA, MOEA and the MMEA).

Let us start by describing how we can update the QR factorization, which is a critical part of this modified computation. Assume that we have QR decomposition of $\boldsymbol{X}$, and we need to exchange ith observation from O_m with jth observation from Z_m. We can simulate this by inserting jth row and consequently extracting ith row from the QR decomposition.

QR insert

First, let us discuss how to update QR decomposition when a row $\boldsymbol{x_j}$ is inserted. If we add a row to $\boldsymbol{A}$ as the last row, our decomposition looks like

$$\overline{\boldsymbol{R}} = \overline{\boldsymbol{Q}}\boldsymbol{A}^{(+)} = \begin{bmatrix} \times & \cdots & \cdots & \cdots & \times \\ 0 & \times & \cdots & \cdots & \times \\ \vdots & 0 & \ddots & \cdots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & 0 & \times \\ \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \times & \cdots & \cdots & \cdots & \times \end{bmatrix} \tag{2.77}$$

Where $\overline{\boldsymbol{R}}$ denotes matrix $\boldsymbol{R}$ which is now not upper triangular and needs to be updated and $\overline{\boldsymbol{Q}} \in \mathbb{R}^{p+1\times p+1}$ denotes matrix created from $\boldsymbol{Q}$ by adding one row and one column with zeros. To preserve zeroes on the diagonal we in addition

put $\overline{\boldsymbol{Q}}_{n+1,n+1} = 1$. To update $\overline{\boldsymbol{R}}$ to upper triangular matrix $\boldsymbol{R}^{(+)}$, we need to create additional orthogonal Givens matrices $\boldsymbol{Q_{e+1}} \dots \boldsymbol{Q_{e+p}}$. Then updated upper triangular matrix is $\boldsymbol{R}^{(+)} = \boldsymbol{Q_{e+p}} \dots \boldsymbol{Q_{e+2}} \boldsymbol{Q_{e+1}} \boldsymbol{R}$.

Updated matrix $\overline{\boldsymbol{Q}}$ which we denote as $\boldsymbol{Q}^{(+)}$ is updated in the same manner, thus $\boldsymbol{Q}^{(+)} = \boldsymbol{Q} \boldsymbol{Q_e^T} \boldsymbol{Q_{e+1}^T} \dots \boldsymbol{Q_{e+p}^T}$. Note that if we do not want to have inserted row $\boldsymbol{x_j}$ as the last but as (say kth) row, then in terms of QR decomposition, we only need to move (last) $\boldsymbol{x_j}$ row to the kth position. In other words, we create a permutation matrix $\boldsymbol{P}$ so that

$$\boldsymbol{P}\boldsymbol{Q}^{(+)} = \begin{bmatrix} \boldsymbol{A}(1:k-1, 1:p) \\ \boldsymbol{x_j} \\ \boldsymbol{A}(1:k+1, 1:p), \end{bmatrix} \tag{2.78}$$

where $\boldsymbol{A}(a:b, 1:p)$ denotes rows of matrix $\boldsymbol{A}$ from a to b. We can describe the whole process by the following pseudocode.

Algorithm 6: QR insert

Input: $\boldsymbol{Q} \in \mathbb{R}^{n \times n}, \boldsymbol{R} \in \mathbb{R}^{n \times p}, \boldsymbol{x_j} \in \mathbb{R}^p, k$
Output: $\boldsymbol{Q}^{(+)} \in \mathbb{R}^{n+1 \times n+1}, \boldsymbol{R}^{(+)} \in \mathbb{R}^{n+1 \times p}$

$\boldsymbol{R}^{(+)} \leftarrow \boldsymbol{R}$ *with appended last row by* $\boldsymbol{x_j}$;
$\boldsymbol{Q}^{(+)} \leftarrow \boldsymbol{Q}$ *with appended last row and last column by zeors*;
$\boldsymbol{Q}^{(+)}{}_{n+1,n+1} \leftarrow 1$; // i.e. $Q^{(+)}$ now has 1 on the diagonal
for $i \leftarrow n$ ***to*** p **do**
  $\boldsymbol{Q}(i, n+1) \leftarrow$ *create Givens matrix* $\boldsymbol{Q}(i, n+1) \in \mathbb{R}^{n+1 \times n+1}$;
  $\boldsymbol{R}^{(+)} \leftarrow \boldsymbol{Q}(i, n+1) \boldsymbol{R}^{(+)}$;
  $\boldsymbol{Q}^{(+)} \leftarrow \boldsymbol{Q}^{(+)} \boldsymbol{Q}^T(i, n+1)$;
end
for $i \leftarrow n+1$ ***to*** k **do**
  $\boldsymbol{Q}^{(+)} \leftarrow \boldsymbol{Q}^{(+)}$ *where we swap the* i *row with the* $i-1$ *row*
end
return $\boldsymbol{Q}^{(+)}$, $\boldsymbol{R}^{(+)}$;

Observation 24. Computing $\boldsymbol{R}^{(+)}$ is $\mathcal{O}(p^2)$ and computing $\boldsymbol{Q}^{(+)}$ is $\mathcal{O}(np)$.

Proof. We have loop over p $\boldsymbol{Q}(i, j)$ Givens matrices. We do not need to create those matrices, because by multiplying $\boldsymbol{R}^{(+)}$ or $\boldsymbol{Q}^{(+)}$ with the matrix $\boldsymbol{Q}(i, j)$ only one row of is affected. That means we can simulate this matrix-matrix multiplication only by iterating over those matrices and multiplying elements with $\cos(\varphi)$ and $\sin(\varphi)$ adequately. This means in case of $\boldsymbol{R}^{(+)}$ we are iterating p times over nonzero rows of $\boldsymbol{R}^{(+)}$ thus p rows. That gives us time complexity of $\mathcal{O}(p^2)$. In the case of $\boldsymbol{Q}^{(+)}$ we are iterating p times over columns of $\boldsymbol{Q}^{(+)}$ and that gives us time complexity of $\mathcal{O}(np)$. □

Note 25. This process can also be done in case we are using an economic version of matrices $\boldsymbol{R}$ and $\boldsymbol{Q}$ thus matrices $\boldsymbol{R_1}$ and $\boldsymbol{Q_1}$ (see (2.11)). In such a case $\boldsymbol{Q_1} \in \mathbb{R}^{p \times p}$ thus then updating $\boldsymbol{Q_1}$ to $\boldsymbol{Q_1}^{(+)}$ is only $\mathcal{O}(p^2)$.

Note 26. The matrix $\overline{\boldsymbol{R}}$ can be updated to $\boldsymbol{R}^{(+)}$ without the presence of matrix $\boldsymbol{Q}$. We use this observation in the algorithm described in Section 2.6.

QR delete

When we extract the row $\boldsymbol{x_i}$ from matrix $\boldsymbol{A}$ we can use following trick [18]. First, we move such row as the first row of the matrix $\boldsymbol{A}$ so ve create permutation matrix $\boldsymbol{P}$ so that

$$\boldsymbol{PA} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{A}(1:i-1,1:p) \\ \boldsymbol{A}(1:i+1,1:p) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{A}^{(-)} \end{bmatrix} = \boldsymbol{PQR} \tag{2.79}$$

where $\boldsymbol{A}(a:b,1:p)$ means rows of matrix $\boldsymbol{A}$ from a to b. We can see that we only need to introduce zeros in the first row $\boldsymbol{q_1}$ (except q_{11}) of the matrix $\boldsymbol{Q}$. We can do this by $n-1$ matrices $\boldsymbol{Q}(i,j) \in \mathbb{R}^{n \times n}$ so that

$$\boldsymbol{Q}(1,2)\dots\boldsymbol{Q}(n-1,n)\boldsymbol{q_1^T} = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix} \tag{2.80}$$

To propagate the change into $\boldsymbol{R}$ we then consequently need to update $\boldsymbol{R}$ so that

$$\boldsymbol{Q}(1,2)\dots\boldsymbol{Q}(n-1,n)\boldsymbol{R} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{R}^{(-)} \end{bmatrix}. \tag{2.81}$$

The result is then

$$\boldsymbol{PA} = (\boldsymbol{PQQ^T}(n-1,n)\dots\boldsymbol{Q^T}(1,2))(\boldsymbol{Q}(1,2)\dots\boldsymbol{Q}(n-1,n)\boldsymbol{R}) = \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{Q}^{(-)} \end{bmatrix} \begin{bmatrix} \times \\ \boldsymbol{R}^{-} \end{bmatrix}, \tag{2.82}$$

so that

$$\boldsymbol{A}^{(-)} = \boldsymbol{Q}^{(-)}\boldsymbol{R}^{(-)} \tag{2.83}$$

Pseudocode is very similar to the algorithm QR Insert 6 so it is not necessary to provide it here.

Observation 27. Time complexity of QR delete is $\mathcal{O}(n^2)$.

Proof. We need to create $n-1$ Givens matrices, and with each of this we need to multiply $\boldsymbol{Q}^{(-)}$ and $\boldsymbol{R}^{(-)}$. As we stated in Observation 24 this can be node in $\mathcal{O}(n)$ for each matrix. So together we get $\mathcal{O}(n^2)$. □

Note 28. In case of QR delete, it is not possible to use the economic version of matrices.

Calculation of OEA using QR decomposition

Given all the required theory above, let us describe the computation. Let us start with (2.70). Here we need inversion to calculate $\boldsymbol{x_i Z^{-1} x_i^T}$, $\boldsymbol{x_j Z^{-1} x_j^T}$ and $\boldsymbol{x_i Z^{-1} x_j^T}$. But this can also be done without inversion, only using the decomposition. We can write this equation

$$\boldsymbol{x_i Z^{-1} x_i^T} = \boldsymbol{v}^T\boldsymbol{v} \iff \boldsymbol{x_i}(\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{x_i^T} = \boldsymbol{v}^T\boldsymbol{v} \tag{2.84}$$

where $\boldsymbol{v}$ can be obtained by solving lower triangular system

$$\boldsymbol{R}^T\boldsymbol{v} = \boldsymbol{x_i}^T. \tag{2.85}$$

The same can be done with $\boldsymbol{x_j Z^{-1} x_j^T}$. Least but not last we need to solve $\boldsymbol{x_i Z^{-1} x_j^T}$. We can write this

$$\boldsymbol{x_i Z^{-1} x_j^T} = \boldsymbol{x_j u} \iff \boldsymbol{x_i}(\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{x_j^T} = \boldsymbol{x_j}^T\boldsymbol{u}, \tag{2.86}$$

where column vector $\boldsymbol{u}$ is defined by (2.51). We can see that

$$\boldsymbol{u} = \boldsymbol{Z^{-1} x_i^T} \iff (\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{x_i^T} = \boldsymbol{u} \tag{2.87}$$

so that $\boldsymbol{u}$ can be obtained by solving the upper triangular system

$$\boldsymbol{Ru} = \boldsymbol{v}, \tag{2.88}$$

where $\boldsymbol{v}$ is solution of (2.85). Other quantities of (2.70) does not require $\boldsymbol{Z}^{-1}$.

Observation 29. We can see that time complexity of all these calculations is the same as in case of calculating with inversion thus $\mathcal{O}(p^2)$.

Proof. Indeed, in the case of the inversion, we are multiplying quantities such as $\boldsymbol{x_i Z^{-1} x_i^T}$ which can be done in $\mathcal{O}(p^2)$. In the case of decomposition, we solve this problem by solving the triangular system of p equations. This can also be done in $\mathcal{O}(p^2)$. □

When optimal exchange $\boldsymbol{c_i}, \boldsymbol{c_j}$ is found then we need to update RSS which can be done same way by (2.69). Updating $\hat{\boldsymbol{w}}$ to $\overline{\hat{\boldsymbol{w}}}$ by (2.54) requires $\boldsymbol{u}$ which in this case we calculate by (2.88) and b which requires $\boldsymbol{x_j Z^{-1} x_j^T}$. This can be done in the same manner as (2.84). Analogous operations can be used to update $\overline{\hat{\boldsymbol{w}}}$ to $\overline{\overline{\hat{\boldsymbol{w}}}}$ by means of (2.63). Only thing wee need to realize that we can express (2.67) as $\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T} = \boldsymbol{x_i Z^{-1} x_i^T} + b(\boldsymbol{u x_j}^2)$.

On the other hand, we cannot use equation (2.52) and (2.62) for updating the inversion, because we do not have one. So we need to update our QR decomposition somehow. We describe algorithms for updating the QR decomposition in Section 2.4.4. First, we can use the QR Insert algorithm to add row $\boldsymbol{x_j}$ to the decomposition and consequently QR delete to extract $\boldsymbol{x_j}$ from the decomposition.

Observation 30. This approach is slower than updating inversion directly. On the other hand, this solution is numerically stable. Updating inversion can be done in $\mathcal{O}(p^2)$ (see Observation 22) while QR insert has $\mathcal{O}(np)$ (see Observation 24) and time complexity and QR delete even $\mathcal{O}(n^2)$ (see Observation 27).

Note 31. Because time complexity of the QR delete is $\mathcal{O}(n^2)$, it is up to consideration if instead of recycling QR decomposition is not worth it to recalculate it from scratch which takes $\mathcal{O}(p^2 n)$ (see (2.20)).

Finally let us talk about matrix $\tilde{\boldsymbol{Z}}$ (2.40) which we used for derivation of our equations. If we use Observation 2.19 then we realize that if we make QR factorization of $\boldsymbol{A} = (\boldsymbol{X}, \boldsymbol{y})$ then

$$\tilde{\boldsymbol{Z}} = \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^{\boldsymbol{T}} & 0 \\ \boldsymbol{\phi}^{\boldsymbol{T}} & r \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{\phi} \\ 0 & r \end{bmatrix} \tag{2.89}$$

where

$$\tilde{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{\phi} \\ 0 & r \end{bmatrix} = \tilde{\boldsymbol{Q}^{\boldsymbol{T}}}\boldsymbol{A} \tag{2.90}$$

is matrix from QR factorization of $\boldsymbol{A}$. Next, we can realize that $\boldsymbol{R} \in \mathbb{R}^{p \times p}$ is matrix $\boldsymbol{R}$ which we can obtain by QR factorization of $\boldsymbol{X}$. Moreover $\boldsymbol{\phi} \in \mathbb{R}^{p \times 1}$ is column vector which is actually equal to

$$\boldsymbol{\phi} = \boldsymbol{Q}^{\boldsymbol{T}}\boldsymbol{y} \tag{2.91}$$

where $\boldsymbol{Q}$ is matrix $\boldsymbol{Q}$ from QR factorization of $\boldsymbol{R}$. Due to this fact $\hat{\boldsymbol{w}}$ is solution of upper triangular system

$$\boldsymbol{R}\hat{\boldsymbol{w}} = \boldsymbol{\phi} \tag{2.92}$$

Finally $r \in \mathbb{R}$ is scalar such that

$$r^2 = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}} = RSS \tag{2.93}$$

Remark 32. We can see that all the quantities we use in the algorithm can be obtained from matrix $\tilde{\boldsymbol{R}}$.

Now we have described both versions of calculation the algorithm OEA. As we already stated, it easy to use this approach on algorithms FSA, MOEA, and MMEA. Using inversion $\boldsymbol{Z}^{-1}$ is slightly faster, but numerically less stable, second, using QR decomposition is more numerically stable. Asymptotically both approaches provide the same performance. Finally, we have also shown that using matrix $\tilde{\boldsymbol{R}}$ is useful because it contains all necessary quantities for the algorithms.

2.5 Combined algorithm

In this section, we shortly describe how we can utilize Lemma 10; thus that strong necessary condition is not satisfied unless the weak necessary condition is satisfied.

Algorithm FAST-LTS outputs h element subset satisfying the weak necessary condition. One step of this algorithm has time complexity $\mathcal{O}(p^2n)$ (see Observation 16). The iteration of this step is quite low and moreover usually limited by parameter.

On the other hand, algorithms for finding strong necessary condition FSA, OEA and MOEA have time complexity of one step $\mathcal{O}(p^2n^2)$ (we now don't take into account eager version MMEA, because we have no proof that it finds h element subset satisfying strong necessary condition).

We can use this in our favor so that we can find h element subset satisfying weak necessary condition by the FAST-LTS algorithm and consequently use this h element subset as an input to some of the algorithms which find h element subset satisfying the strong necessary condition.

Let us consider the combination of the FAST-LTS and the MOEA. We have two options on how to perform this: z

1. Let the FAST-LTS converge and use this h element subset as input to the MOEA which we let converge.

2. Let the FAST-LTS converge to some h element subset. Perform one step of the MOEA and use the result as the input to the FAST-LTS. We can then iterate between steps of these two algorithms till convergence.

On the large data sets, where even one step of the MOEA is too exhausting, we can use the eager MMEA which time complexity of one step is lower than in the case of MOEA.

In Chapter 3 we show the experimental results of various combinations of these algorithms.

2.6 BAB algorithm

In this section, we describe the algorithm from [5].Very similar algorithm can also be found in [23].

Unlike previous algorithms, this one is exact. As we discussed in Section 1.4.1, exact algorithm calculate OLS fit on all of the $\binom{n}{h}$ h element subsets. For larger data sets this approach is computationally prohibitive.

This version of the exact algorithm is based on the branch and bound design paradigm; thus it tries to avoid exhaustive computation on all h element subsets. First, let us describe how the combination tree is built.

Let us denote subset of indexes $J_k = (j_1, j_2, \ldots, j_k) \subset 1, 2, \ldots, n$. We can then mark $\boldsymbol{X}_{\boldsymbol{J_k}}$ and $\boldsymbol{Y}_{\boldsymbol{J_k}}$ matrices created from $\boldsymbol{X}$ and $\boldsymbol{Y}$ so that all rows that

are not indexed by J_k are removed. We can see that the number of subsets is given by $\binom{n}{k}$.

Let us now consider tree such that at level k is $\binom{n}{k}$ nodes representing all J_k subsets. The depth of the tree is h, so this tree has $\binom{n}{h}$ leaves representing all h index subsets. Example of such a tree we can see at Figure 2.3.

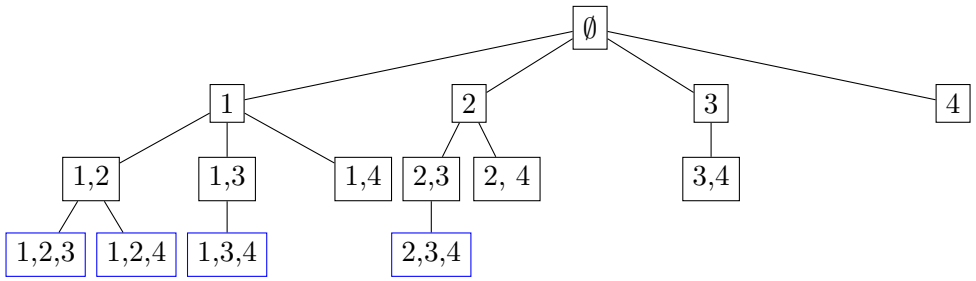


Figure 2.3: Tree consisting of all element subsets for $n = 4$ and $h = 3$. Leaves with blue border color represents h-element subsets.

In the case of the exhaustive approach, we can then traverse the tree starting in the root using the right to left (RTL) preorder traversal, and in each node, we can calculate the OLS fit on $\boldsymbol{X}_{\boldsymbol{J_k}}$ and $\boldsymbol{Y}_{\boldsymbol{J_k}}$. If we are in the level $k = h$, we calculate the fit on the h element subset. This approach is exhaustive, so let us now describe how we can improve efficiency.

First, it is not necessary to calculate fit on kth level of the tree in a case $k < p$ because this would mean that matrix $\boldsymbol{X}_{\boldsymbol{J_k}}$ is not regular. The main improvement can because we can skip parts of the tree (perform pruning) based on the following criteria.

If at any step (at any node) of the tree traversal we calculate the $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k)$ where $m_{k_i} = 1$ when $k_i \in J_k$ and $m_{k_i} = 0$ and we find out that this value is higher than minimal value found for some $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_h)$ thus value of OLS fit on some h element subset, then we can discard all subsets which contain J_k. That means we can trim all descendants of the node which represent J_k subset.

We can describe the algorithm as follows:

1. Set $RSS^* = \inf$. m^*
2. For each node in RTL preorder traversal calculate $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k)$
3. If $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k) > RSS^*$, skip all siblings of this node in the traversal.
4. If $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k) < RSS^*$ and $k = h$, then set $RSS^* = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k)$ and m^* so that $m_{k_i} = 1$ when $k_i \in J_k$ and $m_{k_i} = 0$.
5. After the traversal return m^* and $\hat{\boldsymbol{w}}$

During the traversal of the tree, we only need to know the path back to the root. Thus the whole tree does not have to be in the memory.

Remark 33. If we calculated $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_k)$ than we do not need to calculate $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m_{k+1})$ from scratch. In Section 2.4 we described the way how to update it when a row is added using the matrix inversion any we can apply it here. In Section 2.4.4 we described the way of doing this using the matrix decomposition which is another option.

Moreover, we described that if rows are not extracted and only inserted, then matrix $\boldsymbol{Q}$ do not have to be present and only matrix $\boldsymbol{R}$ can be updated. We can use this in our favor because we can keep all the decompositions on the path from the root in the memory. If we use matrix $\tilde{\boldsymbol{Z}}$ and $\tilde{\boldsymbol{R}}$ we can (see Remark 32) calculate all the quantities required for the update only using the matrix $\tilde{\boldsymbol{R}}$. This means that both versions of updating are possible and both can be done in $\mathcal{O}(p^2)$. Thus we recommend using the matrix decomposition version of the updating because it can provide higher numerical stability at the same speed.

2.6.1 Improvements

Because the tree is pruned based on the smallest value RSS^*, it would be convenient if we were able to obtain small value RSS^* early in the traversal. We can obtain this value by calculating an approximative LTS estimate. This can be done using any probabilistic algorithm described in the previous sections of this chapter. An ideal candidate may be some combined algorithm described in Section 2.5. When we find such small value, it is then also convenient to permutate the observations based on its absolute residuals based on this LTS estimate in the decreasing order.

Another improvement can be made using the following sorting rule *sorting rule*: If we are at the level k (we assume that $k > p$) in the node J_k and this node has s siblings indexed as $s_1 \dots s_s$ then we can change the order of those siblings first by calculating partial increment (2.56) for each sibling and consequently ordered them in the descending order from left to right. This means we visit siblings with a lowest partial increment first. Trimming can then be done in the same manner. Moreover, if we calculate RSS in one of those siblings we and realize that $RSS > RSS^*$, then we can on the top of trimming all siblings of this sibling trim also all brothers left to this sibling. That means we can trim parent node J_k because all of his siblings have already been explored or can be trimmed. We can see such a procedure at Figure 2.4.

2.7 BSA algorithm

In this section, we'll introduce another exact algorithm. It has a little different approach than previous algorithms. First of all, build a theoretical basis for

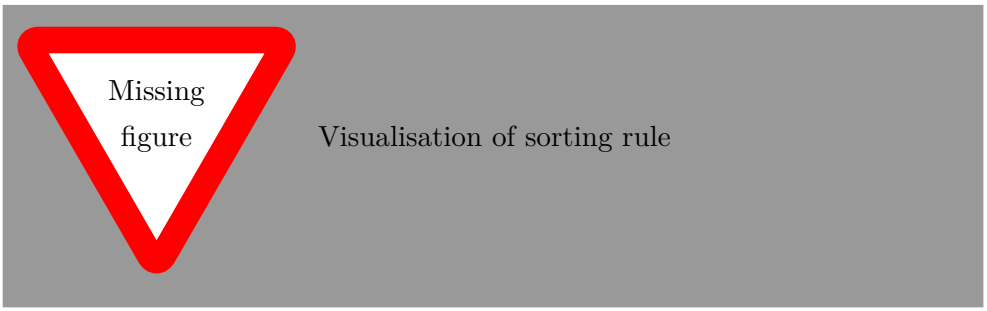


Figure 2.4: Step in the algorithms when due to the sorted siblings node j_k can be trimmed

this algorithm.

2.7.0.1 Domain of OF-LTS

We introduced two version of OF-LTS. First one is $\mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}), \boldsymbol{w} \in \mathbb{R}^p$ and the second is discrete version $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}), \boldsymbol{m} \in Q^{(n,h)}$. We also know that $\min_{\boldsymbol{w}\in\mathbb{R}^p} \mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}) = \min_{\boldsymbol{m}\in Q^{(n,h)}} \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$.

(see (1.27)) Let us now talk about the non-discrete version and introduce some new feature.

Definition 34. Let $Z \subset \mathbb{R}^p \times Q^{(n,h)}$ be an relation defined as

$$(\boldsymbol{w}, \boldsymbol{m}) \in Z \Leftrightarrow \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) = \sum_{i=1}^{n} m_i r_i^2(\boldsymbol{w}) \tag{2.94}$$

Z is not a mapping. To show this we can take simple example so that $r_h^2 = r_{h+1}^2$. Then we have for the given $\boldsymbol{w}$ two different vectors $\boldsymbol{m}$ that are in relation with it.

For that reason, let us define $\mathcal{U} \subset \mathbb{R}^p$ as a maximal set where Z is a mapping. Next we define $\mathcal{H} = \mathbb{R}^p \setminus \mathcal{U}$ as a complement of $\mathcal{U}$.

Let's now describe some properties based on which $\boldsymbol{w}$ is in $\mathcal{U}$ or $\mathcal{H}$ set.

Theorem 35. $\boldsymbol{w} \in \mathcal{U}, \boldsymbol{w} \in \mathbb{R}^p$ if and only if $r_{(h:n)}^2(\boldsymbol{w}) < r_{(h+1:n)}^2(\boldsymbol{w})$

Proof. As shown in example above. If $r_i^2(\boldsymbol{w}) = r_{(h:n)}^2(\boldsymbol{w}) = r_{(h+1:n)}^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w}), i, j \in \{1, 2, \ldots, n\}$ then $(\boldsymbol{w}, \boldsymbol{m_1}) \in Z$ and also $(\boldsymbol{w}, \boldsymbol{m_2}) \in Z$ where $\boldsymbol{m_1}$ has ones at indexes of h smallest residuals and $\boldsymbol{m_2}$ has ones at the same indexes except swap ith one with jth zero. □

Corollary 36. Complement of vectors $\boldsymbol{w}$ from Theorem 35 are vectors such that

$$r_{(h:n)}^2(\boldsymbol{w}) = r_{(h+1:n)}^2(\boldsymbol{w}) \tag{2.95}$$

thus

$$\mathcal{H} = \{\boldsymbol{w} \in \mathbb{R}^p | r^2_{(h:n)}(\boldsymbol{w}) = r^2_{(h+1:n)}(\boldsymbol{w})\}. \tag{2.96}$$

That means that for each $\boldsymbol{w} \in \mathcal{H}$ there are two different $(\boldsymbol{x_i}, y_i)$ and $(\boldsymbol{x_j}, y_j)$ so that

$$(y_i - \boldsymbol{x_i w})^2 = r_i^2(\boldsymbol{w}) = r^2_{(h:n)}(\boldsymbol{w}) = r^2_{(h+1:n)}(\boldsymbol{w}) = r_j^2(\boldsymbol{w}) = (y_j - \boldsymbol{x_j w})^2. \tag{2.97}$$

We can see that

$$(y_i - \boldsymbol{x_i w})^2 = (y_j - \boldsymbol{x_j w})^2 \iff y_i \pm y_j + (\boldsymbol{x_i} \pm \boldsymbol{x_j})\boldsymbol{w} = 0 \tag{2.98}$$

Assumption 37. For $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ let as assume that for all $i, j, \in \{1, 2 \ldots, n\}$ if $i \neq j$ then $\boldsymbol{x_i} \neq \pm \boldsymbol{x_j}$ and $\|\boldsymbol{x_i}\| \neq 0$.

If Assumption 37 is fulfilled then $y_i \pm y_j + (\boldsymbol{x_i} \pm \boldsymbol{x_j})\boldsymbol{w} = 0$ represents a hyperplane.

It's easy to show that set $\mathcal{U}$ is an open and Lebesgue measure of $\mathcal{H}$ is 0 [24]. Moreover $\mathcal{H}$ splits $\mathbb{R}^p$ into finite number of m open disjoint subsets of $\mathcal{U}$.

Definition 38. Let's define set of $m \in \mathbb{N}$ sets $\mathcal{U}^{(set)} = \{U_i\}_{i=1}^m$ so that all U_i are open and connected sets, U_i U_j are disjoined thus $U_i \cap U_j = \emptyset$, $\cup_{i=1}^m U_i = \mathcal{U}$ and $\cup_{i=1}^m \partial U_i = \mathcal{H}$.

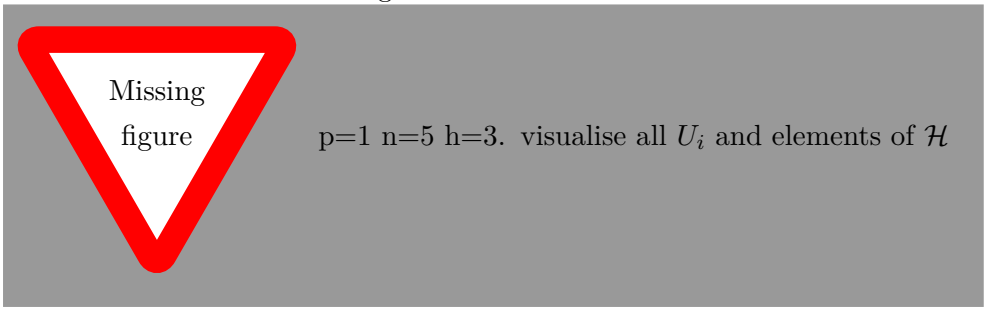
We define neighbor sets $U_i, U_j, i \neq j$ so that $U_i \cap U_j \neq \emptyset$.

We also define $M^{(min)}$ set of m vectors $\boldsymbol{m} \in Q^{(n,h)}$ so that

$$M^{(min)} = \{\boldsymbol{m_1}, \ldots, \boldsymbol{m_m} | \boldsymbol{m_i} = Z(\boldsymbol{w})\boldsymbol{w} \in U_i\}.$$

where $Z(\boldsymbol{w})$ represent unique vector $\boldsymbol{m_i}$. That means $Z(\boldsymbol{w}) = Z(\boldsymbol{w'})$ for all $w, w' \in U_i$.

For a better understanding definition above see



We say that matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has *h-full* if for all $m \in Q^{(n,h)}$ matrix $\boldsymbol{MX}$ has rank p.

Theorem 39. If the matrix $\boldsymbol{X}$ has h-full rank then for every local minima at point satisfying weak-necessary condition $\boldsymbol{w_0}$ of the OF-LTS function, then there exists a vector $\boldsymbol{m} \in Q^{(n,h)}$ such that $(\boldsymbol{w_0}, \boldsymbol{m}) \in Z$.

Proof can be found in [24, Theorem 7].

2.7.1 One-dimensional version of the algorithm

Based on the Theorem 39 we can see that

$$\min_{\boldsymbol{m}\in Q^{(n,h)}} \mathrm{OF}^{(OLS,\boldsymbol{MX},\boldsymbol{MY})}(\boldsymbol{w}) = \min_{m\in M^{(min)}} \mathrm{OF}^{(OLS,\boldsymbol{m^iX},\boldsymbol{m^iY})}(\boldsymbol{w}) \tag{2.99}$$

where $\boldsymbol{M} = diag(\boldsymbol{m})$, $\boldsymbol{m^i} = diag(m)$. Set $M^{(min)}$ is very useful because we can iterate only through this set and not through whole $Q^{(n,h)}$.

Then minimizing objective function $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ (1.31) can be reformulated as

$$\min_{\boldsymbol{m}\in Q^{(n,h)}} \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) = \min_{m\in M^{(min)}} \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m) \tag{2.100}$$

That means if we can find all $m \in M^{(min)}$ then minimizing $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}$ would be easy. So how we can find the elements of $M^{(min)}$ set?

For now, let us assume that we know elements of $\mathcal{H}$.

Because for each $m \in M^{(min)}$ exists at least one $\boldsymbol{w} \in \mathcal{H}$ such that $(\boldsymbol{w}, \boldsymbol{m}) \in Z$ then for given $\boldsymbol{w} \in \mathcal{H}$ we can find all $\boldsymbol{m}$ by algorithm described by following pseudocode:

write the pseudocode

Algorithm 7: Find all $\boldsymbol{m}$

Input: $\boldsymbol{Q} \in \mathbb{R}^{n\times n}, \boldsymbol{R} \in \mathbb{R}^{n\times p}, \boldsymbol{x_j} \in \mathbb{R}^p, k$
Output: $\boldsymbol{Q^{(+)}} \in \mathbb{R}^{n+1\times n+1}, \boldsymbol{R^{(+)}} \in \mathbb{R}^{n+1\times p}$
1 **return** $\boldsymbol{Q^{(+)}}$, $\boldsymbol{R^{(+)}}$;

Finally we need to find all elements of $\mathcal{H}$. As we know for all elements $\boldsymbol{w} \in \mathcal{H}$ $r^2_{(h:n)}(\boldsymbol{w}) = r^2_{(h+1:n)}(\boldsymbol{w})$. So if we find all $\boldsymbol{w}$ that solve this, then we found $\mathcal{H}$. The problem is that residuals are sorted. To overcome this issue by finding some candidates which would possible be equal $(h:n)$th and $(h+1:n)$th squared residuals. In [24] we are provided necessary condition for such a candidates.

Definition 40. If $\boldsymbol{w} \in \mathcal{H}$ then there exists $i, j, i \neq j$ so that $r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w})$. We denote set H containing $\boldsymbol{w}$ satisfying this necessary condition, thus

$$H = \{\boldsymbol{w} \in \mathbb{R}^p | r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w}), i \neq j\}. \tag{2.101}$$

Note that $\binom{|H|\leq 2}{np+1}$. So in the case of the one dimensional case $\binom{|H|\leq 2}{n2=n(n-1)}$.

Finding all elements of H is then equal for for solving $\binom{}{np+1}$ equations. Moreover, because equations are quadratic, we can have up to two solutions for each equation.

So idea of the whole algorithm is to find all elements of H (by solving quadratic equations), consequently finding which of them are part of $\mathcal{H}$ (by ordering squared residuals and checking if $r^2_{(h:n)}(\boldsymbol{w}) = r^2_{(h+1:n)}(\boldsymbol{w})$ and finally for each $\boldsymbol{w} \in \mathcal{H}$ find $\boldsymbol{m}$ that are in relation. All of those $\boldsymbol{m}$ vectors form $M^{(min)}$ set. Finally we can use $\boldsymbol{m} \in M^{(min)}$ for minimizing $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ by the terms of (2.100). The whole algorithm, which authors call Border Scanning Algorithm (BSA) is described by the following pseudocode:

Algorithm 8: BSA

Input: $\boldsymbol{Q} \in \mathbb{R}^{n \times n}, \boldsymbol{R} \in \mathbb{R}^{n \times p}, \boldsymbol{x_j} \in \mathbb{R}^p, k$

Output: $\boldsymbol{Q}^{(+)} \in \mathbb{R}^{n+1 \times n+1}, \boldsymbol{R}^{(+)} \in \mathbb{R}^{n+1 \times p}$

write the pseudocode

1 **return** $\boldsymbol{Q}^{(+)}$, $\boldsymbol{R}^{(+)}$;

2.7.2 Multidimensional BSA

If we would like to scale the algorithm into the higher dimension, we have to face one major complication. The algorithm finds all elements of H and $\mathcal{H}$. This is possible in one dimension, but because $\mathcal{H}$ forms a hyperplane, then for higher dimensions is infinite.

Authors then propose to find finite set $mathcal H_p$ such that

$$\forall \boldsymbol{m} \in M^{(min)} \exists \boldsymbol{w} \in \mathcal{H}_p : (\boldsymbol{w}, \boldsymbol{m}) \in Z \tag{2.102}$$

and also finite set H_p, so that $mathcal H_p \subset H_p \subset H$. We can see that for $p = 1$ we need only one equation so the solution has dimension 0. If we would like to have some solution with dimension equal to 0 for $p > 1$ we need system of p equations. Moreover this system of equations have to be independent. So we can consider system of p equations of type $r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w})$

$$\begin{aligned} r_{i_1}^2(\boldsymbol{w}) &= r_{i_2}^2(\boldsymbol{w}) \\ r_{i_2}^2(\boldsymbol{w}) &= r_{i_3}^2(\boldsymbol{w}) \\ \vdots \qquad & \qquad \vdots \\ r_{i_p}^2(\boldsymbol{w}) &= r_{i_{p+1}}^2(\boldsymbol{w}) \end{aligned} \tag{2.103}$$

where $i_1, i_2 \dots i_{p+1}$ is $(p+1)$-element subset of $1, 2, \dots n$.

System of those quadratic equations can be rewritten as Because this is a system of quadratic equations, we can have up to 2^p solutions.

Moreover total number of $(p+1)$-element subsets out of n is $\binom{n}{p+1}$. This means that the total number of solutions can be up to $\binom{n}{p+1}2^p$. We denote set of all these solutions H_p.

So if set H_p satisfies (2.102) then it can be used for finding $\mathcal{H}_p$ subset, thus we would be able to use this set for the multidimensional version of the algorithm.

Let $\circ \in +, -$ be either operation $+$ or $-$. Then system of quadratic equations (2.103) is equivalent to 2^p linear systems

$$\begin{aligned} (\boldsymbol{x}_{i_1} \circ_1 \boldsymbol{x}_{i_2})^T \boldsymbol{w} &= y_{i_1} \circ_1 y_{i_2} \\ \vdots \qquad & \qquad \vdots \\ (\boldsymbol{x}_{i_p} \circ_p \boldsymbol{x}_{i_{p+1}})^T \boldsymbol{w} &= y_{i_p} \circ_p y_{i_{p+1}}. \end{aligned} \tag{2.104}$$

Moreover (2.104) is equivalent to the set of equations

$$\begin{aligned}(\boldsymbol{x}_{i_1} \circ_1 \boldsymbol{x}_{i_2})^T \boldsymbol{w} &= y_{i_1} \circ_1 y_{i_2} \\ \vdots \qquad &\vdots \\ (\boldsymbol{x}_{i_1} \circ_p \boldsymbol{x}_{i_{p+1}})^T \boldsymbol{w} &= y_{i_1} \circ_p y_{i_{p+1}}.\end{aligned} \tag{2.105}$$

Elements of set $\mathcal{H}_p$ then satisfies $r_{i_1^2}(\boldsymbol{w}) = r_{(h:n))}(\boldsymbol{w}) = r_{(h+1:n))}(\boldsymbol{w})$.

Set H_p was proven to be suitable for finding $\mathcal{H}_p$ in [24] thus that set $\mathcal{H}_p$ satisfies (2.102) if following assumptions are satisfied (we already assume that Assumption 37 holds). First is that for all $\boldsymbol{w}$ $r_{(h:n))}(\boldsymbol{w}) > 0$, second , for all $(\circ_1, \circ_2 \ldots, \circ_{n-1}) \in \times_{i=1}^{n} +, -$ the matrix system of $n-1$ equations

$$\begin{aligned}(\boldsymbol{x}_{i_1} \circ_1 \boldsymbol{x}_{i_2})^T \boldsymbol{w} &= y_{i_1} \circ_1 y_{i_2} \\ \vdots \qquad &\vdots \\ (\boldsymbol{x}_{i_1} \circ_{n-1} \boldsymbol{x}_{i_n})^T \boldsymbol{w} &= y_{i_1} \circ_p y_{i_n}..\end{aligned} \tag{2.106}$$

The seconds assumption prevents the set $\mathcal{H}_p$ to be empty thus the case where all systems (2.105) are dependent.

This assumption can cause problem because when we assume model with the intercept, thus for all rows $\boldsymbol{x_i}$ of matrix $\boldsymbol{X}$ is $x_{i1} = 1$ then in the case $\circ_1, \circ_2 \ldots, \circ_{n-1} = -$, first column of matrix $\boldsymbol{X}$ only contains zeroes. We can modify this assumption so that instead of using $(\circ_1, \circ_2 \ldots, \circ_{n-1}) \in \times_{i=1}^{n} +, -$ we use only $(\circ_1, \circ_2 \ldots, \circ_{n-1}) \in \times_{i=1}^{n} +, - \setminus (-, \ldots, -)$ [24].

Given that we now know that set H_p and $\mathcal{H}_p$ are suitable for our algorithm and we also know how to calculate them, so let us describe the multidimensional version of the algorithm.

write the pseudocode

Let us now describe the time complexity of this algorithm.

write the time complexity

Chapter 3

Experiments

In this chapter we describe experiments and their results for our implementation of all algorithms described in the previous chapter. For testing performance of algorithms we have implemented data set generator which provide wide variety of configuration options of generating the data.

3.1 Data set generator

When we want to generate n observations without outliers that satisfies linear regression model we can do it as follows:

Algorithm 41 (Generate clean data)**.**

1. Generate regression coefficients $\boldsymbol{w} = (w_1, \ldots, w_p)$ and set σ^2.
2. Generate explanatory variable $\boldsymbol{x_i}$.
3. Generate random noise $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$.
4. Simulate dependent variable $y_i = \boldsymbol{w}^T \boldsymbol{x_i} + \varepsilon_i$.
5. Repeat steps 2–4 n times.

As a result we obtain $\boldsymbol{X}$ and $\boldsymbol{y}$. Regression coefficients can be set to arbitrary values, but if we do not want to have dummy explanatory variables, then they are supposed to be non-zero. All $\boldsymbol{x_i}$ should be generated independently and it is very common that we generate all $\boldsymbol{x_i}$ from normal distribution.

Another thing that needs to be considered is the intercept. In this work we assumed that our data already include intercept, so in that case w_1 is equal to intercept and all x_{i_1} should be equal to 1. Note that same result can be obtained by generating data without intercept and with $\varepsilon_i \sim \mathcal{N}(\mu, \sigma^2)$. We can then extend matrix $\boldsymbol{X}$ so that we add first column that contains only 1s. This approach is very common and software for estimating regression coefficients usually allows to set parameter which determines if intercept should be

used; if so, the column of 1s is added. For that reason we generate our data sets using this approach. That means we generate $\varepsilon_i \sim \mathcal{N}(\mu, \sigma^2)$ and column of 1s is included only in case we set parameter for using intercept when fitting the data set.

3.1.1 Generating outliers

As we have already described in Section 1.3.1 we can describe different types of the outliers: vertical outliers and two types leverage points — good leverage points and bad leverage points. Those types of outliers are visualized on Figure 3.1. We can see that good leverage points are not deemed as an outliers here, even if they are distant observations, because they follow the linear pattern.

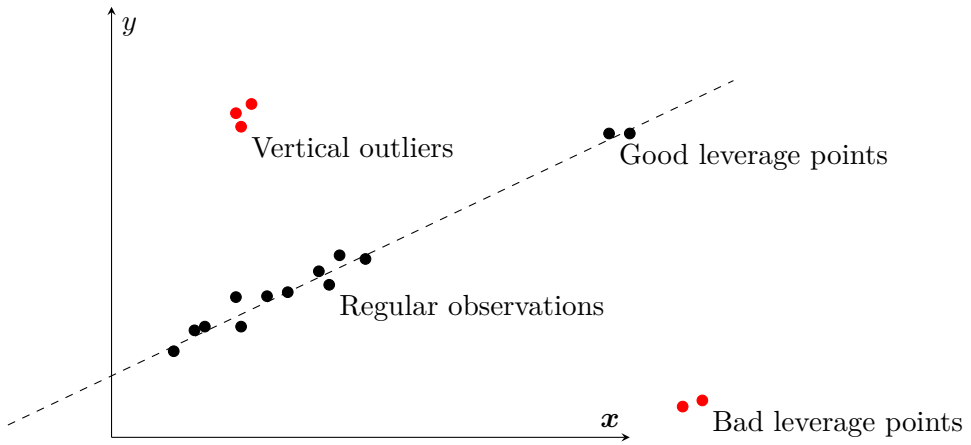


Figure 3.1: Different types of outliers.

Moreover, the data set can contain multiple observations that satisfies linear regression model but with different regression coefficients. That means such data set can contain data from multiple different models.

To generate the vertical outliers we only need to modify step 3 of Algorithm 41. We have multiple options:

- Generate ε_i from $\mathcal{N}(\mu, \sigma^2)$ but use different parameter μ and σ^2 than which were used for generating regular observations.
- Generate ε_i from some heavy tailed or asymmetrical distribution like Log-normal or exponential distribution.
- Combine both above so that we randomly choose distribution and randomly generate parameters for such distribution.

The last option is the most versatile so we use this option.

Because we generate $\boldsymbol{x_i}$ from the normal distribution, we can generate leverage points just by changing parameter μ of this distribution. If we consequently generate ε_i from the same distribution with same parameters as for the regular observations, we obtain good leverage points. On the other hand if we generate ε_i as described above, we obtain bad leverage points.

If we want to generate outliers that correspond to the different model we can just set the regression coefficients $\boldsymbol{w}$ differently. It is also possible to use different parameters of normal distribution for generating $\boldsymbol{x_i}$ and parameters for generating ε_i. Theoretically we are able to introduce outliers even into this model, but when this model is "outlier" by itself relative to the original model, it is not needed. By this approach we would be able to generate the observations from arbitrary number of differ models, but for the sake of the simplicity we introduce only one different model.

3.2 Data sets

We have implemented ideas from previous section to the data set generator with following parameters:

- n and p for setting number of the generated observations and dimension of the explanatory variables
- *outlier ratio* for setting proportion of the outliers in the data set. This include vertical outliers, bad leverage points and also outliers from the second model.
- *leverage ratio* proportion of the explanatory variables that are generated as leverage points.
- $\mu_{\boldsymbol{x}}, \sigma^2_{\boldsymbol{x}}$ parameters of the normal distribution for generating non outlying $\boldsymbol{x_i}$
- $\mu_{\boldsymbol{x_o}}, \sigma^2_{\boldsymbol{x_o}}$ parameters of the normal distribution for generating outlying $\boldsymbol{x_i}$ (leverage points)
- $\mu_\varepsilon, \sigma^2_\varepsilon$ parameters of the normal distribution for generating non outlying errors ε_i
- $\mu_{\varepsilon_o}, \sigma^2_{\varepsilon_o}$ parameters of the distribution for generating outlying errors ε_i.
- $distrib_{\varepsilon_o}$ distribution from which outlying errors are generated — options are normal distribution, log-normal distribution and exponential distribution (when exponential distribution is chosen, then only $\sigma^2_{\varepsilon_o}$ parameter is used)
- $2m$ *ratio* proportion from the outliers which are generated from the second model

- $\mu_{\boldsymbol{x_{M2}}}, \sigma^2_{\boldsymbol{x_{M2}}}$ and $\mu_{\varepsilon_{M2}}, \sigma^2_{\varepsilon_{M2}}$ parameters for normal distributions for generating $\boldsymbol{x_i}$ and ε_i respectively from the second model

We used this generator to generate three data sets $D1$, $D2$ and $D3$ which differ by the types of the outliers they contain:

- $D1$ contains outliers which are not from the second model: vertical outliers, bad leverage points and good leverage points ($2m\ ratio = 0$)
- $D2$ contains only outliers from the second model ($2m\ ratio = 1$)
- $D3$ contain outliers of all described types. ($2m\ ratio = 0.4$)

All three data sets are set to contain 20% leverage points (thus *leverage ratio* = 0.2) and non outlying $\boldsymbol{x_i}$ are generated from $\mathcal{N}(0, 10)$ (thus $\mu_{\boldsymbol{x}} = 0, \sigma^2_{\boldsymbol{x}} = 10$). Other parameters are independently randomly generated from uniform distribution so that:

- $\mu_{\boldsymbol{x_o}} \sim \mathcal{U}(20,\ 60), \sigma^2_{\boldsymbol{x_o}} \sim \mathcal{U}(10,\ 20)$
- $\mu_{\varepsilon} \sim \mathcal{U}(0,\ 10), \sigma^2_{\varepsilon} \sim \mathcal{U}(1,\ 5)$
- $\mu_{\varepsilon_o} \sim \mathcal{U}(-50,\ 50), \sigma^2_{\varepsilon_o} \sim \mathcal{U}(50,\ 200)$
- $\mu_{\boldsymbol{x_{M2}}} \sim \mathcal{U}(-30,\ 30), \sigma^2_{\boldsymbol{x_{M2}}} \sim \mathcal{U}(10,\ 20)$
- $\mu_{\varepsilon_{M2}} \sim \mathcal{U}(-10,\ 10), \sigma^2_{\varepsilon_{M2}} \sim \mathcal{U}(1,\ 5)$
- $distrib_{\varepsilon_o}$ is uniformly randomly set to normal, log-normal or exponential distribution

Finally parameters n, p and *outlier ratio* are set depending on the experiment.

3.3 Implementation of the algorithms

We have implemented all described algorithms, moreover because algorithms for computing the feasible solution could be implemented both by calculating inversion and by calculating QR decomposition, we have implemented both version of those algorithms. Here is the list of all the implemented algorithm with their acronyms we use for labeling them:

FAST-LTS from Section 2.2 with all described improvements

FSA-INV from Section 2.3

FSA-QR FSA using theory from Section 2.4

MOEA-INV from Section 2.4 it is the improved version of OEA

MOEA-QR MOEA using theory from Section 2.4.4

MMEA-INV from Section 2.4.3

MMEA-QR MMEA using theory from Section 2.4.4

BAB from Section 2.6

BSA is the implementation of improved BAB (BSABAB) from Section 2.7

FAST-LTS-MMEA-INV combination of algorithms from Section 2.5

FAST-LTS-MOEA-INV another combination of algorithm from Section 2.5

MOEA-QR-BAB BAB with sorting speedup as described in Section 2.6

MOEA-QR-BSA BSABAB using our idea from Section 2.7

P-BSA probabilistic BSA using our idea from Section 2.7

These algorithms were first implemented in Python using the NumPy package [25]. The performance was low, so we have implemented all of the algorithms in C++ using the Eigen library [26] for matrix manipulation. On the other hand it is very popular today to use Python for data processing and manipulation; for that reason we have used pybind11 library [27], that exposes C++ types for Python and vice versa and written Python wrappers around the C++ implementation.

Moreover pybind11 allows to bind Eigen types directly to the NumPy types (because both libraries are LAPACK compatible), so that it is possible to share pointers to the matrices between Eigen and NumPy so that the data does not have to be copied when transferring between Python an C++.

That means that the data generator, all tests and experiments are implemented in Python; the C++ code is called only within the Python wrappers. It is also appropriate to mention that interface of the algorithms was created with regards to the popular scikit-learn package [28]; the interface is almost identical, so all of the classes implementing the algorithms can be used in the same manner as classes from scikit-learn linear regression module.

3.4 Results

In two previous sections we have described our experimental setup. Here we report results of our experiments.

In the first experiment we run multiple simulations where we compare speed and accuracy of the algorithm finding subsets satisfying strong necessary condition e.g. h-element feasible subsets. For each combination parameters n, p and *outlier ratio* we generate we generate data sets $D1$, $D2$ and $D3$ 100 times (each time new data sets are generated) and run all algorithms on those

data sets. Value of h is conservatively chosen so that $h = [(n/2] + [(p+1)/2]$. For all runs we use the intercept, so value p represents dimension of $\boldsymbol{x}$ including intercept. All algorithm are set to run at most for 50 steps and each of them is starting only from 1 randomly selected h-element subset. The results are given in table 3.1 were average CPU time from 100 runs for each algorithm. We also measure cosine similarity of a given solution compared to the OLS solution on the subset which does not contain outliers. For $n > 500$ cells for algorithm FSA-I and FSA-QR are empty. That is because times of the run of these algorithms were too slow and it would take weeks to finish all simulations.

		algorithm	FAST-LTS		FSA-I		FSA-QR		MMEA-I		MMEA-QR		MOEA-I		MOEA-QR	
n	p	out	time	cos	time	cos	time	cos	time	cos	time	cos	time	cos	time	cos
20	2	0.10	0.004614	0.988359	0.000202	0.982532	0.006610	0.990883	0.000071	0.982120	0.000147	0.989808	0.000364	0.817895	0.000187	0.987816
		0.30	0.004291	0.985132	0.000213	0.986326	0.006309	0.963036	0.000068	0.814509	0.000152	0.993869	0.000370	0.656299	0.000202	0.972483
		0.45	0.004183	0.964252	0.000209	0.847824	0.004229	0.946639	0.000067	0.634588	0.000145	0.916261	0.000276	0.609704	0.000192	0.862572
100	3	0.10	0.010263	0.997496	0.019629	0.997817	0.178909	0.998082	0.000839	0.981119	0.002535	0.997572	0.002137	0.911982	0.006485	0.997943
		0.30	0.009968	0.998368	0.020972	0.998185	0.185812	0.998594	0.000854	0.924159	0.002704	0.996901	0.001721	0.877451	0.007001	0.998725
		0.45	0.009569	0.998725	0.020623	0.998749	0.150607	0.998839	0.000860	0.936188	0.002638	0.998588	0.001674	0.890495	0.006928	0.997922
	5	0.10	0.012194	0.996578	0.021666	0.996789	0.265649	0.997602	0.000884	0.990368	0.002995	0.997628	0.002414	0.885812	0.007262	0.997378
		0.30	0.012523	0.996829	0.024384	0.997198	0.286329	0.997886	0.000912	0.923997	0.003313	0.997691	0.002860	0.852555	0.007919	0.997339
		0.45	0.012282	0.997724	0.024928	0.996658	0.220120	0.994148	0.000933	0.891392	0.003232	0.997575	0.001798	0.865642	0.007690	0.993144
500	2	0.10	0.130725	0.999446	1.708373	0.999614	6.854093	0.999600	0.076088	0.999035	0.172415	0.999533	0.151864	0.998190	0.496092	0.999639
		0.30	0.134994	0.999591	1.661876	0.999801	6.661488	0.999791	0.075539	0.997468	0.173445	0.999742	0.157311	0.997426	0.488501	0.999702
		0.45	0.119165	0.999817	1.786019	0.999110	7.326588	0.999472	0.074628	0.998243	0.165817	0.999305	0.141019	0.998003	0.503233	0.999351
	5	0.10	0.100103	0.999442	2.013579	0.999605	8.534774	0.999680	0.076842	0.996679	0.161578	0.999745	0.111078	0.996261	0.480725	0.999641
		0.30	0.094455	0.999282	2.062271	0.999604	8.700532	0.999516	0.071619	0.994945	0.162314	0.999543	0.104943	0.994869	0.485604	0.999472
		0.45	0.091336	0.999652	2.077072	0.999163	8.865941	0.999343	0.067733	0.993651	0.152552	0.999013	0.103857	0.994371	0.484279	0.999033

Table 3.1: Table to test captions and labels

We can see that

Conclusion

To date, multiple exact and probabilistic algorithms for calculating LTS estimate has been proposed. Those algorithms have been proposed across the last few decades. Most recently proposed algorithms are just a few years old. That means that research in the filed of LTS algorithms is still prevalent.

Although to date exact finite algorithm has polynomial time, we showed that currently used probabilistic algorithm provide sufficiently fast solutions which, even though are not exact, are good enough. Though it is proven that the exact solution cannot be obtained faster than in polynomial time, we showed that currently used algorithms could be combined to obtain better results.

Algorithms for calculating the LTS estimate is an exciting topic for further research. One of the ways is the research of combining exact algorithms with probabilistic ones. As our experimental results prompt, we would possibly propose probabilistic algorithms which provide even better solutions at the same time.

Another way would be to use algorithms for computing LTS estimate on different robust statistic methods because some of them are very similar to LTS problem. Just put there are still a lot of future works on least trimmed squares, and we are planning to research this in the future.

Bibliography

[1] McCullagh, P. *Generalized linear models*. Routledge, 2018.

[2] Rousseeuw, P.; C. van Zomeren, B. Unmasking Multivariate Outliers and Leverage Points. *Journal of The American Statistical Association - J AMER STATIST ASSN*, volume 85, 06 1990: pp. 633–639, doi:10.1080/01621459.1990.10474920.

[3] Hampel, F. R.; Ronchetti, E. M.; et al. *Robust statistics*. Wiley Online Library, 1986.

[4] Massart, D. L.; Kaufman, L.; et al. Least median of squares: a robust method for outlier and model error detection in regression and calibration. *Analytica Chimica Acta*, volume 187, 1986: pp. 171–179.

[5] Agulló, J. New algorithms for computing the least trimmed squares regression estimator. *Computational Statistics & Data Analysis*, volume 36, no. 4, 2001: pp. 425–439.

[6] Bernholt, T. Robust estimators are hard to compute. Technical report, Technical Report/Universität Dortmund, SFB 475 Komplexitätsreduktion, 2006.

[7] Klouda, K. *Studium senzitivity odhadu metodou nejmenších usekaných čtverců*. Bachelor's thesis, Technical University of Prague, Faculty of Nuclear Sciences and Physical Engineering, Department of Mathematics, 2006.

[8] Hawkins, D. M. The feasible solution algorithm for least trimmed squares regression. *Computational statistics & data analysis*, volume 17, no. 2, 1994: pp. 185–196.

[9] Bai, E.-W. A random least-trimmed-squares identification algorithm. *Automatica*, volume 39, no. 9, 2003: pp. 1651–1659.

[10] Rousseeuw, P. J.; Leroy, A. M. *Robust regression and outlier detection*. John wiley & sons, 1987.

[11] Hawkins, D. M.; Olive, D. J. Improved feasible solution algorithms for high breakdown estimation. *Computational statistics & data analysis*, volume 30, no. 1, 1999: pp. 1–11.

[12] Strassen, V. Gaussian elimination is not optimal. *Numerische mathematik*, volume 13, no. 4, 1969: pp. 354–356.

[13] Coppersmith, D.; Winograd, S. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, volume 9, no. 3, 1990: pp. 251–280.

[14] Ballard, G.; Benson, A. R.; et al. Improving the numerical stability of fast matrix multiplication. *arXiv preprint arXiv:1507.00687*, 2015.

[15] Anderson, E.; Bai, Z.; et al. *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third edition, 1999, ISBN 0-89871-447-8 (paperback).

[16] Krishnamoorthy, A.; Menon, D. Matrix inversion using Cholesky decomposition. In *2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, IEEE, 2013, pp. 70–72.

[17] Businger, P.; Golub, G. H. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, volume 7, no. 3, 1965: pp. 269–276.

[18] Hammarling, S.; Lucas, C. Updating the QR factorization and the least squares problem. 2008.

[19] Rousseeuw, P. J.; Driessen, K. V. An Algorithm for Positive-Breakdown Regression Based on Concentration Steps. In *Data Analysis: Scientific Modeling and Practical Application*, edited by M. S. W. Gaul, O. Opitz, Springer-Verlag Berlin Heidelberg, 2000, pp. 335–346.

[20] Hoare, C. A. Algorithm 65: find. *Communications of the ACM*, volume 4, no. 7, 1961: pp. 321–322.

[21] Atkinson, A. C.; Weisberg, S. Simulated annealing for the detection of multiple outliers using least squares and least median of squares fitting. *Institute for Mathematics and Its Applications*, volume 33, 1991: p. 7.

[22] Bartlett, M. S. An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, volume 22, no. 1, 1951: pp. 107–111.

[23] Hofmann, M.; Kontoghiorghes, E. J. Matrix strategies for computing the least trimmed squares estimation of the general linear and sur models. *Computational Statistics & Data Analysis*, volume 54, no. 12, 2010: pp. 3392–3403.

[24] Klouda, K. An exact polynomial time algorithm for computing the least trimmed squares estimate. *Computational Statistics & Data Analysis*, volume 84, 2015: pp. 27–40.

[25] Oliphant, T. NumPy: A guide to NumPy. USA: Trelgol Publishing, 2006– , [Online; accessed ¡today¿]. Available from: `http://www.numpy.org/`

[26] Guennebaud, G.; Jacob, B.; et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[27] Jakob, W.; Rhinelander, J.; et al. pybind11 – Seamless operability between C++11 and Python. 2017, https://github.com/pybind/pybind11.

[28] Pedregosa, F.; Varoquaux, G.; et al. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, volume 12, 2011: pp. 2825–2830.

Appendix A

Data sets

GUI Graphical user interface

XML Extensible markup language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format