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 16, 2019

Acknowledgements

I wish to express my sincere thanks to Ing. Karel Klouda, Ph.D., chair of the Department of Applied Mathematics, for providing me with all the facilities and encouragement necessary in finishing this work.

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 16, 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 čvterců je robustní verzí známé metdoy nejmenších čvterců, jedné 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 čvterce, 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, 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

Citation of this thesis vi

Introduction **1**

1 Least trimmed squares **3**
1.1 Linear regression model 3
1.1.1 Prediction with the linear regression model 4
1.2 Ordinary least squares 5
1.2.1 Properties of the OLS estimate 6
1.3 Robust statistics 7
1.3.1 Outliers 7
1.3.2 Measuring robustness 8
1.4 Least trimmed squares 8
1.4.1 Discrete objective function 9

2 Algorithms **13**
2.0.1 First attempts 13
2.0.2 Strong and weak necessary conditions 14
2.1 Computing OLS 15
2.1.1 Computation using a matrix inversion 15
2.1.2 Computation using the Cholesky decomposition 16
2.1.3 Computation using the QR decomposition 17
2.2 FAST-LTS 22
2.2.1 C-step 22
2.2.2 Choosing an initial $\boldsymbol{m_1}$ subset 26
2.2.3 Speed-up of the algorithm 28
2.2.4 Putting all together 29
2.3 Feasible solution 29
2.4 OEA, MOEA, MMEA 33
2.4.1 Multiplicative formula 33

2.4.2 The OEA and its properties . . . 37
2.4.3 Minimum-maximum exchange algorithm . . . 39
2.4.4 Different method of computation of the inversion . . . 40
2.5 Combined algorithm . . . 44
2.6 BAB algorithm . . . 45
2.6.1 Improvements . . . 47
2.7 BSA algorithm . . . 47
2.7.1 Domain of OF-LTS . . . 47
2.7.2 One-dimensional version of the algorithm . . . 49
2.7.3 Multidimensional BSA . . . 50
2.7.4 Speed ups and modifications . . . 51

3 Experiments **53**
3.1 Data set generator . . . 53
3.1.1 Generating outliers . . . 54
3.2 Data sets . . . 55
3.3 Implementation of the algorithms . . . 56
3.4 Results . . . 58
3.4.1 The strong necessary condition algorithms . . . 58
3.4.2 Algorithms for finding the exact solution . . . 58
3.4.3 FAST-LTS and combinations of the algorithms . . . 60
3.4.4 Random algorithm and RBSA . . . 61

Conclusion **63**

Bibliography **65**

A Results of the experiments **69**

B Contents of enclosed CD **83**

List of Figures

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. 9

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 $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_2})$). 24

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). 26

2.3 The tree consisting of all 3-element subsets for $n = 4$. The leaves with blue border color represents 3-element subsets of $\{1, 2, 3, 4\}$. . 46

3.1 Different types of outliers. 54

3.2 Similarity of the solutions given by the algorithms finding h-element subsets satisfying strong necessary condition compared to the OLS solution on the subset of the data set that does not contain outliers. On the left are two box plots for all algorithms. Since the visualization is influenced by the scale of MMEA-I and MOEA-I box plots, we provide also two graphs without these two algorithms on the right. 59

3.3 For various combinations of parameters n and p we calculated average multiplicative improvement of the CPU time for running FSA-QR-BAB and FSA-QR-BSA instead of BAB and BSA. 60

3.4 Cosine similarity and L^2 norm for multiple algorithms. Random algorithms are outperformed by the algorithms finding the weak and strong necessary conditions. 62

List of Tables

3.1 Average, minimum and maximum CPU times of computation time for BSA for $p = 2$ and various combinations of parameters n and *out* for each dataset. 60

3.2 Percentage of the h-elements subsets provided by FAST-LTS which did not satisfied strong necessary condition, and the MMEA-QR and MOEA-QR were able to improve them. 61

A.1 Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D1$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm. 70

A.2 Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D2$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm. 71

A.3 Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D3$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm. 72

A.4 Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D1$ for various configurations of the parameters n, p and *out*. 73

A.5 Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D2$ for various configurations of the parameters n, p and *out*. 74

A.6 Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D3$ for various configurations of the parameters n, p and *out*. 75

A.7 Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D1$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR. 76
A.8 Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D2$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR. 77
A.9 Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D3$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR. 78
A.10 Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D1$. 79
A.11 Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D2$. 80
A.12 Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D3$. 81

Introduction

Least trimmed squares (LTS) is one of the many modifications of the very well known method of ordinary least squares (OLS). Both these methods are tools of regression analysis, which is a group of the 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.

In order to OLS estimate produces reliable results, many strong assumptions about the data have to be fulfilled. It is assumed that data is generated in a 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.

Robust statistic tries to solve problems of classical statistics methods. Its methods are usually emulations of classical statistic methods but try to provide reliable estimates 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.

The idea of the LTS method is simple, but unlike the OLS method, the exact solution is known to be NP-hard, hence only suboptimal probabilistic algorithms are usually used in practice.

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 the properties of both methods and also describe the field 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 al-

gorithms for calculating LTS estimate. They consist of multiple 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. Last but not least, we also propose several improvements to those algorithms.

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 sets affected by various types of outliers. Next, we provide information about our implementation of all algorithms from chapter two. Finally, we present our results for specific data sets.

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

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

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

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

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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$ 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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$ to any desired value [5] thus

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

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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$ tends to zero. We can see that ordinary least squares estimator is 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

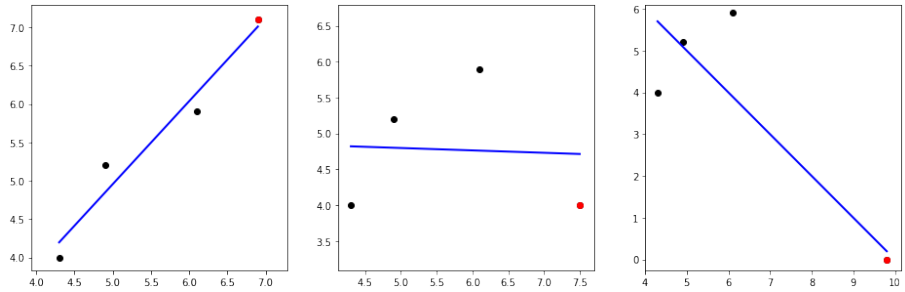

Figure 1.1: Change of the regression hyperplane given by coefficients estimated with OLS method when one of the four observations (highlighted with red color) starts to deviate from the linear pattern.

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

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

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

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

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 $\boldsymbol{M}_{LTS} = diag(\boldsymbol{m}^{(LTS)})$:

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

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, as described in [7],

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

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

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

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

Substituting $\boldsymbol{w}$ with the OLS estimate as in (1.25) we get the discrete objective function with domain $\boldsymbol{m} \in Q^{(n,h)}$

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

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 $r_i^2(\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{MX},\boldsymbol{My})})$ for all trimmed observation is greater or equal to the greatest non-trimmed squared residuum $r_j^2(\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{MX},\boldsymbol{My})})$.

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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{MX},\boldsymbol{My})}$ 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

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

Thus

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

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

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

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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$. 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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})} = \boldsymbol{CB}$ 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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$ 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 $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{X},\boldsymbol{y})}$.

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. Let us note that this theory is extensively used in Section 2.4.4 where we also show how to update QR decomposition when adding or deleting a from the matrix $\boldsymbol{X}$.

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

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

□

Based on the previous theorem, using some h-element subset $\boldsymbol{m_{old}}$ with corresponding $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{M}_{old}\boldsymbol{X},\boldsymbol{M}_{old}\boldsymbol{y})}$ we can construct $\boldsymbol{m_{new}}$ with corresponding $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{M}_{new}\boldsymbol{X},\boldsymbol{M}_{new}\boldsymbol{y})}$ 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 in Figure 2.1 where we start with the 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.

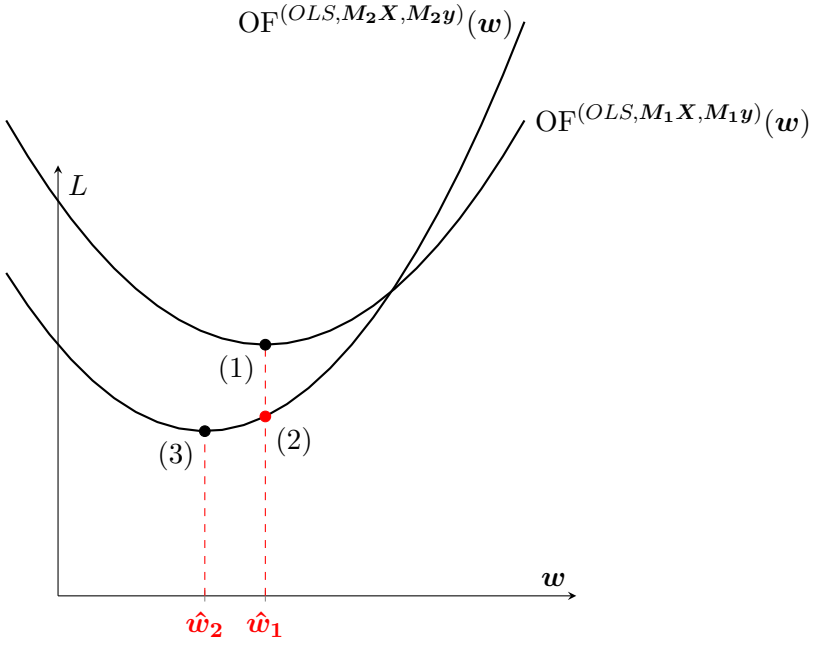

Figure 2.1: Illustration of the C-step algorithm. (1) represents the value of $\mathrm{OF}^{(OLS,\boldsymbol{M_1X},\boldsymbol{M_1y})}(\hat{\boldsymbol{w}}_{\boldsymbol{1}})$ (which is equal to the value $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_1})$). (2) represents the value of $\mathrm{OF}^{(OLS,\boldsymbol{M_2X},\boldsymbol{M_2y})}(\hat{\boldsym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

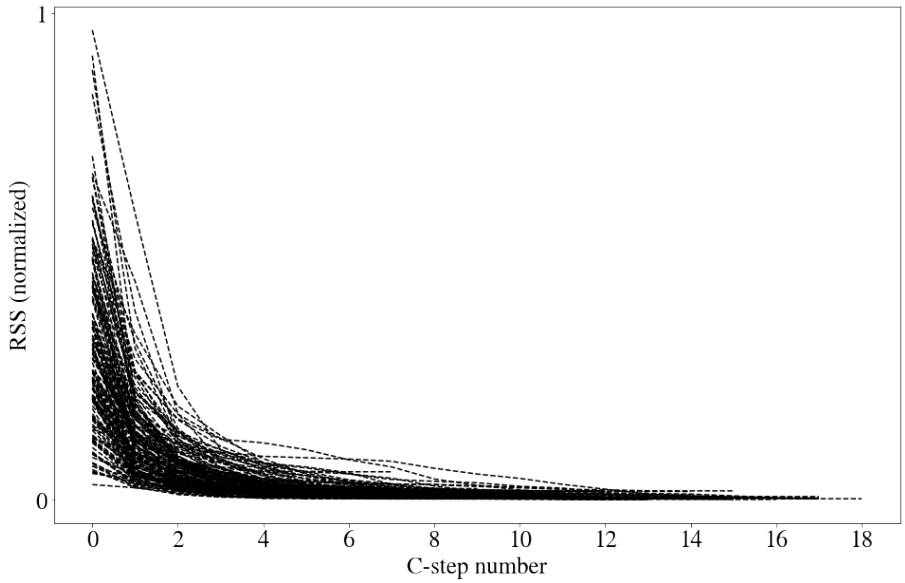

Figure 2.2: Value of the residual sum of squares (normalized) based on the number of the step of C-step algorithm for 100 different starting subsets. Dataset D3 was used with configuration $n = 500, p = 20$ and 30% of the the outliers (see Section 3 for more details about the dataset).

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 $|\operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) - \operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})| \leq r$ instead of $\operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) = \operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})$.

2.3 Feasible solution

In this section we introduce the Feasible solution algorithm (FSA) 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 $\boldsymbol{m} \in Q^{(n,h)}$, let us denote $O_m = \{i \in \{1, 2, \ldots, n\}; w_i = 1\}$ and $Z_m = \{j \in \{1, 2, \ldots, n\}; w_j = 0\}$ the sets of indices of 0s and 1s in vector $\boldsymbol{m}$. We can think about it as indices of non-trimmed observations in the h-element subset and trimmed $n-h$ observations respectively. Let $\boldsymbol{m}^{(i,j)}$ be a vector which is constructed by swapping ith and jth element $\boldsymbol{m}$ where $i \in O_m$ and $j \in Z_m$. Such vector corresponds to the vector $\boldsymbol{m}_{\boldsymbol{swap}}$ from Definition 8.

With this in mind, put

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

to express a change of the LTS objective function by swapping one observation from the non-trimmed subset with another from the trimmed subset. To calculate this we can obviously first calculate the $\operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$, then the $\operatorname{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)})$ and finally subtract both results. Although it is an option, it it computationally exhaustive. So the question is whether there is an easier way of calculating $\Delta S_{i,j}^{(\boldsymbol{m})}$. The answer is positive and we describe it now.

Let $M = diag(\boldsymbol{m})$, $M^{(i,j)} = diag(\boldsymbol{m}^{(i,j)})$ and $\boldsymbol{Z_M} = (\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{X})$. For now let us assume that we also have computed $\boldsymbol{Z_M}^{-1}$ 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 $\boldsymbol{r}^{(\boldsymbol{m})} = \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{w}}$ be the vector of residuals and $d_{r,s} = \boldsymbol{x_r}{\boldsymbol{Z_M}}^{-1}\boldsymbol{x_s}$, 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 FSA. It is similar to the FAST-LTS algorithm in terms of iterative refinement of a h-element subset. 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 $i \in O_m$ and $j \in Z_m$. 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 the $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)})$ are greater 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 the third case, the strong necessary condition is not satisfied, and we can make the swap decreasing the objective function. 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 practice, the third option is the winner, because it gives parameter which can be used to affect number if iterations of the algorithm. On the other hand, none of the options improve the time complexity of the algorithm, so from now on let us assume that we use the case number two. So if there are some negative $S_{i,j}^{(\boldsymbol{m})}$, we choose the one with the least 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} to bound the number of swaps without finding an h-element subset satisfying the strong necessary condition. One step of this algorithm,

called 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: i_{swap}, j_{swap}, S

$S \leftarrow 0$;
$i_{swap} \leftarrow \emptyset$;
$j_{swap} \leftarrow \emptyset$;
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; we need co multiply vector $\in \mathbb{R}^p$ with matrix from $\mathbb{R}^{p\times p}$ and vector from $\mathbb{R}^p$ 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 main loops go through all pairs; thus it is $\mathcal{O}(n^2)$ and $d_{i,j}$, which can be calculated in $\mathcal{O}(p^2)$, must be calculated inside the loops each time.

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 sort of a local optimum, i.e., to a h-element subset satisfying the strong necessary condition. In [8] they refer to this set as to the *feasible set*. As a process is not guaranteed to find the global minimum, the algorithm needs to be run multiple times say, t times. The h-element subset corresponding to the solution with the smallest value of the objective function is chosen as the final solution.

The problem of finding the initial h-element subset $\boldsymbol{m}$ was already discussed when describing the FAST-LTS algorithm. More importantly, an h-

element subset satisfying the weak necessary condition do not need to satisfy the strong necessary condition, so passing such a h-element subset as input to this algorithm is another option which we discuss in Section 2.5. We now describe the FSA using a pseudocode; we assume that we have function `generate_intial_subset` that generates h-element subsets.

Algorithm 5: FSA

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

$\hat{\boldsymbol{w}}_{final} \leftarrow \emptyset$;
$\boldsymbol{m}_{final} \leftarrow \emptyset$;
$RSS_{min} \leftarrow \infty$;
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**
    $\boldsymbol{M} \leftarrow diag(\boldsymbol{m})$;
    $\boldsymbol{Z_M} = (\boldsymbol{X}^T \boldsymbol{M} \boldsymbol{X})$;
    ${\boldsymbol{Z_M}}^{-1} \leftarrow inversion\ of\ \boldsymbol{Z_M}$;
    $\hat{\boldsymbol{w}} \leftarrow \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(m)$;
    $\boldsymbol{r}^{(\boldsymbol{m})} \leftarrow \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{w}}$;
    $i, j, S_{i,j} = OSAA(\ {\boldsymbol{Z_M}}^{-1}, \boldsymbol{r}^{(\boldsymbol{m})}, O_m, Z_m, \boldsymbol{X})$;
    if $S_{i,j} \geq 0$ ***or*** $l \geq i_{max}$ **then**
      $RSS_{new} \leftarrow \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}$;
      if $RSS_{new} < RSS_{min}$ **then**
        $RSS_{min} \leftarrow RSS_{new}$;
        $\boldsymbol{m}_{final} \leftarrow \boldsymbol{m}$;
        $\hat{\boldsymbol{w}}_{final} \leftarrow \hat{\boldsymbol{w}}$;
      end
      break;
    end
    else
      $\boldsymbol{m} \leftarrow \boldsymbol{m}^{(i,j)}$;
      $l \leftarrow l + 1$;
    end
  end
end
return $\hat{\boldsymbol{w}}_{final}$, $\boldsymbol{m}_{final}$;

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

see that the FSA time complexity is dominated by the OSAA.

In this section, we have described the FSA algorithm. In the next section, we introduce a very similar algorithm having better numerical stability and performance.

2.4 OEA, MOEA, MMEA

In the FSA, we assumed that after each cycle of OSAA we need to recalculate the inversion of $\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{X}$ together with $\hat{\boldsymbol{w}}$. In this section, we introduce a different approach described in [5]. Moreover, these ideas lead not only to speeding up the FSA but also to construction of other algorithms.

In Section 2.3 we have introduced additive formula (2.39) which value is $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}^{(i,j)}) - \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$. Let us now try to obtain a similar formula but with focus on how the individual elements in our current algorithm changes, namely the inversion iof $\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{X}$ and $\hat{\boldsymbol{w}}$. We also split the process of swapping ith and jth elements into the insertion of jth and removal of ith element.

2.4.1 Multiplicative formula

Let us denote $\boldsymbol{Z} = \boldsymbol{X}^T\boldsymbol{X}$ and let $\boldsymbol{A} = (\boldsymbol{X}\boldsymbol{y})$ stand for the matrix $\boldsymbol{X}$ containing target variables 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 a symmetric square matrix from $\mathbb{R}^{p\times p}$, we also assume 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. The OLS estimate $\hat{\boldsymbol{w}}$ is then given by

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

and the $RSS(\hat{\boldsymbol{w}})$ by

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

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

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

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

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

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

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

and

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

Including the observation

Using the formulas above, let us express how the determinant $\det(\boldsymbol{Z})$ and the inverse of the $\boldsymbol{Z}^{-1}$ changes when an 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

$$
\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & x_{i1} \\ x_{21} & x_{22} & \dots & x_{2n} & x_{i2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \dots & x_{pn} & x_{ip} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \\ x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix} = \boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{x_i^T}\boldsymbol{x_i} = \boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i}, \tag{2.47}
$$

so the 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

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

and

$$\boldsymbol{u} = \boldsymbol{Z}^{-1}\boldsymbol{x_i^T}. \tag{2.51}$$

Then (2.49) can be written as

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

Now, using the same idea as in (2.47) the updated $\hat{\boldsymbol{w}}$, which we denote as $\overline{\hat{\boldsymbol{w}}}$, is

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

This can be simplified so that we get [1]

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

Last but not least, we want to express the updated $RSS(\hat{\boldsymbol{w}})$. This can be done easily from (2.43) and (2.48) as

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

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

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

We can see that $\gamma^+(\boldsymbol{c_i})$ measures how $RSS(\hat{\boldsymbol{w}})$ increases after we extend the data set with observation $\boldsymbol{c_i}$, thus $\gamma^+(\boldsymbol{c_i}) \geq 0$.

[1] In [5] is a typing error in this formula so that $\overline{\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} + (y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})b\boldsymbol{u}$ is used instead of $\overline{\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} - (y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})b\boldsymbol{u}$

Excluding the observation

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

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

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

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

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

Once again, it is convenient to denote

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

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

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

Using the same approach as before we can express the up-dated estimate denoted as $\overline{\overline{\hat{\boldsymbol{w}}}}$:

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

The updated $RSS(\overline{\overline{\hat{\boldsymbol{w}}}})$ can be expressed using (2.43) and (2.48) and (2.62) as

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

Putting

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

we get

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

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

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

and get

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

Finally, we can express the $RSS(\overline{\overline{\hat{\boldsymbol{w}}}})$ as

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

where

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

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 (2.39) but here $\rho(\boldsymbol{c_i},\boldsymbol{c_j})$ represents multiplicative increment. Moreover if $0 < \rho(\boldsymbol{c_i},\boldsymbol{c_j}) < 1$ then the swap leads to improvement in terms of the value of the objective function.

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 only to update it. The authors of [5] call this algorithm optimal exchange algorithm (OEA).

2.4.2 The OEA and its properties

We can apply the previous section to the FSA. The sets O_m and Z_m (see Section 2.3) contain indices of observations to be excluded and included within the swap.

In terms of the FSA, as described in Algorithm 5, we first need to change the OSAA. There are only minor tweaks. First, we need to pass one more argument $RSS(\hat{\boldsymbol{w}})$. Second, we want to find the 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 remain unchanged as well as the time complexity.

This algorithm returns $\rho(\boldsymbol{c_i},\boldsymbol{c_j})$, and indices $i_{swap} \in O_m$ and $j_{swap} \in Z_m$ of observations we want to swap. With those in hand, we can update the $RSS(\hat{\boldsymbol{w}})$ 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).

As we said, the time complexity of modified OSAA is $\mathcal{O}(n^2p^2)$. Thus there is no asymptotic improvement.

When an $RSS(\hat{\boldsymbol{w}})$ improvement is found i.e., when $0 < \rho(\boldsymbol{c_i}, \boldsymbol{c_j}) < 1$, then we update $RSS(\hat{\boldsymbol{w}})$. Next we update inversion by (2.52) which is $\mathcal{O}(4p^2)$ and by (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 $\mathcal{O}(p^2 + p)$.

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 $\mathcal{O}(p^2n)$ of updating those quantities in the FSA (see Observation 21).

Now it seems 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 $\rho(\boldsymbol{c_i}, \boldsymbol{c_j}) \geq 1$.

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 fraction with the numerator

$$(1+\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}+\frac{e_j^2}{RSS(\hat{\boldsymbol{w}})})(1-\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}-\frac{e_i^2}{RSS(\hat{\boldsymbol{w}})})+(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}+\frac{e_ie_j}{RSS(\hat{\boldsymbol{w}})})^2. \tag{2.71}$$

Since $(\frac{e_ie_j}{RSS(\hat{\boldsymbol{w}})})^2 \geq 0$, then whole numerator is greater or equal to

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

On the other hand, we can see that denominator is

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

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

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

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

$$1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}. \tag{2.75}$$

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

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

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. This quantity cannot be computed outside the for 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 than or equal to ρ_{min}, we can continue to the 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 $\rho_{min} := \rho(\boldsymbol{c_i}, \boldsymbol{c_j})$.

This means that in the double loop whose time complexity is $\mathcal{O}(n^2)$ we do not always need to calculate $\rho(\boldsymbol{c_i}, \boldsymbol{c_j})$ with time complexity $\mathcal{O}(n^2)$. This does not improve the asymptotic time complexity, but as we will see later, can improve the speed of the algorithm.

Finally, let us note that in [5] the 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 OEA. The main difference is the greediness of this algorithm: algorithm does not find the optimal swap, but rather first find the $\boldsymbol{c_j}$ whose inclusion increases the $RSS(\hat{\boldsymbol{w}})$ as less as possible. Next, it finds $\boldsymbol{c_j}$ such that excluding this observation lead to the maximum decrease of the $RSS(\hat{\boldsymbol{w}})$.

The minimum increase can be found by calculating $\gamma^+(\boldsymbol{c_i})$ using (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 $\hat{\boldsymbol{w}}$ to $\overline{\hat{\boldsymbol{w}}}$ using (2.54).

Then we can find maximum $\gamma^-(\boldsymbol{c_j})$ by (2.65) among $O_m \cap \{i\}$.

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

Observation 22. One step of the algorithm MMEA has time complexity $O(p^2 n)$. So the whole algorithm has time complexity $\mathcal{O}(tlp^2 n)$ where t and l are given parameters bounding the number of iterations.

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

2.4.4 Different method of computation of the inversion

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 directly is not practical due to low numerical stability. In practice, we usually use QR decomposition. In this section, we describe how we can modify the 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 deleting 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 the matrix $\boldsymbol{R}$ with the added row and $\overline{\boldsymbol{Q}} \in \mathbb{R}^{p+1,p+1}$ denotes the matrix created from $\boldsymbol{Q}$ by adding one row and one column with zeros and putting $\overline{\boldsymbol{Q}}_{n+1,n+1} = 1$.

To update $\overline{\boldsymbol{R}}$ to upper triangular matrix $\boldsymbol{R}^{(+)}$, we need to create additional orthogonal Givens matrices $\boldsymbol{Q_{e+1}} \dots \boldsymbol{Q_{e+p}}$ zeroing the last row of $\overline{\boldsymbol{R}}$. Then the upper triangular matrix $\boldsymbol{R}^{(+)}$ equals $\boldsymbol{Q_{e+p}} \dots \boldsymbol{Q_{e+2}}\boldsymbol{Q_{e+1}}\boldsymbol{R}$.

Updated matrix $\overline{\boldsymbol{Q}}$ denoted as $\boldsymbol{Q}^{(+)}$ is updated in the same manner: $\boldsymbol{Q}^{(+)} = \boldsymbol{Q}\boldsymbol{Q}_{\boldsymbol{e}}^{\boldsymbol{T}}\boldsymbol{Q}_{\boldsymbol{e+1}}^{\boldsymbol{T}}\dots\boldsymbol{Q}_{\boldsymbol{e+p}}^{\boldsymbol{T}}$. 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: $\boldsymbol{Q} \in \mathbb{R}^{n,n}, \boldsymbol{R} \in \mathbb{R}^{n,p}, \boldsymbol{x_j} \in \mathbb{R}^p, k$
Output: $\boldsymbol{Q}^{(+)} \in \mathbb{R}^{n+1,n+1}, \boldsymbol{R}^{(+)} \in \mathbb{R}^{n+1,p}$

$\boldsymbol{R}^{(+)} \leftarrow \boldsymbol{R}$ *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* $\boldsymbol{Q}(i, n+1) \in \mathbb{R}^{n+1,n+1}$;
  $\boldsymbol{R}^{(+)} \leftarrow \boldsymbol{Q}(i, n+1)\boldsymbol{R}^{(+)}$;
  $\boldsymbol{Q}^{(+)} \leftarrow \boldsymbol{Q}^{(+)}\boldsymbol{Q}^T(i, n+1)$;
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 23. Computing $\boldsymbol{R}^{(+)}$ is $\mathcal{O}(p^2)$ and computing $\boldsymbol{Q}^{(+)}$ is $\mathcal{O}(np)$.

Proof. We have to loop over p 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 two rows are affected. That means we can simulate this matrix multiplication only by iterating over those matrices and multiplying the corresponding elements by $\cos(\varphi)$ and $\sin(\varphi)$ adequately. In case of $\boldsymbol{R}^{(+)}$ we are iterating p times over p 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 24. This process can also be done in the case when 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 $\boldsymbol{Q_1} \in \mathbb{R}^{p,p}$ thus then updating $\boldsymbol{Q_1}$ to $\boldsymbol{Q_1}^{(+)}$ is only $\mathcal{O}(p^2)$.

Note 25. 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 the following trick [18]. First, we move such row as the first row of the matrix $\boldsymbol{A}$ by creating 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. Hence we only need to introduce zeros in the first row $\boldsymbol{q_1}$ (except for q_{11}) of the matrix $\boldsymbol{Q}$. We can do this by $n-1$ matrices $\boldsymbol{Q}(i,j) \in \mathbb{R}^{n,n}$ 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 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

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

and

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

Observation 26. The time complexity of QR delete is $\mathcal{O}(n^2)$.

Proof. We need to create $n-1$ Givens matrices and with each multiply $\boldsymbol{Q^{(-)}}$ and $\boldsymbol{R^{(-)}}$. As we stated in Observation 23, this can be done in $\mathcal{O}(n)$ for each matrix. So together we get $\mathcal{O}(n^2)$. □

Note 27. In case of the 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 $\boldsymbol{x_i}\boldsymbol{Z^{-1}}\boldsymbol{x_i^T}$, $\boldsymbol{x_j}\boldsymbol{Z^{-1}}\boldsymbol{x_j^T}$ and $\boldsymbol{x_i}\boldsymbol{Z^{-1}}\boldsymbol{x_j^T}$. But this can also be done without inversion, only using the:

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

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

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

The same can be done with $\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$. Last but not least we need to solve $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$. We have

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

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

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

where $\boldsymbol{u}$ can be obtained by solving the upper triangular linear system

$$\boldsymbol{R}\boldsymbol{u} = \boldsymbol{v}, \tag{2.88}$$

where $\boldsymbol{v}$ is solution of (2.85).

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

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

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

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 have describe algorithms for updating the QR decomposition in Section 2.4.4.

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

Note 28. Because the time complexity of the QR delete is $\mathcal{O}(n^2)$, it is to be considered if instead of recycling QR decomposition is not worth it to recalculate it from scratch which takes $\mathcal{O}(p^2n)$ (see (2.20)).

Finally let us consider the matrix $\tilde{\boldsymbol{Z}}$ from (2.40) which we used for derivation of our equations. Employing the Observation 2.19 we get for $\boldsymbol{A} = (\boldsymbol{X}, \boldsymbol{y})$

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

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 $\boldsymbol{R}$ is the 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 the QR factorization of $\boldsymbol{X}$. Due to this fact $\hat{\boldsymbol{w}}$ is the solution of upper triangular linear system

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

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

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

Remark 29. We can see that all the quantities we use in the algorithm can be obtained from matrix $\tilde{\boldsymbol{R}}$.

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

2.5 Combined algorithm

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

THe algorithm FAST-LTS outputs h-element subset satisfying the weak necessary condition. One step of this algorithm has time complexity $\mathcal{O}(p^2 n)$ (see Observation 16). The number of iterations 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 $\mathcal{O}(p^2 n^2)$ (we now do not take into account greedy version MMEA, because we have no proof that it finds h-element subset satisfying the strong necessary condition).

We can use this in our favor so that we can find an 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 of how to proceed: z

1. Run the FAST-LTS algorithm and use its output as the h-element subset input to the MOEA.

2. Get the output h-element subset of the FAST-LTS algorithm an then perform one step of the MOEA and use the result as the input to the FAST-LTS. Then repeat these 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 greedy MMEA which time complexity 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]. Let us note that very similar algorithm can also be found in [23].

Unlike previous algorithms, this one is exact, meaning that it is guaranteed to fit the exact LTS estimate. As we discussed in Section 1.4.1, there is a exhaustive exact algorithm calculating the 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 $J_k = (j_1, j_2, \ldots, j_k) \subset \{1, 2, \ldots, n\}$. Then we can mark $\boldsymbol{X}_{\boldsymbol{J_k}}$ and $\boldsymbol{Y}_{\boldsymbol{J_k}}$ the matrices created from $\boldsymbol{X}$ and $\boldsymbol{Y}$ by removing all rows that are not indexed by J_k. We can see that the number of such subsets is given by $\binom{n}{k}$.

Let us now consider the 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-element subset indices. An example of such a tree we can be seen in Figure 2.3.

In the case of the exhaustive approach, we can then traverse the tree starting in the root using the left to right (LTR) preorder traversal, and in each node, we can calculate the OLS fit on $\boldsymbol{X}_{\boldsymbol{J_k}}$ and $\boldsymbol{Y}_{\boldsymbol{J_k}}$. This approach is exhaustive, so let us now describe how we can get better efficiency.

The main improvement is done by skipping parts of the tree (subtree pruning) according to the following. Moreover, at some point of the traversal we encounter nodes from which we cannot get into the depth h and hence

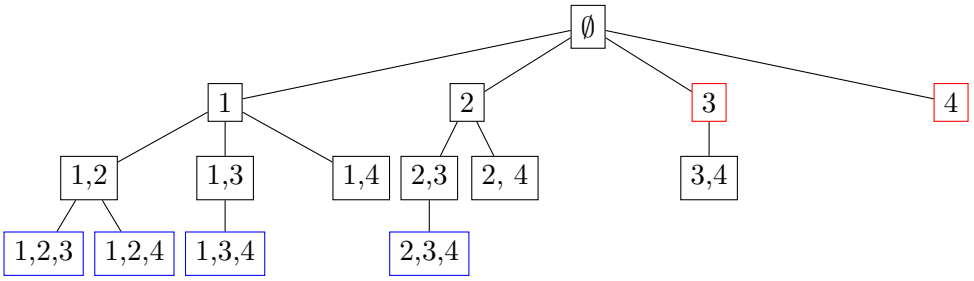

Figure 2.3: The tree consisting of all 3-element subsets for $n = 4$. The leaves with blue border color represents 3-element subsets of $\{1, 2, 3, 4\}$.

these nodes can also be skipped. In Figure 2.3 these nodes are highlighted with red border color.

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

We can describe the algorithm as follows:

1. Set $RSS^* = \inf$, $\boldsymbol{m}^* = \emptyset$, $\boldsymbol{w}^* = \emptyset$,
2. for each node in LTR preorder traversal calculate $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_k})$.
3. If $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_k}) > RSS^*$, skip all siblings of this node in the traversal.
4. If $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_k}) < RSS^*$ and $k = h$, then set $RSS^* = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_k})$, $\boldsymbol{m}^*$ so that $m_{k_i} = 1$ when $k_i \in J_k$ and $m_{k_i} = 0$ otherwise, and also set $\boldsymbol{w}^* = \hat{\boldsymbol{w}}^{(OLS, \boldsymbol{M}_k\boldsymbol{X}, \boldsymbol{M}_k\boldsymbol{y})}$.
5. When the traversal is finished, return $\boldsymbol{m}^*$ and $\hat{\boldsymbol{w}}^*$

Note that 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 held in the memory.

Remark 30. If we calculated $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_k})$, than we do not need to calculate $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m_{k+1}})$ from the scratch. In Section 2.4 we have described the way of 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 QR decomposition which is another option.

Moreover, we have described that if rows are not removed but only inserted, then the matrix $\boldsymbol{Q}$ do not have to be present and only the 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 so we update the decomposition only when inserting the rows to the matrix $\boldsymbol{X}$. If we use matrix $\tilde{\boldsymbol{Z}}$ and $\tilde{\boldsymbol{R}}$ we can (see Remark 29) 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 $\mathcal{O}(p^2)$.

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 for this LTS estimate in the decreasing order. That means that during the traversal first h-element subset we encounter represents the best known solution given by the approximative algorithm.

Another improvement can be made using the following *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 for one of those siblings,and if $RSS > RSS^*$, 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 the parent node J_k because all of his siblings have already been explored or can be trimmed.

2.7 BSA algorithm

In this section, we present another exact algorithm called BSA introduced in [24]. It uses a little different approach than the previous algorithms.

2.7.1 Domain of OF-LTS

We have introduced two versions of OF-LTS. First one is $\mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w})$ with the domain $\mathbb{R}^p$ and the second one is the discrete version $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ with the domain $Q^{(n,h)}$. 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 consider the non-discrete version and introduce some its features.

Definition 31. Let $Z \subset \mathbb{R}^p \times Q^{(n,h)}$ be an relation defined as

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

Z is not a mapping. To show this, we can take a simple example: assume that $\boldsymbol{w}$ is a vector of regression coefficients such that $r^2_{h:n}(\boldsymbol{w}) = r^2_{h+1:n}(\boldsymbol{w})$. Then for this $\boldsymbol{w}$ we have two different vectors $\boldsymbol{m}$ that are in the relation with it.

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

Let us now describe some properties that can help us to decide whether $\boldsymbol{w}$ is in $\mathcal{U}$ or in $\mathcal{H}$.

Lemma 32. It holds that $\boldsymbol{w} \in \mathcal{U}, \boldsymbol{w}$ if and only if $r^2_{h:n}(\boldsymbol{w}) < r^2_{h+1:n}(\boldsymbol{w})$

Proof. As shown in the example above, if $r_i^2(\boldsymbol{w}) = r^2_{h:n}(\boldsymbol{w}) = r^2_{h+1:n}(\boldsymbol{w}) = r_j^2(\boldsymbol{w}), i, j \in \{1, 2, \ldots, n\}$ are distinct, then $(\boldsymbol{w}, \boldsymbol{m_1}) \in Z$ and $(\boldsymbol{w}, \boldsymbol{m_2}) \in Z$ where $\boldsymbol{m_1}$ has ones at indices of h smallest residuals and $\boldsymbol{m_2}$ has ones at the same indices except for swapping ith one with jth zero. □

Corollary 33. It holds that

$$\mathcal{H} = \{\boldsymbol{w} \in \mathbb{R}^p | r^2_{h:n}(\boldsymbol{w}) = r^2_{h+1:n}(\boldsymbol{w})\}. \tag{2.95}$$

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

$$(y_i - \boldsymbol{x_i w})^2 = r_i^2(\boldsymbol{w}) = r^2_{h:n}(\boldsymbol{w}) = r^2_{h+1:n}(\boldsymbol{w}) = r_j^2(\boldsymbol{w}) = (y_j - \boldsymbol{x_j w})^2. \tag{2.96}$$

We can see that

$$(y_i - \boldsymbol{x_i w})^2 = (y_j - \boldsymbol{x_j w})^2 \iff y_i \pm y_j + (\boldsymbol{x_i} \pm \boldsymbol{x_j})\boldsymbol{w} = 0 \tag{2.97}$$

Assumption 34. 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 34 is fulfilled, then $y_i \pm y_j + (\boldsymbol{x_i} \pm \boldsymbol{x_j})\boldsymbol{w} = 0$ represents a hyperplane.

It is easy to show that the set $\mathcal{U}$ is open and Lebesgue measure of $\mathcal{H}$ is 0. Moreover, $\mathcal{H}$ splits $\mathbb{R}^p$ into finite number of k open disjoint subsets of $\mathcal{U}$.

Definition 35. Let us define the set of k sets $\mathcal{U}^{(set)} = \{U_i\}_{i=1}^k$ so that all U_i are open and connected sets, U_i and U_j are mutually disjoint such that $\cup_{i=1}^k U_i = \mathcal{U}$ and $\cup_{i=1}^k \partial U_i = \mathcal{H}$.

We define neighbor sets $U_i, U_j, i \neq j$ as the sets whose borders are not disjoint. We also define $M^{(min)}$ as the set of k vectors $\boldsymbol{m} \in Q^{(n,h)}$ by

$$M^{(min)} = \{\boldsymbol{m} | \boldsymbol{m} = Z(\boldsymbol{w}) \ for \ some \ \boldsymbol{w} \in U_i\}.$$

Note that we have $Z(\boldsymbol{w}) = Z(\boldsymbol{w'})$ for all $w, w' \in U_i$.

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

Theorem 36. If the matrix $\boldsymbol{X}$ has h-full rank, then for every local minima of the OF-LTS at point $\boldsymbol{w_0}$ satisfying the weak necessary condition, 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].

2.7.2 One-dimensional version of the algorithm

Based on the Theorem 36 we can see that

$$\min_{\boldsymbol{m} \in Q^{(n,h)}} \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}(\boldsymbol{w}) = \min_{\boldsymbol{m} \in M^{(min)}} \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}(\boldsymbol{w}), \tag{2.98}$$

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 $Q^{(n,h)}$. Hence minimizing objective function $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ (1.31) can be reformulated as

$$\min_{\boldsymbol{m} \in Q^{(n,h)}} \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) = \min_{m \in M^{(min)}} \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}). \tag{2.99}$$

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

For now, let us assume that we know all the elements of $\mathcal{H}$. Because for each $\boldsymbol{m} \in M^{(min)}$ there exists at least one $\boldsymbol{w} \in \mathcal{H}$ such that $(\boldsymbol{w}, \boldsymbol{m}) \in Z$, then for a given $\boldsymbol{w} \in \mathcal{H}$ we can find all $\boldsymbol{m}$ by the following algorithm.

Algorithm 37 (Find all $\boldsymbol{m}$ in relation Z with a given $\boldsymbol{w}$)**.**

1. Calculate all squared residuals $r_i^2(\boldsymbol{w})$ for $i \in \{1, \ldots, n\}$.
2. Sort the residuals so that $r_{i_k}^2(\boldsymbol{w}) = r_{k:n}^2(\boldsymbol{w})$.
3. If $r_{i_h}^2(\boldsymbol{w}) < r_{i_{h+1}}^2(\boldsymbol{w})$ there is a unique $\boldsymbol{m}$ in relation with $\boldsymbol{w}$, namely $\boldsymbol{m}$ where $m_i = 1 \Leftrightarrow r_i^2(\boldsymbol{w}) \leq r_{i_h}^2(\boldsymbol{w})$. Return this $\boldsymbol{m}$.
4. Find the greatest index l such that $r_{i_l}^2(\boldsymbol{w}) < r_{i_h}^2(\boldsymbol{w})$.
5. Find the greatest index t such that $r_{i_h}^2(\boldsymbol{w}) = r_{i_t}^2(\boldsymbol{w})$.
6. Create all vectors $\boldsymbol{m}$ by combining first l unique indices $i_1, \ldots, i_l$ with all combinations of $(h-l)$-element subsets of indices $i_{l+1}, \ldots i_t$. Number of this vector is given by $\binom{t-l}{h-l}$.

Finally we need to find all the elements of $\mathcal{H}$. As we know, for all the elements $\boldsymbol{w} \in \mathcal{H}$ it holds that $r_{h:n}^2(\boldsymbol{w}) = r_{h+1:n}^2(\boldsymbol{w})$. So, if we find all $\boldsymbol{w}$ that are solutions of such an equation, then we are done. The problem is that the residuals are sorted. To overcome this issue we define the following set of suitable candidates for the desired $\boldsymbol{w}$.

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 define the set H containing $\boldsymbol{w}$ satisfying this necessary condition, as

$$H = \{\boldsymbol{w} \in \mathbb{R}^p | r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w}), i \neq j\}. \tag{2.100}$$

Finding all elements of H requires solving $\binom{n}{2}$ equations of type $r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w})$. Moreover, because these equations are quadratic, we can have up to two solutions for each equation.

So the idea of the whole algorithm is to find all elements of H (by solving the quadratic equations), consequently finding which of them are elements of $\mathcal{H}$ (by ordering squared residuals and checking if $r_{h:n}^2(\boldsymbol{w}) = r_{h+1:n}^2(\boldsymbol{w})$) and finally for each $\boldsymbol{w} \in \mathcal{H}$ find subsets $\boldsymbol{m}$ that are in relation Z with it. All of those $\boldsymbol{m}$ vectors form the $M^{(min)}$ set. Finally we can use $\boldsymbol{m} \in M^{(min)}$ for minimizing $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m})$ in terms of (2.99). The author calls this algorithm Border Scanning Algorithm (BSA).

Algorithm 38 (BSA for the $p = 1$)**.**

1. Set $RSS_{min} = \infty, \boldsymbol{m}_{min} = \emptyset, \boldsymbol{w}_{min} = \emptyset$.
2. For each combination of two observations (x_i, y_i) and (x_j, y_j):
3. solve $r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w})$ for $\boldsymbol{w}$ and denote the two solutions as $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$.
4. For $\boldsymbol{w_1}$ and $\boldsymbol{w_2}$ (if $\boldsymbol{w_1}$ is equal to $\boldsymbol{w_2}$, then only for w_1):
5. calculate all squared residuals $r_i^2(\boldsymbol{w_j})$ for $i \in \{1, \ldots, n\}$.
6. Sort the residuals.
7. If $r_{i_h}^2(\boldsymbol{w_j}) = r_{i_{h+1}}^2(\boldsymbol{w_j})$ find all $\boldsymbol{m}$ in relation with $\boldsymbol{w_j}$ using Algorithm 37.
8. For each subset $\boldsymbol{m}$:
9. if $\mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}) < RSS_{min}$, set $RSS_{min} = \mathrm{OF}_{\mathrm{D}}^{\mathrm{LTS}}(\boldsymbol{m}), \boldsymbol{m}_{min} = \boldsymbol{m}$ and $\boldsymbol{w}_{min} = \boldsymbol{w_j}$.
10. Return $\boldsymbol{m}_{min}$ and $\boldsymbol{w}_{min}$

2.7.3 Multidimensional BSA

If we would like to scale the algorithm into the higher dimension $p > 1$, we have to face one major complication. The algorithm finds all elements of H and $\mathcal{H}$ but if $p > 1$, then $\mathcal{H}$ is union of hyperplanes and so contain an infinite number of points.

Therefore the author of BSA proposes to find a finite subset $\mathcal{H}_p$ of $\mathcal{H}$ that satisfies following condition

$$\forall \boldsymbol{m} \in M^{(min)} \exists \boldsymbol{w} \in \mathcal{H}_p : (\boldsymbol{w}, \boldsymbol{m}) \in Z \tag{2.101}$$

and also finite set H_p, so that $\mathcal{H}_p \subset H_p \subset H$. It is proven in [24] the set $\mathcal{H}_p$ can be defined as the set of solutions of p quadratic equations $r_i^2(\boldsymbol{w}) = r_j^2(\boldsymbol{w})$

$$\begin{aligned} r_{i_1}^2(\boldsymbol{w}) &= r_{i_2}^2(\boldsymbol{w}) \\ r_{i_2}^2(\boldsymbol{w}) &= r_{i_3}^2(\boldsymbol{w}) \\ \vdots \qquad & \qquad \vdots \\ r_{i_p}^2(\boldsymbol{w}) &= r_{i_{p+1}}^2(\boldsymbol{w}) \end{aligned} \tag{2.102}$$

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

Let $\circ \in \{+, -\}$ be either the operation $+$ or $-$. The system of quadratic equations (2.102) is equivalent to 2^p linear systems

$$\begin{aligned} (\boldsymbol{x}_{i_1} \circ_1 \boldsymbol{x}_{i_2})^T \boldsymbol{w} &= y_{i_1} \circ_1 y_{i_2} \\ \vdots \qquad & \qquad \vdots \\ (\boldsymbol{x}_{i_p} \circ_p \boldsymbol{x}_{i_{p+1}})^T \boldsymbol{w} &= y_{i_p} \circ_p y_{i_{p+1}}, \end{aligned} \tag{2.103}$$

where $\circ_i$ represents ether the addition or subtraction. The set $\mathcal{H}_p$ with the property (2.101) can be define as the set of elements of H_p such that $r_{i_1}^2(\boldsymbol{w}) = r_{h:n}^2(\boldsymbol{w}) = r_{h+1:n}^2(\boldsymbol{w})$.

The multidimensional version of the algorithm is indeed very similar to the one dimensional one. We basically just solve greater amount of bigger linear system to find all the elements of $\mathcal{H}_p$. In terms of Algorithm 38 the difference is in the step 2 where we choose combinations of $p+1$ observations instead of two. Consequently in step 3 we construct 2^p systems of p linear equations instead of only two. This means that the total number of vectors elements of $\mathcal{H}_p$ can be up to $\binom{n}{p+1}2^p$. Algorithm 37 works the same way for the multidimensional version of the algorithm. This algorithm can produce up to $\binom{p}{[p/2]}$ vectors $\boldsymbol{m}$ for each vector $\boldsymbol{w}$. That means in the worst case we have to calculate $\hat{\boldsymbol{w}}^{(OLS,\boldsymbol{MX},\boldsymbol{My})}$ up to $\binom{n}{p+1}\binom{p}{[p/2]}2^p$ times. We can already see that this algorithm is not suitable for higher dimensional observations.

2.7.4 Speed ups and modifications

Author proposed modification of the algorithm called BSABAB. This modification tries to utilize the branch and bound paradigm described in the Section 2.6. This can be done only at a smaller scale in step 4 of Algorithm 37. If the sum of l smallest unique squared residuals is larger than currently obtained RSS_{min} then we can skip up to $\binom{p}{[p/2]}$ calculations of the OLS.

This idea can be pushed further in the similar way as was described in the Section 2.6. We propose to use some approximative algorithm which solution

can be used to set the RSS_{min} in advance and let the algorithm cut as much as possible "branches".

Another possibility of modifying the algorithm is to instead of using all $(p+1)$-element subsets in the step 2 of the multidimensional version of the Algorithm 38 use only limited amount of randomly chosen $(p+1)$-element subsets. We refer to this modification as to the Random border scanning algorithm (RBSA).

Chapter 3

Experiments

In this chapter we introduce experiments and their results for our implementation of all algorithms described in the previous chapter. In order to test the performance of algorithms we have implemented data set generator providing artificial data sets with various properties.

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 39 (Generate clean data)**.**

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

As a result we obtain a data set stored in the matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$. Regression coefficients can be set to arbitrary values. All $\boldsymbol{x_i}$ should be generated independently but to avoid huge numbers we generate all $\boldsymbol{x_i}$ from some 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_{i1} should be equal to 1. Note that the same result can be obtained by generating data without intercept and with $\varepsilon_i \sim \mathcal{N}(\mu, \sigma^2)$ for some $\mu \in \mathbb{R}$. We can then extend matrix $\boldsymbol{X}$ by adding the 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 distinguish different types of the outliers: vertical outliers and two types of leverage points — good leverage points and bad leverage points. Those types of outliers are visualized in 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.

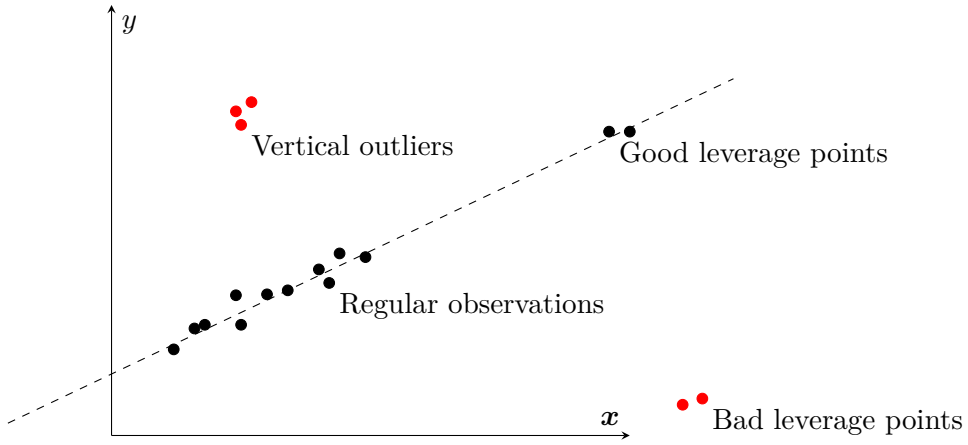

Figure 3.1: Different types of outliers.

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 39. We have multiple options:

- Generate ε_i from $\mathcal{N}(\mu, \sigma^2)$ but use different parameter μ and σ^2.
- 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 it in our data generator.

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 the 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 choose 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 an "outlier" by itself relative to the original model, it is not needed. By this approach are able to generate the observations from arbitrary number of different models, but for the sake of the simplicity we use only one different model in our data sets.

3.2 Data sets

We have implemented random data set generator as described in the previous section with the following parameters:

- n and p for setting the number of the generated observations and the 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* is proportion of the explanatory variables that are generated as leverage points,
- $\mu_{\boldsymbol{x}}, \sigma^2_{\boldsymbol{x}}$ are the parameters of the normal distribution for generating non outlying $\boldsymbol{x_i}$,
- $\mu_{\boldsymbol{x_o}}, \sigma^2_{\boldsymbol{x_o}}$ are the parameters of the normal distribution for generating outlying $\boldsymbol{x_i}$ (leverage points),
- $\mu_\varepsilon, \sigma^2_\varepsilon$ are the parameters of the normal distribution for generating non outlying errors ε_i,
- $\mu_{\varepsilon_o}, \sigma^2_{\varepsilon_o}$ are the parameters of the distribution for generating outlying errors ε_i,
- $distrib_{\varepsilon_o}$ is the 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),

- $2m\_ratio$ is the proportion from the outliers which are generated from the second model,
- $\mu_{\boldsymbol{x_{M2}}}, \sigma^2_{\boldsymbol{x_{M2}}}$ and $\mu_{\varepsilon_{M2}}, \sigma^2_{\varepsilon_{M2}}$ are the 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 ($leverage\_ratio = 0.2$) 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 separately for each particular experiment. Implemented data generator provides not only matrix $\boldsymbol{X}$ and vector $\boldsymbol{y}$ but also their subsets which does not contain outliers. This is useful, because it gives us the ability to compare appropriative solution to the original model.

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 (Section 2.2 with all described improvements),

FSA-I (Section 2.3),

FSA-QR (FSA using theory from Section 2.4),

MOEA-I (Section 2.4 it is the improved version of OEA),

MOEA-QR (MOEA using theory from Section 2.4.4),

MMEA-I (Section 2.4.3),

MMEA-QR (MMEA using theory from Section 2.4.4),

BAB (Section 2.6),

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

FAST-LTS-MMEA-I (combination of algorithms from Section 2.5),

FAST-LTS-MOEA-I (other combination of algorithm from Section 2.5),

FSA-QR-BAB (BAB with sorting speedup as described in Section 2.6),

FSA-QR-BSA (BSABAB using our idea from Section 2.7),

RBSA (probabilistic BSA using our idea from Section 2.7),

RANDOM (random resampling algorithm outlined in section Section 2.0.1).

These algorithms were first implemented in Python using the NumPy package [25]. To seed up the algorithms,we have implemented all the algorithms in C++ using the Eigen library [26] for matrix manipulation. Since it is very popular today to use Python for data processing and manipulation, 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 and hence the data does not have to be copied when transferring between Python an C++.

Therefore 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 the 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 the scikit-learn linear regression module.

3.4 Results

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

3.4.1 The strong necessary condition algorithms

Here we present the results of multiple simulations where we compared speed and accuracy of the algorithm finding subsets satisfying the strong necessary condition. For each combination of parameters n, p and *outlier_ratio* (*out*) we generated data sets $D1$, $D2$ and $D3$ 100 times (each time new data sets were generated) and run all algorithms on those data sets. Value of h was conservatively chosen so that $h = [(n/2] + [(p+1)/2]$. For all the runs we used the intercept, hence the value of p represents the dimension of $\boldsymbol{x}$ including intercept. All algorithms were set to run at most for 50 steps and each of them is starting from 1 randomly selected h-element subset. [2] The results are given in Appendix A in Tables A.1, A.2 and A.3 for data sets $D1$, $D2$ and $D3$, respectively. Beside CPU time, we also measured the cosine similarity and L^2 norm of the resulting estimate and the regression coefficients given by the original model which does not contain outliers. For $n > 500$, cells for the FSA-I and FSA-QR are empty, these algorithms were too slow and it would take weeks to finish all simulations. In Figure 3.2 are given box plots showing cosine similarity and L^2 norms of the results.

As expected, the algorithms using the QR decomposition provide slightly better results. Moreover, the best results are given by the FSA-QR. On the other hand, both FSA-I and FSA-QR are much slower because they does not use bounding condition as in the case of MOEA-I and MOEA-QR. We can also see that MMEA-QR provides very similar results to the MOEA-QR. So in the case of algorithms finding strong necessary condition we would recommend using the FSA-QR for small data sets and for large ones the MMEA-QR.

3.4.2 Algorithms for finding the exact solution

Here we provide results of the algorithms finding the exact solution. For the improved versions of the BAB and BSA algorithms, which are able to incorporate pre-computed results, we decided to use the estimate given by the FSA-QR. This algorithm, as observed in previous section, provides the best results. FSA-QR is quite slow compared to the MOEA and MMEA variations, but because the exact algorithms have much higher time complexity (thus can be used only on very small data sets), this slowing down is insignificant with respect to the total time complexity.

[2] This was done primarily because the computation was exhaustive due to the large number of parameter combinations. In experiments we describe later, number of the starting subsets is higher.

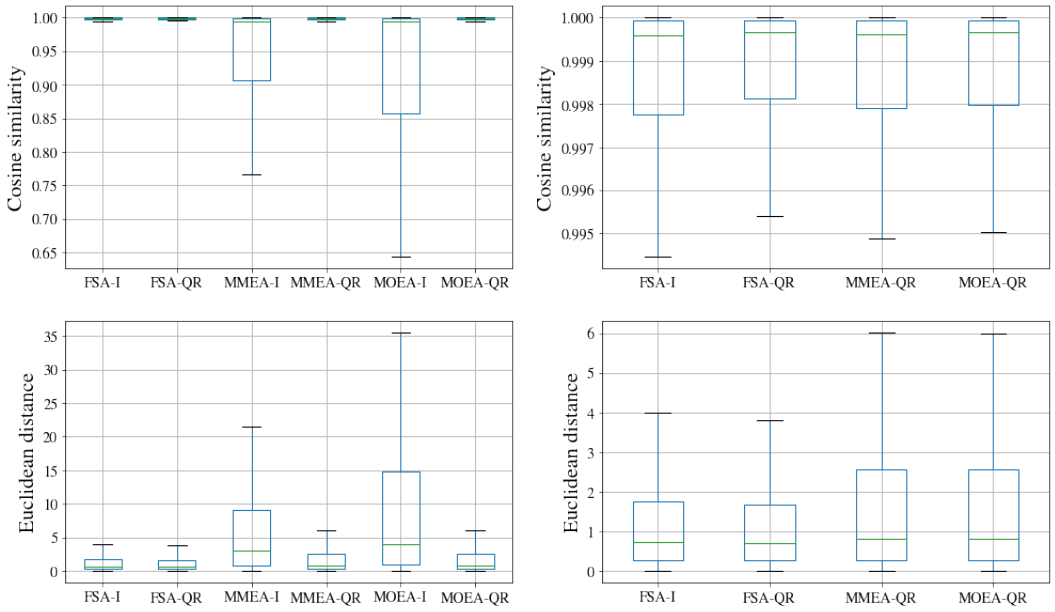

Figure 3.2: Similarity of the solutions given by the algorithms finding h-element subsets satisfying strong necessary condition compared to the OLS solution on the subset of the data set that does not contain outliers. On the left are two box plots for all algorithms. Since the visualization is influenced by the scale of MMEA-I and MOEA-I box plots, we provide also two graphs without these two algorithms on the right.

The setup for the simulations was the same as in the previous section, but we only compare speed this time because all of the algorithms provide the exact solution.

The results are given in Appendix A in Tables A.4, A.5 and A.6 for data sets $D1$, $D2$ and $D3$ respectively.

Interesting observation is given by the Figure 3.3 depicting where is the average time of the calculation improved by the FSA-QR-BAB instead of the BAB and the FSA-QR-BSA instead of the BSA. Whereas FSA-QR-BAB improves the time of the computation significantly, the FSA-QR-BSA does not leads to any time improvement at all.

We can observe from the results that BSA (as well as FSA-QR-BSA) is, as expected, sensitive to increasing the parameter p. On the other hand, when p stays low, BSA outperforms both BAB and FSA-QR-BAB. We have ran 10 simulations for higher values of n for each dataset $D1$, $D2$ and $D3$ for algorithm BSA, while preserving value of $p = 2$. In this simulation we have not used intercept. Resulting average, minimum and maximum CPU times

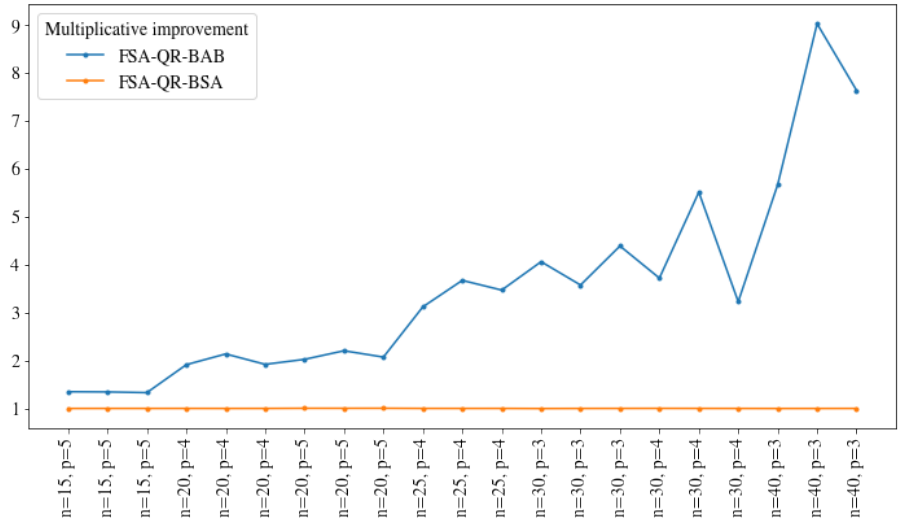

Figure 3.3: For various combinations of parameters n and p we calculated average multiplicative improvement of the CPU time for running FSA-QR-BAB and FSA-QR-BSA instead of BAB and BSA.

are given in Table 3.1. For the data sets $D2$ and $D3$ the simulation was not performed for $n = 400$, so the respective cells are empty.

		out	$D1$			$D2$			$D3$		
n	*out*	h	avg	min	max	avg	min	max	avg	min	max
100	0.10	51	5.499	5.213	6.860	5.206	5.150	5.273	5.215	5.166	5.342
	0.30	51	5.598	5.125	6.852	5.345	5.129	6.707	5.198	5.142	5.309
	0.45	51	5.978	5.161	6.902	5.347	5.099	6.724	5.425	5.164	6.989
200	0.10	101	81.172	80.042	81.582	81.280	80.816	81.575	82.678	81.153	93.849
	0.30	101	80.992	80.416	81.396	82.482	80.789	94.172	81.250	80.008	81.921
	0.45	101	81.101	80.660	82.054	81.165	80.836	81.496	81.332	81.088	81.631
300	0.10	151	422.982	421.551	423.973	425.611	422.521	466.379	423.312	422.593	424.112
	0.30	151	426.895	421.704	465.063	423.236	421.937	424.155	427.746	422.930	466.564
	0.45	151	427.169	421.060	467.341	427.391	421.636	466.195	423.404	422.699	424.172
400	0.10	201	1390.836	1378.586	1483.780	–	–	–	–	–	–
	0.30	201	1389.465	1373.600	1483.066	–	–	–	–	–	–
	0.45	201	1380.151	1373.839	1383.228	–	–	–	–	–	–

Table 3.1: Average, minimum and maximum CPU times of computation time for BSA for $p = 2$ and various combinations of parameters n and *out* for each dataset.

3.4.3 FAST-LTS and combinations of the algorithms

In this section we present the results of the FAST-LTS algorithm compared to the MOEA-QR and MMEA-QR because, as described in Section 3.4.1, these algorithms seems to be very fast while providing reliable results. We also run simulations on the combinations of those two algorithms as described in

Section 2.5. All algorithms were set to start from 50 randomly chosen subsets for each simulation and maximum number of inner cycles was set to 40. We ran simulations 100 times for each combination of the parameters n, p and *out* for each data set. The results are given in Appendix A in Tables A.7, A.8 and A.9 for data sets $D1$, $D2$ and $D3$ respectively. Beside measuring the cosine similarity and L^2 norm, we also provide the number of the inner cycles for each algorithms. We can see, that providing solution from the FAST-LTS to the MOEA-QR and MMEA-QR significantly reduce those numbers.

In Table 3.2 we can see that approximately in 30 % of cases MOEA-QR and MMEA-QR were able to improve the solution of the FAST-LTS in all three data sets. That means in those cases FAST-LTS algorithm have provided solutions that satisfied weak necessary condition but not the strong one.

	$D1$	$D2$	$D3$
FAST-LTS-MMEA-QR	30.33 %	34.00 %	33.00 %
FAST-LTS-MOEA-QR	30.33 %	35.33 %	33.33 %

Table 3.2: Percentage of the h-elements subsets provided by FAST-LTS which did not satisfied strong necessary condition, and the MMEA-QR and MOEA-QR were able to improve them.

Let us note, that it would be interesting to exhaustively enumerate data sets and count number of h-element subsets which satisfies the weak and strong necessary conditions and, similarly, enumerate the "domains" of each such h-element subsets in terms how many h-elements subsets leads to the particular h-element subset satisfying the weak or strong necessary condition, respectively.

3.4.4 Random algorithm and RBSA

In Section 2.7 we proposed the probabilistic version of BSA. We compare it to the Random solution algorithm (RANDOM) described in Section 2.0.1 and also to the FAST-LTS and MMEA-QR algorithms which were observed to provide efficient solutions in previous sections. In this experiments we set RANDOM and RBSA algorithm to start from 1000 randomly chosen $(p+1)$-element subsets. FSA-LTS was set to start from 100 p-element subsets and MMEA-QR from 100 h-element subsets. Both algorithms were set to the maximum of 50 inner cycles. For each combination of the parameters n, p and *out* we ran 10 simulations. This was repeated for all three data sets. The results are given in Appendix A in Tables A.10, A.11 and A.12 for data sets $D1$, $D2$ and $D3$, respectively, and include average CPU times, cosine similarity and L^2 norm in the same way as above.

In Figure 3.4 box plots describing the quality of the solution are given. The L^2 norm of the RBSA estimate is in average less, the cosine similarity is

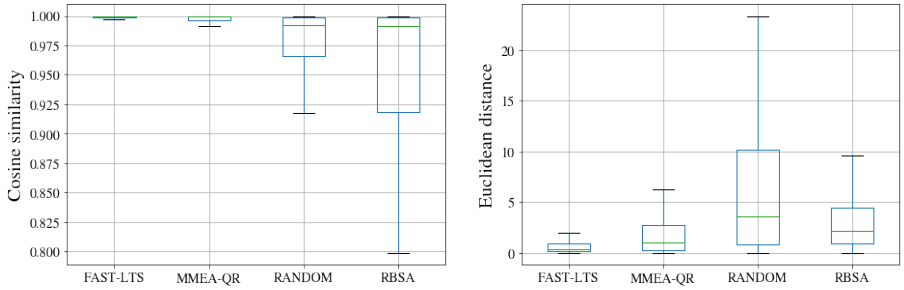

Figure 3.4: Cosine similarity and L^2 norm for multiple algorithms. Random algorithms are outperformed by the algorithms finding the weak and strong necessary conditions.

worse. Moreover we can clearly see that MMEA-QR and FAST-LTS provide much more reliable results (even in shorter CPU times).

There are many other possibilities for other experiments than what is provided in this chapter, especially the experiments suggested in the Section 3.4.3 for exploring domains of h-element subsets satisfying the weak and strong necessary conditions. These experiments are out of the scope of this work, because already provided simulations were quite exhaustive due to the lot of possible variations of the algorithms.

Conclusion

We have surveyed, implemented, and described multiple exact and probabilistic algorithms for calculating the LTS estimate. Those algorithms have been proposed across the last few decades. Most recently proposed algorithms are just a few years old. This means that research in the field of LTS algorithms is still ongoing.

Although the exact algorithms have polynomial time complexity, we showed that currently used probabilistic algorithm provide sufficiently fast solutions which, even though that may not be exact, are good enough. Even though it was proven that the exact solution could not be obtained faster than in polynomial time, we showed that the currently used algorithms could be combined to obtain better results.

Algorithms for calculating the LTS estimate are still the open topic for further research. One of the possible research direction is to study the possibility of combining the exact algorithms with probabilistic ones. As our experimental results suggest, we could come up with probabilistic algorithms which provide even better performance.

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. 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

Results of the experiments

			alg	FSA-I			FSA-QR			MMEA-I			MMEA-QR			MOEA-I			MOEA-QR		
n	p	*out*	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.000	0.983	1.496	0.007	0.991	1.475	0.000	0.895	4.693	0.000	0.990	1.488	0.000	0.699	23.235	0.000	0.988	1.447
		0.30	12	0.000	0.986	2.457	0.006	0.963	2.544	0.000	0.737	11.694	0.000	0.994	1.310	0.000	0.557	44.050	0.000	0.972	3.916
		0.45	12	0.000	0.848	8.643	0.004	0.947	4.337	0.000	0.652	20.731	0.000	0.916	5.594	0.000	0.594	37.565	0.000	0.863	6.180
100	4	0.10	52	0.020	0.998	0.855	0.179	0.998	0.965	0.001	0.965	2.710	0.003	0.998	0.978	0.003	0.877	11.660	0.006	0.998	0.836
		0.30	52	0.021	0.998	0.903	0.186	0.999	0.885	0.001	0.927	5.429	0.003	0.997	0.916	0.002	0.872	10.986	0.007	0.999	0.839
		0.45	52	0.021	0.999	0.611	0.151	0.999	0.656	0.001	0.939	6.489	0.003	0.999	0.653	0.002	0.880	11.619	0.007	0.998	0.640
	6	0.10	53	0.022	0.997	0.925	0.266	0.998	0.910	0.001	0.950	3.637	0.003	0.998	0.761	0.003	0.833	14.147	0.007	0.997	0.765
		0.30	53	0.024	0.997	0.887	0.286	0.998	0.914	0.001	0.915	6.815	0.003	0.998	0.854	0.003	0.831	17.549	0.008	0.997	0.975
		0.45	53	0.025	0.997	0.671	0.220	0.994	1.023	0.001	0.895	8.875	0.003	0.998	0.740	0.002	0.854	14.923	0.008	0.993	1.066
500	3	0.10	252	1.708	1.000	0.529	6.854	1.000	0.484	0.076	0.999	0.883	0.172	1.000	0.523	0.152	0.998	0.946	0.496	1.000	0.465
		0.30	252	1.662	1.000	1.176	6.661	1.000	1.153	0.076	0.997	1.973	0.173	1.000	1.194	0.157	0.997	2.053	0.489	1.000	1.217
		0.45	252	1.786	0.999	2.781	7.327	0.999	2.679	0.075	0.998	3.419	0.166	0.999	2.749	0.141	0.998	3.284	0.503	0.999	2.681
	6	0.10	253	2.014	1.000	0.421	8.535	1.000	0.366	0.076	0.994	1.117	0.162	1.000	0.375	0.112	0.994	1.411	0.481	1.000	0.430
		0.30	253	2.062	1.000	1.094	8.701	1.000	0.979	0.071	0.995	2.187	0.162	1.000	1.155	0.105	0.995	2.256	0.486	0.999	1.145
		0.45	253	2.077	0.999	2.632	8.866	0.999	2.464	0.068	0.994	3.868	0.153	0.999	2.602	0.104	0.994	3.735	0.484	0.999	2.719
1000	3	0.10	502	–	–	–	–	–	–	0.149	1.000	0.370	0.509	1.000	0.354	0.379	1.000	0.373	1.865	1.000	0.327
		0.30	502	–	–	–	–	–	–	0.147	0.999	3.021	0.367	0.999	3.029	0.265	1.000	2.911	1.665	1.000	2.909
		0.45	502	–	–	–	–	–	–	0.035	0.996	10.661	0.239	0.996	10.998	0.139	0.995	10.603	1.487	0.996	10.682
	6	0.10	503	–	–	–	–	–	–	0.172	1.000	0.455	0.585	1.000	0.345	0.360	1.000	0.486	1.895	1.000	0.349
		0.30	503	–	–	–	–	–	–	0.124	0.999	3.016	0.440	0.999	3.146	0.242	0.999	2.837	1.697	0.999	2.924
		0.45	503	–	–	–	–	–	–	0.030	0.994	8.955	0.345	0.994	9.092	0.143	0.994	8.933	1.562	0.993	9.095
	11	0.10	506	–	–	–	–	–	–	0.211	1.000	0.412	0.723	1.000	0.265	0.359	1.000	0.400	1.974	1.000	0.292
		0.30	506	–	–	–	–	–	–	0.094	0.999	2.708	0.616	0.999	2.657	0.230	0.999	2.622	1.874	0.999	2.559
		0.45	506	–	–	–	–	–	–	0.037	0.993	8.305	0.561	0.992	8.335	0.170	0.992	8.338	1.810	0.992	8.113
	21	0.10	511	–	–	–	–	–	–	0.204	1.000	0.449	1.186	1.000	0.294	0.457	1.000	0.577	2.719	1.000	0.315
		0.30	511	–	–	–	–	–	–	0.076	0.999	2.943	1.053	0.999	3.057	0.284	0.999	2.901	2.431	0.999	2.874
		0.45	511	–	–	–	–	–	–	0.051	0.989	10.646	1.018	0.991	11.064	0.280	0.991	10.797	2.464	0.991	10.594
mean				0.763	0.987	1.739	3.217	0.992	1.456	0.066	0.956	5.054	0.320	0.994	2.760	0.152	0.924	9.304	0.979	0.991	2.828

Table A.1: Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D1$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm.

			alg	FSA-I			FSA-QR			MMEA-I			MMEA-QR			MOEA-I			MOEA-QR		
n	p	*out*	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.000	0.990	1.945	0.006	0.992	1.973	0.000	0.888	7.989	0.000	0.993	1.645	0.000	0.734	35.866	0.000	0.992	1.733
		0.30	12	0.000	0.870	9.287	0.006	0.883	10.860	0.000	0.582	23.583	0.000	0.869	11.504	0.000	0.506	47.430	0.000	0.953	7.286
		0.45	12	0.000	0.691	21.370	0.004	0.749	16.951	0.000	0.545	32.270	0.000	0.716	17.784	0.000	0.533	38.293	0.000	0.732	17.072
100	4	0.10	52	0.019	0.998	0.984	0.165	0.998	0.943	0.001	0.771	8.848	0.003	0.998	0.937	0.003	0.621	27.151	0.006	0.998	1.041
		0.30	52	0.021	0.983	1.930	0.203	0.983	1.809	0.001	0.699	17.484	0.003	0.995	2.020	0.003	0.681	32.773	0.007	0.992	1.485
		0.45	52	0.019	0.837	9.497	0.123	0.830	11.490	0.001	0.677	30.080	0.003	0.894	7.186	0.003	0.660	35.166	0.006	0.894	8.129
	6	0.10	53	0.021	0.997	0.863	0.271	0.998	0.835	0.001	0.689	20.252	0.003	0.998	0.861	0.004	0.572	48.255	0.007	0.998	0.904
		0.30	53	0.025	0.986	3.821	0.290	0.972	4.816	0.001	0.672	34.943	0.003	0.996	1.514	0.004	0.670	49.795	0.008	0.991	2.085
		0.45	53	0.025	0.892	18.193	0.189	0.896	19.771	0.001	0.627	58.788	0.003	0.883	20.926	0.004	0.662	81.034	0.008	0.908	15.608
500	3	0.10	252	1.644	1.000	0.420	6.933	1.000	0.452	0.014	0.936	1.525	0.072	1.000	0.439	0.041	0.930	1.919	0.402	1.000	0.451
		0.30	252	1.669	0.999	0.547	7.049	1.000	0.540	0.014	0.814	4.182	0.074	0.999	0.603	0.043	0.821	4.607	0.415	1.000	0.593
		0.45	252	1.744	0.863	7.336	7.205	0.910	4.894	0.015	0.693	14.545	0.078	0.888	8.072	0.043	0.725	14.494	0.422	0.920	5.066
	6	0.10	253	2.001	0.999	0.507	8.659	1.000	0.493	0.015	0.879	5.676	0.104	0.999	0.421	0.048	0.848	8.561	0.435	0.999	0.463
		0.30	253	2.095	0.999	3.193	8.913	0.999	3.078	0.015	0.817	16.367	0.107	1.000	3.148	0.047	0.820	17.580	0.446	1.000	2.929
		0.45	253	2.049	0.922	13.144	8.838	0.916	14.667	0.015	0.769	44.826	0.107	0.911	14.810	0.043	0.768	43.490	0.433	0.906	15.377
1000	3	0.10	502	–	–	–	–	–	–	0.027	1.000	0.278	0.237	1.000	0.236	0.134	1.000	0.253	1.478	1.000	0.249
		0.30	502	–	–	–	–	–	–	0.029	0.999	0.634	0.267	1.000	0.653	0.146	1.000	0.654	1.582	1.000	0.647
		0.45	502	–	–	–	–	–	–	0.030	0.811	1.996	0.282	0.728	1.775	0.149	0.878	1.969	1.598	0.821	1.972
	6	0.10	503	–	–	–	–	–	–	0.030	1.000	0.306	0.348	1.000	0.255	0.146	1.000	0.298	1.571	1.000	0.285
		0.30	503	–	–	–	–	–	–	0.032	1.000	1.634	0.391	1.000	1.633	0.155	1.000	1.659	1.665	1.000	1.648
		0.45	503	–	–	–	–	–	–	0.035	0.748	5.133	0.439	0.705	4.894	0.158	0.708	5.064	1.728	0.686	4.304
	11	0.10	506	–	–	–	–	–	–	0.037	1.000	0.311	0.559	1.000	0.301	0.184	1.000	0.345	1.820	1.000	0.301
		0.30	506	–	–	–	–	–	–	0.038	0.977	4.805	0.584	0.980	3.649	0.181	0.983	4.620	1.849	0.992	3.478
		0.45	506	–	–	–	–	–	–	0.039	0.739	8.669	0.616	0.711	7.955	0.175	0.781	8.871	1.868	0.763	8.356
	21	0.10	511	–	–	–	–	–	–	0.053	0.999	0.432	1.024	1.000	0.277	0.294	0.993	0.683	2.579	1.000	0.326
		0.30	511	–	–	–	–	–	–	0.055	0.903	13.742	1.115	0.933	9.280	0.294	0.903	13.823	2.512	0.945	9.462
		0.45	511	–	–	–	–	–	–	0.055	0.646	15.593	1.151	0.626	13.667	0.279	0.645	15.189	2.610	0.609	13.806
mean				0.756	0.935	6.202	3.257	0.942	6.238	0.021	0.810	13.885	0.281	0.919	5.053	0.096	0.794	19.994	0.943	0.930	4.632

Table A.2: Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D2$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm.

			alg	FSA-I			FSA-QR			MMEA-I			MMEA-QR			MOEA-I			MOEA-QR		
n	p	*out*	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.000	0.987	1.648	0.007	0.993	1.584	0.000	0.850	9.931	0.000	0.992	1.505	0.000	0.625	37.956	0.000	0.990	1.749
		0.30	12	0.000	0.978	2.893	0.007	0.980	2.322	0.000	0.595	23.852	0.000	0.957	2.431	0.000	0.440	55.069	0.000	0.973	3.275
		0.45	12	0.000	0.855	9.627	0.003	0.898	8.978	0.000	0.641	26.917	0.000	0.889	10.512	0.000	0.621	43.391	0.000	0.910	13.152
	4	0.10	52	0.021	0.998	1.030	0.192	0.998	0.916	0.001	0.917	4.816	0.003	0.998	0.877	0.004	0.757	24.437	0.007	0.998	0.956
		0.30	52	0.021	0.998	0.849	0.197	0.998	0.795	0.001	0.752	10.570	0.003	0.997	0.811	0.003	0.684	28.215	0.007	0.998	0.789
		0.45	52	0.023	0.999	0.564	0.155	0.999	0.554	0.001	0.778	13.069	0.003	0.999	0.535	0.003	0.742	22.952	0.008	0.999	0.561
	6	0.10	53	0.023	0.997	1.075	0.291	0.997	0.943	0.001	0.723	15.112	0.003	0.997	0.876	0.004	0.625	39.288	0.008	0.997	0.942
		0.30	53	0.024	0.997	0.746	0.271	0.998	0.740	0.001	0.674	21.592	0.003	0.997	1.111	0.004	0.603	51.948	0.008	0.998	0.779
		0.45	53	0.027	0.999	0.616	0.240	0.996	1.404	0.001	0.642	40.547	0.004	0.997	1.006	0.004	0.645	52.690	0.009	0.997	0.759
500	3	0.10	252	1.736	1.000	0.495	7.368	1.000	0.439	0.015	0.974	1.041	0.077	1.000	0.495	0.045	0.925	1.754	0.424	1.000	0.503
		0.30	252	1.625	1.000	0.930	6.812	1.000	0.838	0.014	0.951	3.346	0.070	0.999	0.868	0.040	0.958	2.699	0.395	1.000	0.948
		0.45	252	1.712	0.999	2.197	7.186	0.999	2.046	0.015	0.924	4.274	0.074	0.999	2.124	0.042	0.908	4.275	0.413	1.000	2.088
	6	0.10	253	2.114	0.999	0.469	9.017	1.000	0.440	0.016	0.931	2.587	0.110	0.999	0.472	0.052	0.791	8.573	0.448	0.999	0.480
		0.30	253	2.138	1.000	0.964	9.101	1.000	0.951	0.016	0.898	6.597	0.110	1.000	1.054	0.049	0.873	8.730	0.456	1.000	1.035
		0.45	253	2.044	0.999	2.216	8.850	0.999	1.986	0.015	0.853	9.545	0.107	0.999	2.201	0.044	0.872	8.614	0.434	0.999	2.160
1000	3	0.10	502	–	–	–	–	–	–	0.030	1.000	0.321	0.299	1.000	0.279	0.152	1.000	0.295	1.645	1.000	0.278
		0.30	502	–	–	–	–	–	–	0.030	1.000	1.132	0.323	1.000	1.081	0.154	1.000	1.041	1.688	1.000	1.060
		0.45	502	–	–	–	–	–	–	0.163	0.998	3.358	0.520	0.998	3.559	0.401	0.998	3.452	1.774	0.999	3.469
	6	0.10	503	–	–	–	–	–	–	0.032	1.000	0.366	0.427	1.000	0.285	0.157	0.999	0.417	1.702	1.000	0.307
		0.30	503	–	–	–	–	–	–	0.032	1.000	1.856	0.418	1.000	1.688	0.155	1.000	1.772	1.707	1.000	1.755
		0.45	503	–	–	–	–	–	–	0.183	0.996	5.619	0.596	0.996	5.572	0.368	0.997	5.583	1.823	0.996	5.515
	11	0.10	506	–	–	–	–	–	–	0.038	1.000	0.380	0.645	1.000	0.265	0.183	0.999	0.362	1.936	1.000	0.259
		0.30	506	–	–	–	–	–	–	0.037	0.999	2.249	0.596	0.999	2.330	0.175	0.999	2.364	1.877	0.999	2.172
		0.45	506	–	–	–	–	–	–	0.210	0.993	8.146	0.752	0.993	8.049	0.358	0.992	8.073	1.950	0.992	8.198
	21	0.10	511	–	–	–	–	–	–	0.055	0.999	0.462	1.239	1.000	0.356	0.279	0.997	0.554	2.785	1.000	0.321
		0.30	511	–	–	–	–	–	–	0.053	0.997	3.195	1.132	0.999	2.661	0.271	0.996	3.057	2.530	0.999	2.730
		0.45	511	–	–	–	–	–	–	0.206	0.968	13.257	1.212	0.971	12.368	0.436	0.969	12.822	2.614	0.971	12.274
mean				0.767	0.987	1.755	3.313	0.990	1.662	0.043	0.891	8.672	0.323	0.992	2.421	0.125	0.852	15.940	0.987	0.993	2.538

Table A.3: Results of simulations for algorithms finding h-element subsets satisfying the strong necessary condition for the data set $D3$ for various configurations of the parameters n, p and *out*. Results include average time and cosine similarity and L^2 norm.

			alg	BAB			BSA			EXACT			FSA-QR-BAB			FSA-QR-BSA		
n	p	*out*	h	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max
15	5	0.10	10	0.003	0.002	0.005	2.362	1.873	2.944	0.008	0.007	0.014	0.002	0.001	0.004	2.363	1.811	3.358
		0.30	10	0.002	0.002	0.003	2.063	1.777	2.476	0.008	0.007	0.008	0.002	0.001	0.003	2.061	1.773	2.486
		0.45	10	0.003	0.002	0.005	2.040	1.781	2.425	0.007	0.007	0.008	0.002	0.001	0.004	2.045	1.782	2.438
	20	0.10	12	0.017	0.012	0.034	1.839	1.744	1.994	0.283	0.278	0.306	0.009	0.004	0.019	1.838	1.764	1.994
		0.30	12	0.016	0.012	0.025	1.868	1.740	2.040	0.290	0.272	0.327	0.007	0.004	0.020	1.866	1.732	2.032
		0.45	12	0.014	0.011	0.027	1.855	1.751	2.019	0.289	0.272	0.307	0.007	0.003	0.016	1.851	1.743	1.995
	5	0.10	13	0.022	0.018	0.036	13.814	12.808	15.674	0.204	0.194	0.217	0.010	0.005	0.022	13.766	12.676	15.680
		0.30	13	0.021	0.017	0.033	13.441	12.564	14.544	0.204	0.194	0.238	0.009	0.004	0.017	13.389	12.593	14.420
		0.45	13	0.020	0.017	0.030	13.577	12.651	14.664	0.206	0.193	0.234	0.009	0.005	0.020	13.525	12.610	14.626
25	4	0.10	15	0.208	0.099	0.563	7.706	5.996	14.464	9.814	7.242	16.872	0.066	0.011	0.350	7.689	5.986	14.861
		0.30	15	0.180	0.096	0.437	7.737	5.974	9.681	9.491	7.314	11.406	0.047	0.009	0.277	7.734	5.882	9.633
		0.45	15	0.196	0.143	0.400	9.097	8.950	9.402	11.213	11.081	11.324	0.049	0.013	0.139	9.076	8.831	9.421
30	3	0.10	17	1.709	0.806	3.700	1.485	1.437	2.433	368.801	366.077	375.885	0.427	0.067	1.500	1.489	1.437	2.432
		0.30	17	1.189	0.743	2.176	1.494	1.478	1.518	—	—	—	0.305	0.135	0.722	1.495	1.472	1.531
		0.45	17	1.064	0.731	2.929	1.491	1.472	1.536	—	—	—	0.190	0.040	0.523	1.487	1.471	1.508
	4	0.10	17	1.733	0.858	3.440	24.221	24.009	24.645	—	—	—	0.389	0.117	1.063	24.083	23.932	24.228
		0.30	17	1.110	0.519	1.840	24.131	23.817	24.388	—	—	—	0.202	0.086	0.370	24.133	23.875	24.392
		0.45	17	0.908	0.717	1.545	24.107	23.750	24.265	—	—	—	0.384	0.067	0.948	24.109	23.782	24.421
40	3	0.10	22	103.891	58.453	183.988	9.986	8.386	11.580	—	—	—	16.839	2.723	43.747	9.999	8.386	11.617
		0.30	22	78.415	42.544	118.603	8.397	8.316	8.445	—	—	—	6.615	0.864	26.232	8.402	8.377	8.434
		0.45	22	37.435	22.998	66.874	8.404	8.338	8.442	—	—	—	10.809	0.273	55.871	8.409	8.340	8.454

Table A.4: Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D1$ for various configurations of the parameters n, p and *out*.

			alg	BAB			BSA			EXACT			FSA-QR-BAB			FSA-QR-BSA		
n	p	*out*	h	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max
15	5	0.10	10	0.003	0.002	0.004	2.132	1.782	2.648	0.008	0.007	0.010	0.002	0.001	0.003	2.130	1.782	2.660
		0.30	10	0.003	0.002	0.004	2.105	1.794	2.460	0.008	0.007	0.012	0.002	0.001	0.003	2.104	1.770	2.487
		0.45	10	0.003	0.002	0.004	2.154	1.801	2.543	0.008	0.007	0.010	0.002	0.002	0.004	2.149	1.801	2.535
	20	0.10	12	0.018	0.013	0.033	1.922	1.851	2.026	0.297	0.277	0.335	0.010	0.004	0.018	1.923	1.808	2.051
		0.30	12	0.017	0.012	0.025	1.860	1.744	2.022	0.288	0.271	0.314	0.009	0.004	0.022	1.863	1.747	2.037
		0.45	12	0.016	0.012	0.024	1.829	1.749	2.006	0.283	0.270	0.329	0.008	0.003	0.017	1.831	1.755	2.053
	5	0.10	13	0.022	0.017	0.035	13.544	12.781	14.983	0.200	0.193	0.218	0.011	0.005	0.023	13.476	12.739	14.920
		0.30	13	0.021	0.017	0.033	13.388	12.676	14.732	0.201	0.194	0.275	0.010	0.005	0.021	13.337	12.656	14.594
		0.45	13	0.022	0.017	0.037	13.584	12.720	15.150	0.202	0.194	0.216	0.011	0.006	0.022	13.534	12.713	15.235
25	4	0.10	15	0.260	0.152	0.449	9.144	8.960	9.384	11.217	11.048	11.369	0.089	0.024	0.267	9.155	8.923	9.773
		0.30	15	0.250	0.149	0.549	9.118	8.928	9.335	11.224	11.009	11.370	0.071	0.019	0.206	9.108	8.952	9.309
		0.45	15	0.230	0.144	0.398	9.106	8.808	9.372	11.225	11.112	11.331	0.076	0.021	0.289	9.110	8.901	10.166
30	3	0.10	17	1.555	0.975	2.536	1.483	1.465	1.508	–	–	–	0.349	0.138	0.995	1.486	1.467	1.526
		0.30	17	1.629	0.977	3.007	1.474	1.462	1.485	–	–	–	0.349	0.166	0.630	1.476	1.464	1.490
		0.45	17	1.212	0.759	1.859	1.476	1.465	1.485	–	–	–	0.349	0.076	0.980	1.473	1.465	1.487
	4	0.10	17	1.260	0.888	1.786	24.173	23.993	24.342	–	–	–	0.389	0.165	0.728	24.153	23.977	24.407
		0.30	17	1.449	0.672	1.985	24.201	23.964	24.474	–	–	–	0.217	0.113	0.385	24.136	24.006	24.303
		0.45	17	1.574	0.921	2.425	24.195	24.020	24.511	–	–	–	0.496	0.184	1.078	24.222	24.035	24.423
40	3	0.10	22	107.905	35.374	222.922	8.412	8.365	8.470	–	–	–	20.629	3.565	53.692	8.419	8.369	8.465
		0.30	22	103.866	42.430	161.180	8.477	8.410	8.581	–	–	–	16.070	3.764	44.978	8.493	8.422	8.628
		0.45	22	93.810	31.350	161.078	8.430	8.385	8.468	–	–	–	9.253	1.498	29.853	8.430	8.397	8.466

Table A.5: Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D2$ for various configurations of the parameters n, p and *out*.

				BAB			BSA			EXACT			FSA-QR-BAB			FSA-QR-BSA		
n	p	*out*	alg h	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max
15	5	0.10	10	0.003	0.002	0.005	2.354	2.001	3.149	0.009	0.008	0.012	0.002	0.002	0.003	2.356	1.994	3.167
		0.30	10	0.003	0.002	0.004	2.168	1.834	2.662	0.008	0.007	0.010	0.002	0.002	0.005	2.167	1.833	2.643
		0.45	10	0.003	0.002	0.005	2.039	1.766	2.476	0.008	0.007	0.008	0.002	0.001	0.003	2.034	1.774	2.390
	20	0.10	12	0.018	0.013	0.031	1.896	1.756	2.017	0.291	0.274	0.312	0.009	0.003	0.025	1.892	1.748	2.017
		0.30	12	0.017	0.012	0.029	1.898	1.721	1.993	0.296	0.274	0.304	0.008	0.004	0.017	1.898	1.731	2.031
		0.45	12	0.015	0.011	0.024	1.792	1.720	2.052	0.280	0.275	0.308	0.008	0.003	0.015	1.790	1.721	1.990
	5	0.10	13	0.021	0.017	0.032	13.360	12.684	14.357	0.198	0.194	0.203	0.011	0.006	0.020	13.326	12.722	14.411
		0.30	13	0.021	0.017	0.032	13.588	12.546	15.029	0.205	0.193	0.223	0.010	0.005	0.022	13.554	12.513	14.811
		0.45	13	0.021	0.017	0.031	13.610	12.798	14.894	0.204	0.188	0.244	0.010	0.005	0.022	13.546	12.675	14.851
25	4	0.10	15	0.265	0.165	0.473	9.126	8.946	9.366	11.227	11.000	11.394	0.081	0.020	0.224	9.119	8.911	9.380
		0.30	15	0.228	0.149	0.419	9.095	8.910	9.318	11.229	11.024	11.346	0.063	0.017	0.170	9.096	8.909	9.324
		0.45	15	0.207	0.143	0.378	9.097	8.940	9.307	11.229	11.017	11.395	0.058	0.019	0.153	9.091	8.861	9.356
30	3	0.10	17	1.224	0.871	1.585	1.473	1.445	1.496	–	–	–	0.293	0.074	0.562	1.472	1.445	1.491
		0.30	17	1.409	1.151	1.670	1.473	1.455	1.505	–	–	–	0.531	0.176	1.519	1.471	1.461	1.492
		0.45	17	1.158	0.730	1.625	1.462	1.441	1.491	–	–	–	0.244	0.073	0.464	1.464	1.449	1.481
	4	0.10	17	1.307	0.765	2.136	24.280	24.103	24.472	–	–	–	0.380	0.251	0.673	24.287	24.122	24.415
		0.30	17	1.524	0.899	2.210	24.222	23.975	24.552	–	–	–	0.323	0.113	0.781	24.208	23.829	24.618
		0.45	17	1.235	0.866	2.301	24.180	23.917	24.441	–	–	–	0.274	0.128	0.673	24.148	23.927	24.403
40	3	0.10	22	121.401	56.588	240.316	8.390	8.322	8.452	–	–	–	21.435	7.312	50.445	8.396	8.335	8.504
		0.30	22	93.298	38.782	174.062	8.380	8.321	8.458	–	–	–	7.905	1.334	17.158	8.369	8.312	8.413
		0.45	22	89.391	31.266	174.557	8.394	8.372	8.478	–	–	–	8.883	1.382	38.292	8.380	8.356	8.390

Table A.6: Average, minimum and maximum CPU times of simulations for exact algorithms for the data set $D3$ for various configurations of the parameters n, p and *out*.

			alg	FAST-LTS			FAST-LTS-MMEA-QR			FAST-LTS-MOEA-QR			MMEA-QR			MOEA-QR		
n	p	*out*	h	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter
20	3	0.10	12	0.9911	1.7474	4.4	0.9996	0.8350	1.3	0.9996	0.8350	0.9	0.9912	1.6739	5.2	0.9912	1.6739	4.9
		0.30	12	0.9927	1.1167	4.2	0.9998	0.7831	1.4	0.9998	0.7831	0.9	0.9933	1.4301	5.5	0.9933	1.4301	5.5
		0.45	12	0.9743	3.2395	4.4	1.0000	0.9654	1.2	1.0000	0.9654	0.9	0.9751	2.5531	5.2	0.9751	2.5531	5.0
100	4	0.10	52	0.9985	1.0772	8.6	0.9992	0.8552	8.2	0.9992	0.8552	4.7	0.9981	1.0396	27.1	0.9981	1.0396	28.8
		0.30	52	0.9976	1.0727	8.4	0.9991	0.6979	8.8	0.9991	0.6979	7.8	0.9986	1.0682	29.6	0.9980	0.9500	29.7
		0.45	52	0.9994	0.3328	8.3	0.9995	0.2532	13.2	0.9995	0.2532	10.1	0.9995	0.2532	31.0	0.9995	0.2532	29.3
	6	0.10	53	0.9932	0.8824	8.2	0.9983	0.8412	9.7	0.9983	0.8625	5.5	0.9950	1.0828	27.2	0.9950	1.1060	26.3
		0.30	53	0.9968	0.7212	9.0	0.9994	0.6583	12.5	0.9994	0.6583	9.3	0.9970	0.7211	27.4	0.9970	0.7211	26.5
		0.45	53	0.9988	0.4321	9.6	0.9981	0.3899	13.4	0.9984	0.3784	6.7	0.9980	0.4581	25.8	0.9980	0.4581	27.6
500	3	0.10	252	0.9996	0.6092	15.9	0.9996	0.4456	19.1	0.9997	0.3768	8.0	0.9999	0.3198	40.0	0.9999	0.4262	40.0
		0.30	252	0.9993	0.6235	13.9	0.9998	1.1351	1.8	0.9999	1.1693	1.7	0.9986	3.4802	40.0	0.9994	3.1676	40.0
		0.45	252	0.9998	0.3413	10.9	0.9988	1.1901	2.2	1.0000	1.2632	0.0	0.9977	6.7589	40.0	0.9961	6.4466	40.0
	6	0.10	253	0.9991	0.3925	16.6	0.9998	0.4261	19.6	0.9996	0.2710	12.1	0.9995	0.4136	40.0	0.9997	0.2213	40.0
		0.30	253	0.9992	0.4242	15.9	0.9997	1.0593	7.0	0.9997	0.9453	2.7	0.9983	2.9997	40.0	0.9982	2.3554	40.0
		0.45	253	0.9998	0.4407	11.3	0.9999	1.2118	2.4	0.9998	1.1649	1.8	0.9845	10.6046	40.0	0.9797	10.5189	40.0

Table A.7: Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D1$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR.

			alg	FAST-LTS			FAST-LTS-MMEA-QR			FAST-LTS-MOEA-QR			MMEA-QR			MOEA-QR		
n	p	*out*	h	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter
20	3	0.10	12	0.9924	1.5773	4.1	0.9979	0.9260	1.9	0.9979	0.9260	0.9	0.9860	1.9837	4.4	0.9860	1.9837	3.8
		0.30	12	0.9948	1.4486	4.6	0.9989	0.7790	1.2	0.9989	0.7790	1.1	0.9916	1.2389	5.6	0.9916	1.2389	5.3
		0.45	12	0.4031	4.8100	3.9	0.9996	0.9317	1.1	0.9996	0.9317	0.9	0.5309	3.5610	4.0	0.5309	3.5610	4.7
100	4	0.10	52	0.9964	0.9717	8.9	0.9983	0.8211	11.0	0.9982	0.7931	7.6	0.9963	1.2158	26.3	0.9961	1.1273	25.7
		0.30	52	0.9976	1.0532	11.6	0.9988	0.8582	7.7	0.9988	0.8582	4.6	0.9972	1.1817	28.6	0.9972	1.1817	27.7
		0.45	52	0.4044	2.0463	8.2	0.9991	0.8039	6.5	0.9991	0.8039	5.3	0.4609	1.7392	27.6	0.4609	1.7644	25.5
	6	0.10	53	0.9911	1.1592	8.9	0.9989	0.9261	5.6	0.9989	0.9261	3.9	0.9947	1.7288	25.4	0.9954	1.6332	26.7
		0.30	53	0.9976	0.5820	11.8	0.9984	0.5120	11.7	0.9984	0.5120	4.9	0.9987	0.7010	28.6	0.9983	0.6525	28.8
		0.45	53	0.7651	1.7059	7.7	0.9987	0.4663	10.9	0.9987	0.5876	5.3	0.7877	1.5195	29.9	0.8175	0.7120	26.2
500	3	0.10	252	0.9993	0.4512	15.9	0.9997	0.4054	14.9	0.9996	0.3376	13.0	0.9994	0.3613	40.0	0.9997	0.2560	40.0
		0.30	252	0.9981	0.7668	14.8	0.9996	0.5577	17.1	0.9997	0.6251	7.3	0.9999	0.6945	40.0	0.9999	0.9012	40.0
		0.45	252	0.8346	0.6451	12.0	0.9999	1.0925	8.2	0.9997	1.0894	3.4	0.6584	2.2155	40.0	0.6515	2.1403	40.0
	6	0.10	253	0.9980	0.6675	20.9	0.9997	0.4198	15.8	0.9998	0.3686	14.4	0.9997	0.4117	40.0	0.9996	0.4672	40.0
		0.30	253	0.9996	0.4512	17.9	0.9998	0.8423	9.1	0.9999	0.7832	6.7	0.9996	1.4874	40.0	0.9995	1.9879	40.0

Table A.8: Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D2$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR.

			alg	FAST-LTS			FAST-LTS-MMEA-QR			FAST-LTS-MOEA-QR			MMEA-QR			MOEA-QR		
n	p	*out*	h	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter	cos	l2	iter
20	3	0.10	12	0.9817	1.8058	4.1	0.9949	0.9275	1.1	0.9949	0.9275	0.7	0.9833	1.7182	5.0	0.9833	1.7182	5.1
		0.30	12	0.9922	1.1600	4.4	0.9977	0.6342	1.8	0.9977	0.6342	1.2	0.9967	1.0598	4.9	0.9967	1.0598	5.5
		0.45	12	0.9306	1.3368	3.4	0.9982	0.8009	1.3	0.9982	0.8009	1.0	0.9520	1.2080	4.8	0.9520	1.2080	4.3
100	4	0.10	52	0.9983	1.2923	8.4	0.9997	0.7787	6.3	0.9997	0.7787	2.4	0.9987	1.0823	26.7	0.9987	1.0823	25.7
		0.30	52	0.9983	0.8638	9.3	0.9995	0.7193	12.3	0.9995	0.7193	5.9	0.9993	1.0276	27.8	0.9993	1.0276	26.2
		0.45	52	0.9988	0.7391	8.3	0.9990	0.5489	13.7	0.9990	0.5489	4.4	0.9990	0.7185	27.9	0.9990	0.7185	26.6
	6	0.10	53	0.9892	2.1644	8.3	0.9974	1.0061	10.1	0.9970	1.0014	4.8	0.9913	1.5589	26.8	0.9912	1.5413	25.5
		0.30	53	0.9981	1.0631	9.8	0.9995	0.5988	11.3	0.9993	0.5897	7.6	0.9985	0.8735	25.7	0.9985	0.8735	27.5
		0.45	53	0.9983	0.5501	9.9	0.9985	0.7319	9.2	0.9985	0.7319	5.7	0.9977	0.7688	27.5	0.9977	0.7688	27.7
500	3	0.10	252	0.9997	0.3449	17.0	0.9999	0.2931	15.2	0.9999	0.1774	7.4	0.9999	0.3153	40.0	0.9999	0.2602	40.0
		0.30	252	0.9994	0.6810	15.2	0.9998	0.8462	9.4	0.9997	0.5963	9.7	0.9993	0.6633	40.0	0.9990	1.0597	40.0
		0.45	252	0.9999	0.3151	14.2	0.9982	0.9582	9.5	0.9970	0.9824	5.0	0.9972	2.0233	40.0	0.9968	1.7131	40.0
	6	0.10	253	0.9996	0.4021	20.9	0.9999	0.2557	18.2	0.9999	0.1791	15.4	0.9998	0.1904	40.0	0.9999	0.1477	40.0
		0.30	253	0.9997	0.7210	18.3	0.9997	0.6519	18.2	0.9997	0.8369	6.2	0.9995	0.9374	40.0	0.9993	0.6734	40.0

Table A.9: Results of simulations for combination of FAST-LTS with MOEA-QR and MMEA-QR versus their versions which are not combined, all for the data set $D3$. Results include average cosine similarity, L^2 norm and number of inner cycles of the algorithms. In case of the combined versions, inner cycles represents only cycles of the second algorithm MOEA-QR or MMEA-QR.

			alg	FAST-LTS			MMEA-QR			RANDOM			RBSA		
n	p	out	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.001	0.990	1.629	0.000	0.977	2.153	0.002	0.993	1.762	0.037	0.966	1.811
		0.30	12	0.001	0.997	0.642	0.000	0.987	0.597	0.002	0.990	0.520	0.057	0.998	0.841
		0.45	12	0.001	0.970	0.734	0.000	1.000	1.955	0.002	0.952	6.328	0.057	0.960	1.522
100	4	0.10	52	0.002	0.998	1.297	0.002	0.997	0.967	0.005	0.998	0.986	0.171	0.993	1.157
		0.30	52	0.002	0.999	0.727	0.003	0.998	0.744	0.006	0.973	9.471	0.174	0.997	0.885
		0.45	52	0.002	0.999	1.008	0.003	0.999	0.881	0.005	0.912	18.635	0.175	0.977	3.611
	6	0.10	53	0.002	0.998	0.899	0.003	0.998	1.439	0.006	0.999	0.731	1.280	0.996	0.667
		0.30	53	0.003	0.996	1.241	0.003	0.996	1.114	0.007	0.942	11.855	1.277	0.993	1.861
		0.45	53	0.003	0.997	0.674	0.004	0.994	0.857	0.007	0.757	34.235	1.299	0.936	7.298
500	3	0.10	252	0.043	0.999	0.658	0.208	1.000	0.382	0.093	0.994	3.991	0.460	0.990	1.611
		0.30	252	0.041	1.000	0.409	0.210	0.999	2.880	0.095	0.987	14.196	0.446	0.984	5.212
		0.45	252	0.039	1.000	0.579	0.211	0.989	7.617	0.098	0.960	24.051	0.441	0.655	17.162
	6	0.10	253	0.060	1.000	0.360	0.300	1.000	0.334	0.123	0.990	4.738	3.675	0.992	1.413
		0.30	253	0.061	0.999	0.813	0.291	0.999	2.705	0.125	0.984	11.385	3.663	0.814	9.525
		0.45	253	0.056	1.000	0.663	0.273	0.979	14.460	0.144	0.920	40.441	4.009	0.761	21.161
1000	3	0.10	502	0.080	1.000	0.308	0.370	1.000	0.274	0.174	0.998	5.896	0.774	0.996	1.535
		0.30	502	0.074	1.000	0.421	0.357	0.998	7.809	0.177	0.998	17.351	0.758	0.934	5.605
		0.45	502	0.070	1.000	0.451	0.345	0.991	25.335	0.179	0.984	41.768	0.758	0.737	16.918
	6	0.10	503	0.116	1.000	0.263	0.427	1.000	0.173	0.233	0.994	4.048	7.532	0.994	0.960
		0.30	503	0.103	1.000	0.451	0.419	0.993	10.930	0.240	0.978	24.151	7.543	0.736	26.221
		0.45	503	0.096	1.000	0.442	0.390	0.972	20.910	0.232	0.944	35.656	7.473	0.663	20.738

Table A.10: Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D1$.

			alg	FAST-LTS			MMEA-QR			RANDOM			RBSA		
n	p	*out*	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.001	0.968	1.469	0.000	0.976	1.727	0.002	0.995	0.985	0.061	0.977	1.533
		0.30	12	0.001	0.606	2.568	0.000	0.537	2.674	0.002	0.991	1.621	0.060	0.617	2.770
		0.45	12	0.001	0.656	2.141	0.000	0.817	2.453	0.002	0.749	2.287	0.058	0.654	2.094
100	4	0.10	52	0.003	0.992	1.596	0.003	0.997	1.765	0.006	0.999	0.602	0.193	0.991	1.531
		0.30	52	0.002	0.987	0.970	0.003	0.999	0.927	0.006	0.995	2.420	0.191	0.926	1.526
		0.45	52	0.002	0.884	0.692	0.003	0.643	2.169	0.006	0.513	3.098	0.187	0.790	2.598
	6	0.10	53	0.003	0.999	0.783	0.003	0.997	1.065	0.007	0.999	0.812	1.407	0.994	1.180
		0.30	53	0.003	0.998	0.835	0.003	0.999	0.710	0.007	0.862	4.556	1.399	0.904	1.406
		0.45	53	0.003	0.657	2.025	0.003	0.786	2.638	0.007	0.780	6.363	1.394	0.527	3.739
500	3	0.10	252	0.042	1.000	0.735	0.218	1.000	0.399	0.106	0.999	0.658	0.433	0.987	1.331
		0.30	252	0.043	1.000	0.275	0.210	1.000	0.438	0.105	0.986	2.572	0.415	0.676	2.142
		0.45	252	0.046	0.707	0.602	0.191	0.422	1.612	0.103	0.879	2.704	0.398	0.143	3.024
	6	0.10	253	0.066	0.999	0.465	0.246	1.000	0.352	0.142	0.997	2.815	3.953	0.992	1.973
		0.30	253	0.063	0.999	0.633	0.228	1.000	1.341	0.133	0.967	6.012	3.739	0.861	4.876
		0.45	253	0.061	0.888	0.744	0.210	0.331	3.519	0.141	0.715	5.721	3.895	0.401	5.495
1000	3	0.10	502	0.086	1.000	0.584	0.315	1.000	0.336	0.186	1.000	0.517	0.746	0.992	1.690
		0.30	502	0.080	0.999	0.700	0.297	0.999	1.660	0.174	0.982	2.374	0.692	0.675	1.691
		0.45	502	0.083	0.982	0.205	0.282	0.819	1.873	0.191	0.823	1.649	0.717	0.935	2.925
	6	0.10	503	0.118	1.000	0.374	0.366	1.000	0.231	0.227	0.999	1.993	7.526	0.929	2.949
		0.30	503	0.120	1.000	0.169	0.351	0.998	4.336	0.218	0.969	4.774	7.954	0.948	4.074
		0.45	503	0.118	0.624	1.290	0.336	0.695	5.400	0.196	0.856	6.080	8.003	0.659	10.294

Table A.11: Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D2$.

			alg	FAST-LTS			MMEA-QR			RANDOM			RBSA		
n	p	*out*	h	time	cos	l2	time	cos	l2	time	cos	l2	time	cos	l2
20	3	0.10	12	0.001	0.986	1.558	0.000	0.989	1.154	0.002	0.981	1.280	0.053	0.986	1.512
		0.30	12	0.001	0.951	1.820	0.000	0.951	1.752	0.002	0.974	1.715	0.054	0.952	1.954
		0.45	12	0.001	0.966	1.535	0.000	0.863	2.371	0.002	0.803	3.419	0.050	0.893	1.653
100	4	0.10	52	0.002	0.994	1.382	0.002	0.996	1.739	0.006	0.999	0.333	0.186	0.993	1.121
		0.30	52	0.002	0.998	0.655	0.003	0.999	0.904	0.006	0.992	4.286	0.187	0.994	1.207
		0.45	52	0.002	0.999	0.668	0.003	0.999	0.622	0.005	0.896	8.861	0.173	0.942	3.805
	6	0.10	53	0.003	0.997	0.986	0.003	0.997	0.983	0.007	0.999	0.645	1.385	0.991	1.900
		0.30	53	0.003	0.997	0.922	0.003	0.997	0.697	0.007	0.940	4.723	1.371	0.974	2.500
		0.45	53	0.003	0.995	0.943	0.003	0.991	1.273	0.007	0.882	9.381	1.277	0.886	3.395
500	3	0.10	252	0.044	1.000	0.604	0.182	0.999	0.238	0.101	0.998	1.768	0.387	0.978	1.367
		0.30	252	0.046	1.000	0.272	0.157	1.000	0.679	0.102	0.987	8.324	0.367	0.990	2.876
		0.45	252	0.041	1.000	0.372	0.146	0.996	3.765	0.095	0.966	17.092	0.331	0.451	5.853
	6	0.10	253	0.064	0.999	0.338	0.195	1.000	0.361	0.138	0.995	2.366	3.847	0.991	1.315
		0.30	253	0.062	0.999	0.762	0.179	0.999	2.216	0.141	0.981	9.678	3.855	0.855	5.703
		0.45	253	0.059	0.999	0.493	0.163	0.997	4.041	0.131	0.933	15.445	3.532	0.716	10.914
1000	3	0.10	502	0.089	1.000	0.484	0.265	1.000	0.119	0.172	0.999	2.204	0.698	0.995	1.081
		0.30	502	0.080	1.000	0.333	0.261	0.998	3.223	0.168	0.990	10.513	0.656	0.752	6.644
		0.45	502	0.078	1.000	0.206	0.223	0.990	6.142	0.133	0.976	15.205	0.687	0.714	8.161
	6	0.10	503	0.126	0.999	0.304	0.320	1.000	0.175	0.182	0.997	3.464	7.973	0.988	1.753
		0.30	503	0.110	1.000	0.458	0.304	0.998	4.854	0.180	0.978	12.927	7.336	0.855	8.847
		0.45	503	0.111	1.000	0.259	0.286	0.990	9.805	0.148	0.959	21.085	7.977	0.630	14.644

Table A.12: Average CPU time, cosine similarity and L^2 norm of simulations for RANDOM and RBSA compared to FAST-LTS and MMEA-QR for the data set $D3$.

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `lts` ..... 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