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.

April 22, 2019

Acknowledgements

THANKS to everybody

Declaration

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

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

In Prague on April 22, 2019 ....

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

Klíčová slova nejmenší usekané čtverce, LTS odhad, linearní regrese, robustní statistika, nejmenší 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, and hence only suboptimal probabilistic algorithms are usually used in practice. Besides describing those algorithms, we propose a few ways of combining those algorithms to obtain better performance.

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

Contents

List of Figures

Introduction

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

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

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

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

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

In the second chapter, we cover algorithms for calculating the OLS estimate. It is necessary because most of the algorithm used to compute the LTS estimate relies on those algorithms. Next, we describe all currently used algorithms for calculating LTS estimate. They consist of several probabilistic and also few exact ones. In this chapter, we also show that those algorithms can be easily combined to obtain higher speed and performance.

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

Chapter **1**

Least trimmed squares

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

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

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

In practice, we usually deal with multiple dependent variables so we define multiple linear regression model.

Definition 2. The *multiple linear regression model* is

$$y_i = \boldsymbol{x_i}^T \boldsymbol{w} + \varepsilon_i, i = 1, \dots, n. \tag{1.2}$$

It is common to write the whole model in a matrix form

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{\varepsilon} \tag{1.3}$$

where

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

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

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

It this work we talk about the multiple linear regression model, but for simplicity, we call it merely the linear regression model.

Note 3. It is usual refer to given $\boldsymbol{X}$ and y as to ***data set*** and to y_i with corresponding $\boldsymbol{x_i}$ as to ***data sample*** or ***observation***. Sometimes it's also useful to refer to multiple observations as to subset of observations.

1.1.1 Prediction with the linear regression model

The linear regression model contains the vector of actual 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 $\boldsymbol{\hat{w}}$. Then the predicted values of y are given by

$$\hat{y} = \boldsymbol{\hat{w}}^T \boldsymbol{x}. \tag{1.4}$$

True value of y is given by

$$y = \boldsymbol{w^*}^T \boldsymbol{x} + \varepsilon \tag{1.5}$$

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

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

$$\mathbb{E}(y) = \mathbb{E}(\boldsymbol{x}^T \boldsymbol{w}) + \mathbb{E}(\varepsilon) = \mathbb{E}(\boldsymbol{x}^T \boldsymbol{w}) \tag{1.6}$$

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 4. Let us consider that y represents price of the room and $x \in \mathbb{N}$ 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 $y = \boldsymbol{w}^T\boldsymbol{x} + \varepsilon$ 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 $x_0 = 1$ then the corresponding coefficient w_0 of $\boldsymbol{w}$ is called *intercept*. We refer this model as a *model with an intercept*. Intercept then corresponds to expected value of y when all regressors are zero and prevent the problem from example 4. Given that y is a random variable due to the ε, intercept actually corresponds to $\mathbb{E}(\varepsilon) = \mu$. With regards to this fact we can still assume that in the model with intercept $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$. In this work, we consider the model with the intercept unless otherwise specified. This means that we consider $\boldsymbol{x} = (x_1, x_2 \dots x_p)$ where constant $x_1 = 1$ represents intercept.

Note 5. Sometimes in model with 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.

Some other assumptions about $\boldsymbol{\varepsilon}$ are in practice invalid, and we describe this in section 1.3.

1.2 Ordinary least squares

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

$$r_i(\boldsymbol{w}) = y_i - \hat{y}_i = y_i - \boldsymbol{x_i}^T\boldsymbol{w}, \; i = 1, 2, \dots, n. \tag{1.7}$$

This is commonly know as residual sum of squares RSS

$$RSS(\boldsymbol{w}) = \sum_{i=1}^{n} r_i^2(\boldsymbol{w}) = (y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2. \tag{1.8}$$

Definition 6. The RSS as the function of $\boldsymbol{w}$ is an *objective function* for OLS denoted as

$$\mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}(\boldsymbol{w}) = \sum_{i=1}^{n}(y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2 = (\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w}) \tag{1.9}$$

The point of the minimum of this function

$$\hat{\boldsymbol{w}}^{(\boldsymbol{OLS},\boldsymbol{n})} = \operatorname*{arg\,min}_{\boldsymbol{w} \in \mathbb{R}^p} \mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}(\boldsymbol{w}) \tag{1.10}$$

is a definition of ordinary least squares estimate of true regression coefficients $\boldsymbol{w}^*$.

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

$$\frac{\partial \mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}}{\partial w_j} = \sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T\boldsymbol{x_i})(-x_{ij}), \quad j \in \{1,2,\ldots,p\}. \tag{1.11}$$

By this we obtain the gradient

$$\nabla\,\mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})} = -\sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T\boldsymbol{x_i})\boldsymbol{x_i}. \tag{1.12}$$

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

$$-\sum_{i=1}^{n} 2(y_i - \boldsymbol{w}^T\boldsymbol{x_i})\boldsymbol{x_i} = 0 \tag{1.13}$$

that can be rewritten in matrix form as

$$\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} = 0. \tag{1.14}$$

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

$$\frac{\partial^2 \mathrm{OF}^{(OLS,\boldsymbol{X},\boldsymbol{y})}}{\partial w_h \partial w_j} = \sum_{i=1}^{n} 2(-x_{ik})(-x_{ij}), \quad h \in \{1,2,\ldots,p\}. \tag{1.15}$$

We get

$$\boldsymbol{H}_{\mathbf{OF}^{(\boldsymbol{OLS},\boldsymbol{X},\boldsymbol{y})}} = 2\boldsymbol{X}^T\boldsymbol{X}. \tag{1.16}$$

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

$$\boldsymbol{s}^T(2\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{s} = 2(\boldsymbol{X}\boldsymbol{s})^T(\boldsymbol{X}\boldsymbol{s}) = 2\left\|\boldsymbol{X}\boldsymbol{s}\right\| \tag{1.17}$$

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. Give that the solution of (1.14) gives us the global minimum.

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

$$\hat{\boldsymbol{w}}^{(OLS,n)} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \tag{1.18}$$

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

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

1.2.1 Properties of OLS estimation

Gauss-Markov theorem tells us that under specific conditions (assumptions about the regression model) is OLS estimation unbiased and efficient. In other words, Gauss-Markov theorem states that OLS is the best linear unbiased estimator (BLUE). Efficient means that any other linear unbiased estimator has the same or higher variance than OLS. Those conditions are:

- Expected value of errors $\mathbb{E}(\varepsilon_i) = 0, \ i = 1, 2, \ldots, n$.
- Errors are independently distributed and uncorrelated thus $cov(\varepsilon_i, varepsilon_j), \ i, j = 1, 2 \ldots, n, \ \ i \neq j$
- Regressors $\boldsymbol{x_i}, i = 1, 2, \ldots, n$ and corresponding errors $\varepsilon_i, \ i = 1, 2, \ldots, n$ are uncorrelated.
- All errors have same finite variance. This is known as *homoscedasticity*.

Note 7. As described in example 4 the first assumption is usually fulfilled if we use the model with intercept. Rest of the conditions, on the other hand, are rarely met.

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 statistic methods expect some assumptions to work properly and fail if those assumptions are not met. A robust statistic is statistics that produce acceptable results even when data are from some unconventional distributions or if data contains errors which are not normally distributed. We should be highly motivated to use such methods because in practice we are often forced to work with such data on which classical statistics methods provide poor results.

Such assumptions about 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 ordinary least squares to give a good estimate of $\hat{\boldsymbol{w}}$. Unfortunately, in real conditions, these assumptions are often false so that ordinary least squares do not produce acceptable results. One of the most common reasons for false assumptions are abnormal observations called outliers.

Outliers are very common, and they are for instance erroneous measurements such as transmission errors or noise. Another common reason is that nowadays we are mostly given data which were automatically processed by computers. Sometimes we are also presented 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 most of the data we are dealing with are too big to check, and even more often we do not know how such should look.

Moreover, in higher dimensions, it is complicated to find outliers. Some methods try to find outliers, but such methods are only partially efficient. Let us note that robust models 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 have some terminology to describe certain types of outliers. We use terminology from [1]. Let us have observation $(y_i, \boldsymbol{x_i})$. If observation is not outlying in any direction we call it *regular observation* . If it is outlying in $\boldsymbol{x_i}$ direction 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 liner pattern we call it *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 y_i direction, we call it a *vertical outliers.*

1.3.2 Measuring robustness

There are a couple of tools to measure the robustness of statistics. The most popular one is called *breakdown point.* Then there are *empirical influence function* and *influence function and sensitivity curve.* For the sake of simplicity, we describe only breakdown point right now.

Definition 8. Let T be a statistics, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ be an n-dimensional random sample and $T_n(\boldsymbol{x})$ value of this statistics with parameter $\boldsymbol{x}$. The breakdown point of T at sample $\boldsymbol{x}$ can be defined using sample $\overline{\boldsymbol{x}}^{(k)}$ where we exchange k points from original sample $\boldsymbol{x}$ with random values x_i. We get $T_n(\overline{\boldsymbol{x}}^{(k)})$. Then the *breakdown point* is

$$\mathrm{bdpoint}(T, \boldsymbol{x_n}) = \frac{1}{n} \min S_{T,\boldsymbol{x_n}}, \tag{1.19}$$

where

$$S_{T,\boldsymbol{x_n},D} = \left\{ k \in \{1, 2, \ldots, n\} : \sup_{\boldsymbol{x_{new}}^{(k)}} \left\| T_n(\boldsymbol{x}) - T_n(\boldsymbol{x_{new}}^{(k)}) \right\| = \infty \right\}. \tag{1.20}$$

This definition is very general but let us specify it to our linear regression problem. It says that breakdown point is the function of the minimal number of observations needed to be changed for some others so then the estimator gives incorrect results. More robust estimators have higher breakdown point.

It is intuitive that reasonable breakdown point should not be higher than 0.5 [2] because if more than 50% of the data is exchanged, then the model of exchanged data overrides the model of the original data.

In the context of OLS estimator one outlier is enough to increase the value of T to any desired value [3] thus

$$\mathrm{bdpoint}(OLS, \boldsymbol{x_n}) = \frac{1}{n}. \tag{1.21}$$

Figure [] gives us an example of how one outlier may change the hyperplane given by the OLS estimation of regression coefficients.

change of regression hyperplane based on one exchanged observation.

For an increasing number of data samples n this tends to zero. We can see that ordinary least squares estimator is not resistant to outliers at all. Due to this fact, multiple estimators similar to OLS have been proposed.

1.4 Least trimmed squares

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

Definition 9. 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, $n/2 \le h \le n$. The objective function of LTS is then denoted as

$$\mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}) = \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) \tag{1.22}$$

where $r_{i:n}^2$ denotes ith smallest squared residuum out of n thus $r_{1:n}^2 \le r_{2:n}^2 \dots r_{n:n}^2$ Even though that objective function of LTS seems similar to the OLS objective function, finding minimum is far more complex because the order of the least squared residuals is dependent on $\boldsymbol{w}$. Moreover $r_{i:n}^2(\boldsymbol{w})$ residuum is not uniquely determined if more squared residuals have same value as $r_{i:n}^2(\boldsymbol{w})$. This makes from finding LTS estimate non-convex optimization problem and finding global minimum is NP-hard [4].

1.4.1 Discrete objective function

The LTS objective function from definition 9 is not differentiable and not convex, so we are unable to use the same approach as with OLS objective function. Let us transform this objective function to a discrete version which can appropriately be used by algorithms to minimize it.

Let us assume for now that we know the $\boldsymbol{\hat{w}}^{(\boldsymbol{LTS,h,n})}$ vector of estimated regression coefficients. (LTS, h, n) denotes that coefficients are estimated using LTS on the h subset of n data samples. With this in mind, let π be the permutation of $\hat{n} = \{1, 2, \ldots, n\}$ such that

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

Put

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

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

$$\boldsymbol{\hat{w}}^{(\boldsymbol{LTS,h,n})} = \underset{\sum_{i=1}^{h} r^2_{i:n}(\boldsymbol{w})}{\arg\min} = \underset{\sum_{i=1}^{n} m_i^{(LTS)} r^2_{i}(\boldsymbol{w})}{\arg\min} . \tag{1.25}$$

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

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

In other words, finding the minimum of the LTS objective function can be done by finding the OLS estimate (1.26) for all vectors $\boldsymbol{m} \in Q^{(n,h)}$. Thus if we denote $\boldsymbol{X}_{\boldsymbol{M}} = \boldsymbol{M}\boldsymbol{X}$ and $\boldsymbol{y}_{\boldsymbol{M}} = \boldsymbol{M}\boldsymbol{y}$ which corresponds to h subsets of $\boldsymbol{X}$ and $\boldsymbol{Y}$, then [5]

cite klouda

$$
\begin{aligned}
\min_{\boldsymbol{w}\in\mathbb{R}^p} \mathrm{OF}^{(LTS,h,n)}(\boldsymbol{w}) &= \min_{\boldsymbol{w}\in\mathbb{R}^p} \sum_{i=1}^{h} r_{i:n}^2(\boldsymbol{w}) \\
&= \min_{\boldsymbol{w}\in\mathbb{R}^p, \boldsymbol{m}\in Q^{(n,h)}} \sum_{i=1}^{n} m_i r_i^2(\boldsymbol{w}) \\
&= \min_{\boldsymbol{m}\in Q^{(n,h)}} \left(\min_{\boldsymbol{w}\in\mathbb{R}^p} \mathrm{OF}^{(OLS,\boldsymbol{MX},\boldsymbol{My})}(\boldsymbol{w}) \right) \\
&= \min_{\boldsymbol{m}\in Q^{(n,h)}} \left(\min_{\boldsymbol{w}\in\mathbb{R}^p} \|\boldsymbol{My} - \boldsymbol{MXw}\|^2 \right).
\end{aligned}
$$

Substituting $\boldsymbol{w}$ with (1.25) we get

$$
\begin{aligned}
\hat{\boldsymbol{w}}^{(\boldsymbol{LTS},\boldsymbol{h},\boldsymbol{n})} &= \min_{\boldsymbol{m}\in Q^{(n,h)}} \left(\left\| \boldsymbol{My} - \boldsymbol{MX}(\boldsymbol{X}^T\boldsymbol{MX})^{-1}\boldsymbol{X}^T\boldsymbol{My} \right\|^2 \right) \\
&= \min_{\boldsymbol{m}\in Q^{(n,h)}} \left(\left\| \boldsymbol{My} - \boldsymbol{MX}(\boldsymbol{X}^T\boldsymbol{MX})^{-1}\boldsymbol{X}^T\boldsymbol{My} \right\|^2 \right)
\end{aligned}
$$

We can easily see, that have the objective function with argument $\boldsymbol{m} \in Q^{(n,h)}$. This objective function we mark as

$$
\mathrm{OF}_{\mathrm{D}}(\boldsymbol{m}) = \left\| \boldsymbol{My} - \boldsymbol{MX}(\boldsymbol{X}^T\boldsymbol{MX})^{-1}\boldsymbol{X}^T\boldsymbol{My} \right\|^2. \tag{1.27}
$$

We can also clearly see that the minimizing this OF could by done straightforwardly by iterating over the $Q^{(n,h)}$ set. That means we transformed the objective function to the discrete space of $Q^{(n,h)}$.

Unfortunately, this set is vast, namely $\binom{n}{h}$, so this approach is infeasible on more massive 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 $h = [(n/2] + [(p+1)/2]$ [3](where [.] denotes largest integer function). That 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 cover the theoretical background required to implement algorithms that are in this chapter. We introduced the discrete version of the LTS objective function which minimum is equivalent to the continuous one. Finding the minimum of this function requires to find the correct h subset and then calculate the estimate of regression coefficients $\boldsymbol{w}$. To achieve this, we need to examine all h subsets. The exhaustive approach fails due to the exponential size of $Q^{(n,h)}$. So what are the possibilities?

2.0.1 First attempts

Initial algorithms were based on iterative removal data samples whose residuum had the highest value based on OLS fit on the whole dataset. Such attempts are known to be flawed [6] because the initial OLS fit can be already profoundly affected by outliers and we may remove data samples which represent the actual model.

Other algorithms were based purely on a random approach. On such algorithm is Random solution algorithm [7] which randomly selects l of h subsets and subsequently compute OLS fit on each of them and chooses fit with a minimum value of the objective function. Such approach is straightforward, but the probability of selecting at least one subset from l subsets which don't contain outliers thus has a chance of producing good result tends to zero for an increasing number of data samples n as we'll describe in detail in section 2.2.2.

Another very similar algorithm called Resampling algorithm introduced in [8]. 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 l subsets of size $p+1$ thus it is independent of n gives the nonzero probability of selecting at least one subset such that it does not include 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 subsets is computationally exhaustive and relying on randomly generating "good" h-element subsets does not lead to reliable results. So what are our options? In [9] two criteria called *weak necessary conditions* and *strong necessary conditions* are introduced. They state necessary properties which some h subset must satisfy to be 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 subset of *non-trimmed* observations but also the complementary subset of not used observations. We'll refer to this complementary subset as to *trimmed* subset.

Definition 10. 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 an 1 with one 0, we get $\mathrm{OF_D}(\boldsymbol{m}) \leq \mathrm{OF_D}(\boldsymbol{m_{swap}})$. Thus the value of 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.

The second necessary condition is named *weak necessary condition*

Definition 11. A h-element subset corresponding to $\boldsymbol{m} \in Q^{(n,h)}$ satisfies the *strong necessary condition* if $r_i^2(\hat{\boldsymbol{w}}^{\boldsymbol{(OLS,MX,My)}})$ for all trimmed observation is greater of equal to the greatest non-trimmed squared residuum $r_j^2(\hat{\boldsymbol{w}}^{\boldsymbol{(OLS,MX,My)}})$.

Again, based on this criteria an algorithm can be created. The corollary of this criteria together with proof can be found in section 2.2.

The interesting consequence which we'll use later gives us the following lemma.

Lemma 12. The strong necessary condition is not satisfied unless the weak necessary condition is satisfied. Thus if a strong condition is satisfied then weak is also.

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

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

Thus

$$\mathrm{OF_D}(\boldsymbol{m}) > \mathrm{OF_D}(\boldsymbol{m}) + r_j^2 - r_i^2 \tag{2.1}$$

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

$$\mathrm{OF_D}(\boldsymbol{m}) + r_j^2 - r_i^2 \geq \mathrm{OF_D}(\boldsymbol{m_{swap}}). \tag{2.2}$$

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

Now we have covered all the necessary theoretical background, and it is time to introduce currently known algorithms for computing the LTS estimate.

2.1 Computing OLS

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

In the following algorithms, we describe its time complexity. Because 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 matrix multiplication.

There are multiple algorithms for matrix multiplication, for example:

- Naive algorithm; its time complexity is $\mathcal{O}(n^3)$.
- Strassen algorithm; its time complexity is $\mathcal{O}(n^{2.8074})$ [10].
- Coppersmith-Winograd; its time complexity is $\mathcal{O}(n^{2.375477})$[11]

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 case of the naive algorithm. For this reason, they are not used even when they might improve performance [12].

Moreover current implementations of the linear algebra namely LAPACK and more importantly BLAS level 3 routines implementing matrix-matrix multiplication which are widely used in such linear algebra software implement naive matrix-matrix multiplication [13].

For that reason we assume in this work that 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

Solving OLS using 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 inversion of $\boldsymbol{A}$.
3. Multiply $\boldsymbol{A}^{-1}\boldsymbol{B}$.

Observation 13. 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 matrix inversion is $O(p^2 n)$.

Proof. First, we compute $\boldsymbol{A} = \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$ and $\boldsymbol{B} = \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}$. That gives us $\mathcal{O}(p^2 n + pn)$. Next, we compute inversion $\boldsymbol{C} = \boldsymbol{A}^{-1}$ which gives us $\mathcal{O}(p^3)$. Finally, we compute $\hat{\boldsymbol{w}}^{(OLS,n)} = \boldsymbol{CB}$ which is $\mathcal{O}(p^2)$. Together 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 $n \geq p$ and usually even $n \gg p$ we get $\mathcal{O}(p^2 n + p^3) \sim \mathcal{O}(p^2 n)$. $\square$

Computing OLS estimate using matrix inversion is possible, however in most cases not used in practice because of the low numerical stability computing matrix inversion.

2.1.2 Computation using the Cholesky decomposition

Definition 14. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be an symmetric positive definite matrix. Then it is possible to find lower triangular matrix L so that

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

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

If we look at our problem of finding a solution to

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

we can easily rewrite it as

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

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

$$\boldsymbol{L}\boldsymbol{d} = \boldsymbol{b} \tag{2.7}$$

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

where $\boldsymbol{d}$ is solved by forward substitution and $\boldsymbol{w}$ is then solved by backward substitution which ten represents $\hat{\boldsymbol{w}}^{(OLS,n)}$ estimate.

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

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

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

3. Solve lower triangular system $\boldsymbol{L}\boldsymbol{d} = \boldsymbol{b}$ 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 15. 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 $\mathcal{O}(\frac{1}{3}p^3)$ [14] In the third and fourth step, we are solving triangular systems both of them require $\mathcal{O}(\frac{1}{2}p^2)$, so together $\mathcal{O}(p^2)$.

Putting all steps together we get

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

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

Computation OLS estimate using the Cholesky factorization is more numerically stable than using matrix inversion and time complexity is asymptotically similar. This approach is however not usually used in practice as well.

2.1.3 Computation using the QR decomposition

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

Definition 16. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be any square matrix. Then it is possible to find matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ so that

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}, \tag{2.10}$$

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

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

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

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

That means that $\boldsymbol{X}$ can be factorized as

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

Then

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

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

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

Because $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ is not square matrix then as we shown $\boldsymbol{R} = \begin{bmatrix}\boldsymbol{R_1}\\ \boldsymbol{0}\end{bmatrix}$ so that

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

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

Solution to

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

is then given by

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

and because we assume $\boldsymbol{X}$ have full column rank, thus is invertible, we can simplify it 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 triangular matrix, solution of $\boldsymbol{w}$ is trivial using backward substitution. $\boldsymbol{w}$ then represents our OLS estimate $\hat{\boldsymbol{w}}^{(OLS,n)}$.

Note 17. We can see that Cholesky factorization $\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$ is closely connected to the QR decomposition $\boldsymbol{X} = \boldsymbol{Q_1}\boldsymbol{R_1}$ so that

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

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

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

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

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

find citation

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

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

so Householder transformations are about 50% faster, but Givens rotations are more numerically stable. Moreover, Givens rotations and are suitable for sparse matrices. This property of Givens rotations we use in section . That is why we describe Givens rotation in detail in 2.5. We also make proof of (2.21) time complexity there. ref

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

Observation 18. The time complexity of solving OLS estimate using QR decomposition by Householder transformations is $\mathcal{O}(2p^2n - \frac{2}{3}p^3)$

Proof. In the first step, we calculate QR decomposition. Householder transformations can be done in $\mathcal{O}(2np^2 - \frac{2}{3}p^3)$. Second step consist of matrix multiplication $\boldsymbol{Q_1}^T\boldsymbol{y}$ which is $\mathcal{O}(np)$. In the last step we are solving upper triangular systems 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^2n asymptotically dominates. $\square$

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 OLS estimate exists, and that is 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.

2.2 FAST-LTS

In this section, we introduce the FAST-LTS algorithm[15]. It is, as well as other algorithms we introduce, 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 which leading to better results.

Theorem 19. Consider $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ and $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$. Let us also have $\boldsymbol{w_0} \in \mathbb{R}^p$ and $\boldsymbol{m_0} \in Q^{(n,h)}$. Let us put $L_0 = \sum\limits_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0})$ where $r_i^2(\boldsymbol{w_0}) = y_i - (w_1^0 x_1^i + w_2^0 x_2^i + \ldots + w_p^0 x_p^i)$. Let us take $\hat{n} = \{1, 2, \ldots, n\}$ and mark $\pi : \hat{n} \to \hat{n}$ permutation of $\hat{n}$ such that $|r_0(\pi(1))| \leq |r_0(\pi(2))| \leq \ldots \leq |r_0(\pi(n))|$ 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 subset with smallest absolute residuals $r_i^2(\boldsymbol{w_0})$.

Finally, take $\mathrm{OF}^{(OLS,\boldsymbol{M_1X},\boldsymbol{M_1y})}(\boldsymbol{w})$ least squares fit on m_1 subset of observations and its corresponding $L_1 = \sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1})$. Then

$$L_1 \leq L_0 \tag{2.23}$$

Proof. Because we take h observations with smallest absolute residuals $r_i^2(\boldsymbol{w_0})$, then for sure $\sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0}) \leq \sum\limits_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0}) = L_0$. When we take into account that the OLS estimate minimizes the objective function of $\boldsymbol{m_1}$ subset of observations, then for sure $L_1 = \sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1}) \leq \sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0})$.

Together we get $L_1 = \sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_1}) \leq \sum\limits_{i=1}^{n} m_i^{(1)} r_i^2(\boldsymbol{w_0}) \leq \sum\limits_{i=1}^{n} m_i^{(0)} r_i^2(\boldsymbol{w_0}) = L_0$ □

Corollary 20. Based on the previous theorem, using some $\hat{\boldsymbol{w}}_{\boldsymbol{old}}^{\boldsymbol{OLS(h,n)}}$ of $\boldsymbol{m_{old}}$ subset of observations we can construct $\boldsymbol{m_{new}}$ subset with corresponding $\hat{\boldsymbol{w}}_{\boldsymbol{new}}^{\boldsymbol{OLS(H_{new})}}$ 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 *C-step* of this process is described by the following pseudocode. Note that for C-step we need only $\hat{\boldsymbol{w}}$ without the need of passing $\boldsymbol{m}$.

Algorithm 1: 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: $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$, m_{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 on m_new subset; return ŵ_new , m_new;
```

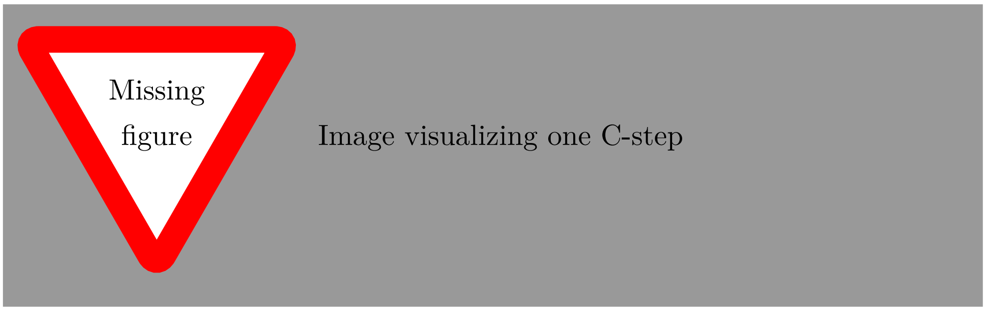

Figure 2.1: todo caption

One step of algorithm C-step is visualized on figure 2.1.

Observation 21. The time complexity of algorithm C-step 1 is asymptotically similar to the time complexity as OLS fit. Thus $O(p^2n)$.

Proof. In 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 $\mathcal{O}(p)$. So time of computation 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 algorithm QuickSelect [16]. Finally, we must compute $\hat{w}$ OLS estimate on a h subset of data. Because h is linearly dependent on n, we can say that it is $\mathcal{O}(p^2n + p^3)$ which is asymptotically dominant against previous steps which are $\mathcal{O}(np + n)$. Because we assume$n \geq p$ then $\mathcal{O}(p^2n + p^3) \sim \mathcal{O}(p^2n)$ $\square$

As we stated above, repeating C-step leads to sequence of $\hat{\boldsymbol{w}}_{\mathbf{1}}, \hat{\boldsymbol{w}}_{\mathbf{2}} \ldots$ on subsets $\boldsymbol{m}_{\mathbf{1}}, \boldsymbol{m}_{\mathbf{2}} \ldots$ with corresponding $L_1 \geq L_2 \geq \ldots$. 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 22. Sequence of C-step converges to $\hat{\boldsymbol{w}}_{\boldsymbol{k}}$ after maximum of $k = \binom{n}{h}$ so that

$$L_k = L_i \,, \forall i \geq k. \tag{2.24}$$

Proof. Since $\mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i}})$ is non-negative and $\mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) \leq \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i+i}})$ the sequence converges. $\hat{\boldsymbol{w}}_{\boldsymbol{i}}$ is computed out of subset $\boldsymbol{m}_{\boldsymbol{i}} \in Q^{(n,h)}$. Since $Q^{(n,h)}$ is finite, namely its size is $\binom{n}{h}$, the sequence converges at the latest after this number of steps. $\square$

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

Algorithm 2: 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: $\hat{\boldsymbol{w}}_{\boldsymbol{final}}$, $\boldsymbol{m}_{\boldsymbol{final}}$

$\hat{\boldsymbol{w}}_{\boldsymbol{new}} \leftarrow \emptyset$;
$\boldsymbol{m}_{\boldsymbol{new}} \leftarrow \emptyset$;
$L_{new} \leftarrow \infty$;
while *True* **do**
  $L_{old} \leftarrow \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{old}})$;
  $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$, $H_{new} \leftarrow$ C-step($\boldsymbol{X}$, $\boldsymbol{y}$, $\hat{\boldsymbol{w}}_{\boldsymbol{old}}$);
  $L_{new} \leftarrow \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{new}})$;
  if *Lold* $==$ *Lnew* **then**
    break
  end
  $\hat{\boldsymbol{w}}_{\boldsymbol{old}} \leftarrow \hat{\boldsymbol{w}}_{\boldsymbol{new}}$
end
return $\hat{\boldsymbol{w}}_{\boldsymbol{new}}$, $\boldsymbol{m}_{\boldsymbol{new}}$;

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

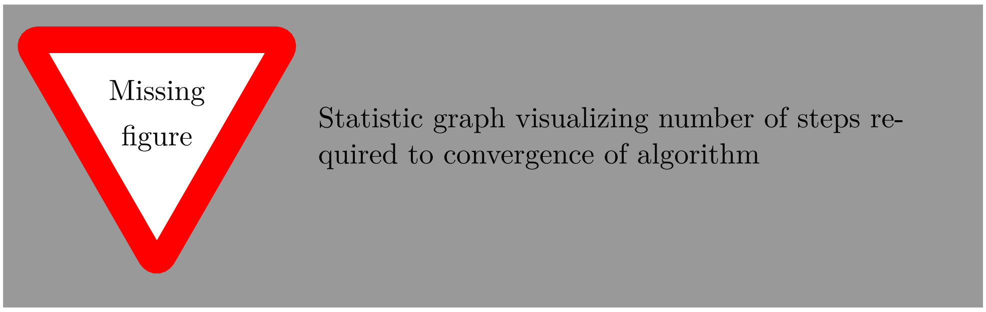

Figure 2.2: todo caption

That is not enough for the algorithm Repeat-C-step to converge to the global minimum. That gives us an idea of how to create the final algorithm. [15]

Choose a lot of initial subsets $\boldsymbol{m_1}$ and on each of them apply algorithm Repeat-C-step. From all converged subsets with corresponding $\hat{\boldsymbol{w}}$ estimates choose the one with the lowest value of $\mathrm{OF_D}(\hat{w})$.

Before we can construct final the algorithm, we must decide how to choose initial subset $\boldsymbol{m_1}$ and how many of them mean "*a lot*". First, let us focus on how to choose an initial subset $\boldsymbol{m_1}$.

2.2.2 Choosing initial $\boldsymbol{m_1}$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 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 $\boldsymbol{m_1}$ 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 23. With an increasing number of data samples, thus with increasing n, the probability of choosing among k random selections of $\boldsymbol{m_{1_1}}, \ldots, \boldsymbol{m_{1_k}}$ the probability of selecting at least one $\boldsymbol{m_{1_i}}$ such that its corresponding data samples does not contains outliers, goes to 0.

Proof. Consider dataset of n containing $\epsilon > 0$ relative amount of outliers. Let h be chosen conservatively so that $h = [(n/2] + [(p+1)/2]$ and k is the number of selections random h subsets. Then

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

Because $n \to \infty$, then $(1-\epsilon)^h \to 0$, then $1-(1-\epsilon)^h \to 1$, then $(1-(1-\epsilon)^h)^m \to 1$, then $1-(1-(1-\epsilon)^h)^m \to 0$ ☐

That means that we should consider other options for selecting $\boldsymbol{m_1}$ subset. If we would like to continue with selecting some random subsets, previous observation gives us a clue, that we should choose it independent of n. Authors of the algorithm came with such a solution, and it goes as follows.

P-subset selection

Let us choose vector $\boldsymbol{c} \in Q^{(n,p)}$. Next we compute rank of matrix $\boldsymbol{X}_C = \boldsymbol{CX}$, where $C = diag(c)$. If $rank(\boldsymbol{X}_C) < p$ add randomly selected rows to $\boldsymbol{X}_C$ without repetition until $rank(\boldsymbol{X}_C) = p$. Let us from now on suppose that $rank(\boldsymbol{X}_C) = p$. Next let us mark $\boldsymbol{\hat{w}_0} = \mathrm{OF_D}(c)$ and corresponding $r_1(\boldsymbol{\hat{w}_0}), r_2(\boldsymbol{\hat{w}_0}), \ldots, r_n(\boldsymbol{\hat{w}_0})$ residuals. Now mark $\hat{n} = \{1, 2, \ldots, n\}$ and let $\pi : \hat{n} \to \hat{n}$ be permutation of $\hat{n}$ such that $|r(\pi(1))| \leq |r(\pi(2))| \leq \ldots \leq |r(\pi(n))|$.

Finally, 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.

Observation 24. With the increasing number of data samples, thus with increasing n, the probability of choosing among k random selections of $\boldsymbol{c_{1_1}}, \ldots, \boldsymbol{c_{1_k}}$ the probability of selecting at least one $\boldsymbol{c_{1_i}}$ such that its corresponding data samples do not contains outliers, goes to

$$1 - (1 - (1 - \epsilon)^h)^k > 0. \tag{2.25}$$

Proof. Similarly as in previous observation. □

Last missing piece of the algorithm is determining the number of k initial $\boldsymbol{m_1}$ subsets, which maximize the probability to at least one of them converges to correct solution. 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 OLS fit on $\boldsymbol{m_i}$ subset and then the calculation of n absolute residuals. How 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.

Due to that convergence of repeating C-step is very fast, it turns out, that we can distinguish between starts that leads to good solutions and those which does not, even after a small amount of the 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 $\mathrm{OF_D}(\boldsymbol{\hat{w}_3})$ or $oflts(\boldsymbol{\hat{w}_4})$ respectively.

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 iteration of the C-steps till convergence only on those best solutions. Authors refer to this process as to *selective iteration*.

Nested extension

C-step computation is usually very fast for small n. Problem starts with very high n say $n > 10^3$ because we need to compute OLS on $\boldsymbol{m_i}$ subset of size h which is dependent on n. And then calculate n absolute residuals.

Authors came up with a solution they call *nested extension*. We describe it briefly now.

- If n is greater than limit l, we create subset of data samples $L\,, |L| = l$ and divide this subset into s disjunctive 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 number of starts $m_{P_i} = \frac{m}{l}$.

- Next in every P_i we create m_{P_i} number of initial $H_{P_{i_1}}$ subsets and iterate C-steps for two iterations.

- Then we choose 10 best results from each subsets and merge them together. We get family of sets F_{merged} containing 10 best $H_{P_{i_3}}$ subsets from each P_i.

- On each subset from F_{merged} family of subsets we again iterate 2 C-steps and then choose 10 best results.

- Finally we use these best 10 subsets and use them to iterate C-steps till convergence.

- As a result we choose best of those 10 converged results.

2.2.4 Putting all together

We 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 t 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 final C-step iteration till convergence. Parameter t denotes threshold stopping criterion such that $|\,\mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) - \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})| \leq t$ instead of $\mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i}}) = \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i+1}})$. When we put all together, we get *FAST-LTS* algorithm which is described by the following pseudocode.

Algorithm 3: FAST-LTS

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

$\hat{\boldsymbol{w}}_{\boldsymbol{final}} \leftarrow \emptyset$;
$\boldsymbol{m}_{\boldsymbol{final}} \leftarrow \emptyset$;
$F_{best} \leftarrow \emptyset$;
if $n \geq l$ **then**
    $F_{merged} \leftarrow \emptyset$;
    for $i \leftarrow 0$ ***to*** s **do**
        $F_{selected} \leftarrow \emptyset$;
        for $j \leftarrow 0$ ***to*** $\frac{l}{s}$ **do**
            $F_{initial} \leftarrow Selective\ iteration(\frac{m}{l})$;
            for $\boldsymbol{m_i}$ ***in*** $F_{initial}$ **do**
                $\boldsymbol{m_i} \leftarrow Iterate\ C\ step\ few\ times(\boldsymbol{m_i})$;
                $F_{selected} \leftarrow F_{selected} \cup \{\boldsymbol{m_i}\}$;
            end
        end
        $F_{merged} \leftarrow F_{merged} \cup Select\ 10\ best\ subsets\ from\ F_{selected}$;
    end
    for $\boldsymbol{m_i}$ ***in*** F_{merged} **do**
        $\boldsymbol{m_i} \leftarrow Iterate\ C\ step\ few\ times(\boldsymbol{m_i})$;
        $F_{best} \leftarrow F_{best} \cup \{\boldsymbol{m_i}\}$;
    end
    $F_{best} \leftarrow Select\ 10\ best\ subsets\ from\ F_{best}$;
else
    $F_{initial} \leftarrow Selective\ iteration(m)$;
    $F_{best} \leftarrow Select\ 10\ best\ subsets\ from\ F_{initial}$;
end
$F_{final} \leftarrow \emptyset$;
$W_{final} \leftarrow \emptyset$;
for $\boldsymbol{m_i}$ ***in*** F_{best} **do**
    $\boldsymbol{m_i}, \hat{\boldsymbol{w}}_{\boldsymbol{i}} \leftarrow Iterate\ C\ step\ till\ convergence(\boldsymbol{m_i}, i_{max}, t)$;
    $F_{final} \leftarrow F_{final} \cup \{\boldsymbol{m_i}\}$;
    $W_{final} \leftarrow W_{final} \cup \{\hat{\boldsymbol{w}}_{\boldsymbol{i}}\}$;
end
$\hat{\boldsymbol{w}}_{\boldsymbol{final}}, \boldsymbol{m}_{\boldsymbol{final}} \leftarrow$
    $select\ \hat{\boldsymbol{w}}_{\boldsymbol{i}}\ and\ \boldsymbol{m_i}\ with\ smallest\ \mathrm{OF_D}(\hat{\boldsymbol{w}}_{\boldsymbol{i}})\ from\ F_{final}$;
return $\hat{\boldsymbol{w}}_{\boldsymbol{final}}, \boldsymbol{m}_{\boldsymbol{final}}$;

2.3 Feasible solution

In this section we'll introduce feasible solution algorithm from [6]. It is based on strong necessary condition we've described at 10. The basic idea can be described as follows.

Let's consider that we have some $\boldsymbol{m} \in Q^{(n,h)}$. It'll be convenient when we'll mark in the following text $O_m = \{i \in \{1, 2, \dots, n\}; w_i = 1\}$ and $Z_m = \{j \in \{1, 2, \dots, n\}; w_j = 0\}$ thus sets of indexes of positions where is 0 respectively 1 in vector $\boldsymbol{m}$. We can think about it as indexes of observations in h set and g set respectively. Then we can mark $\boldsymbol{m}^{(i,j)}$ as a vector which is constructed by swap of its ith and jth element where $i \in O$ and $j \in Z$. Such vector correspond to vector $\boldsymbol{m_{swap}}$ which we marked at 10.

With this in mind we can mark let's mark

$$\Delta S_{i,j}^{(\boldsymbol{m})} = \mathrm{OF}_{\mathrm{D}}(\boldsymbol{m}^{(i,j)}) - \mathrm{OF}_{\mathrm{D}}(\boldsymbol{m}) \tag{2.26}$$

thus change of the LTS objective function by swapping one observation from h subset with another from g subset. To calculate this we can obviously first calculate $\mathrm{OF}_{\mathrm{D}}(\boldsymbol{m})$ and $\mathrm{OF}_{\mathrm{D}}(\boldsymbol{m}^{(i,j)})$ and finally subtract both results. Although it is a option, it it computationally hard. So the question is if there is easier way of calculating $\Delta S_{i,j}^{(\boldsymbol{m})}$ and the answer is positive.

Let's mark $M = diag(\boldsymbol{m}$ and $M^(i,j) = diag(\boldsymbol{m}^{(i,j)})$ and also $\boldsymbol{H} = (\boldsymbol{X}^T \boldsymbol{M}^T \boldsymbol{X})$ For now let's assume that we already calculated $\boldsymbol{H}^{-1}$ and also $\hat{\boldsymbol{w}} = \boldsymbol{H}^{-1}\boldsymbol{X}^T\boldsymbol{M}\boldsymbol{y}$ we now want to calculate $\Delta S_{i,j}^{(\boldsymbol{m})}$ Let's mark vector of residuals $\boldsymbol{r}^{(m)} = \boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{w}}$ and also $d_{r,s} = \boldsymbol{x_r}\boldsymbol{H}^{-1}\boldsymbol{x_s}$ then by equation introduced in [17] we get

$$\Delta S_{i,j}^{(\boldsymbol{m})} = \frac{(r_j^{(m)})^2(1 - d_{i,i}) - (r_i^{(m)})^2(1 + d_{j,j}) + 2r_i^{(m)} r_j^{(m)} d_{j,j}}{(1 - d_{i,i})(1 + d_{j,j}) + d_{i,j}^2}. \tag{2.27}$$

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

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

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

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

1. use the first swap that leads to the improvement (so don't event try to find different swap)
2. from all possible swaps choose one that has highest improvement value $\mathrm{OF}_{\mathrm{D}}(\boldsymbol{m}^{(i,j)})$
3. use the first swap that has improvement higher than some given threshold

O(n2p) or O(n3p) remains

In the terms of complexity all three options give the same results , because to find feasible solution you need to evaluate all pair swaps. In practice third options is winner because it'll lead to least amount of iterations. On the other hand as we said this won't improve complexity of the algorithm, so from now on let's assume that we'll use case number two. So if there positive $S_{i,j}^{(\boldsymbol{m})}$ we'll make the swap and repeat the process again. Algorithm ends when there is no possible improvement i.e. when all $S_{i,j}^{(\boldsymbol{m})}$ are non-negative. Number of iterations needed till algorithm stops is usually quite low, but for practical usage it's still convenient to use some parameter *max_iteration* after which algorithm will stop without finding h subset satisfying strong necessary condition.

put sem part of the pseudocode describing core iteration?

and its time compleixty

One run of this iteration process will lead to some local optima i.e. set satisfying strong necessary condition. In [6] they refer to to this set as to *feasible set*. This because it is not global optima the algorithm needs to be run multiple times say N times. As a final solution is taken such h subset with the smallest residual sum of squares.

We didn't yet mention how to create initial h subset respectively initial $\boldsymbol{m}$. We already had this discussion when describing FAST-LTS algorithm. The [6] describes only random starting h subsets, but using p subsets instead may lead to improvement . More importantly as we already suggested h subset satisfying weak necessary condition don't need to satisfy strong necessary condition so passing such a h subset as input to this algorithm is another option and we'll discuss it in detail later. For that reasons we'll now describe feasible algorithm with pseudocode and we'll assume that we already have

experiment with this and refer here

some function that generates for us h subsets e.g. random one.

Algorithm 4: Feasible solution

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

$\hat{\boldsymbol{w}}_{\boldsymbol{final}} \leftarrow \emptyset$;
$H_{final} \leftarrow \emptyset$;
$Results \leftarrow \emptyset$;
for $k \leftarrow 0$ ***to*** N **do**
    $\boldsymbol{m} \leftarrow generate\_intial\_subset()$; // e.g. random $\boldsymbol{m} \in Q^{(n,h)}$
    while $True$ **do**
        $best_i \leftarrow 0$;
        $best_j \leftarrow 0$;
        $best_{delta} \leftarrow 0$;
        for $i \in O_m$ **do**
            for $j \in Z_m$ **do**
                $delta \leftarrow calculate\Delta S_{i,j}^{(\boldsymbol{m})}$;
                if $delta > best_{delta}$ **then**
                    $best_{delta} \leftarrow delta$;
                    $best_i \leftarrow i$;
                    $best_j \leftarrow j$;
                end
            end
        end
        if $best_{delta} > 0$ **then**
            $\boldsymbol{m} \leftarrow \boldsymbol{m}^{(i,j)}$;
        end
        else
            $H \leftarrow h\ subset\ corresponding\ to\ \boldsymbol{m}$;
            $\boldsymbol{M} \leftarrow diag(\boldsymbol{m})$;
            $\hat{w} \leftarrow \mathrm{OF}^{(OLS, \boldsymbol{MX}, \boldsymbol{My})}$;
            $Results \leftarrow Results \cup \{H, \hat{w}\}$;
            break;
        end
    end
end
$H_{final}, \hat{\boldsymbol{w}}_{\boldsymbol{final}} \leftarrow$ select best from $Results$ based on smallest RSS;
return $\hat{\boldsymbol{w}}_{\boldsymbol{final}}$, H_{final};

time complexity

2.4 Combined algorithms

Here we'll describe what we've indicated above and that is combination of

todo this section and move it somewhere else qwerty both previous algorithms. Two options. z

1. Let fast-lts converge and then run FSA and let it converge
2. Let fast-lts converge make one step of FSA and try fast-LTS again and iterate between this

Second option would be faster ? Let's think about it.. etc etc..

2.5 Improved FSA

So far as we've described FSA algorithm we assumed that after each cycle of the algorithm (after each best swap) wee need to recalculate inversion $(\boldsymbol{X}^T\boldsymbol{X})$ together with $\hat{\boldsymbol{w}}$. In this section we'll introduce different approaches described in [3] which will improve it so that we'll be able to update it instead of recalculate. Moreover these ideas will lead us to bounding condition of FSA which will improve the speed and we'll also be able to construct another algorithms based on this idea.

We've introduced additive formula 2.27. Let's now try to obtain similar formula but let's try to focus on how individual elements in our current algorithm will change not only on the swap of two rows but how those elements will be affected after adding one row and also removing one row. Moreover during this derivation we'll be able to obtain two different approaches of calculating one thing. Both are important and we'll recapitulate them after.

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

$$\tilde{\boldsymbol{Z}} = \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y} \end{bmatrix}. \tag{2.28}$$

Notice that Z is symmetric square matrix $\in \mathbb{R}^{p\times p}$ moreover we suppose that Z is regular. $\boldsymbol{X}^T\boldsymbol{y}$ is column vector $\in \mathbb{R}^{1\times p}$, $\boldsymbol{y}^T\boldsymbol{X}$ is vector $\in \mathbb{R}^{p\times 1}$ and $\boldsymbol{y}^T\boldsymbol{y}$ is scalar. OLS estimate $\hat{\boldsymbol{w}}$ is then

$$\hat{\boldsymbol{w}}^{(\boldsymbol{OLS,n})} = \boldsymbol{Z}^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2.29}$$

and residual sum of squares

$$RSS = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}} \tag{2.30}$$

Also note that all necessary multiplications are already in the matrix $\tilde{\boldsymbol{Z}}$. Let's now realize that RSS can be expressed as ratio of determinants $\det(\tilde{\boldsymbol{Z}})$ and $\det(\boldsymbol{Z})$ (using determinant rule for block matrices) so that

$$RSS = \frac{\det(\tilde{\boldsymbol{Z}})}{\det(\boldsymbol{Z})} \tag{2.31}$$
$$= \frac{\det\begin{pmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y}\end{pmatrix}}{\det(\boldsymbol{M})}$$
$$= \frac{\det(\boldsymbol{Z})\det\left(\boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{Z}^{-1}\boldsymbol{y}\right)}{\det(\boldsymbol{Z})}$$
$$= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}}.$$

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

$$\tilde{\boldsymbol{Z}}^{-1} = \begin{bmatrix} \boldsymbol{Z}^{-1} + \frac{\hat{\boldsymbol{w}}\hat{\boldsymbol{w}}^T}{RSS} & -\frac{\hat{\boldsymbol{w}}^T}{RSS} \\ -\frac{\hat{\boldsymbol{w}}}{RSS} & \frac{1}{RSS} \end{bmatrix}. \tag{2.32}$$

For following equations it's important to notice that for any two observations $z_i = (x_i, y_i)$ and $\boldsymbol{z_j} = (\boldsymbol{x_j}, y_j)$ $\boldsymbol{z_i}, z_i \in \mathbb{R}^{p+1\times 1}$ is

$$\boldsymbol{z_i}\tilde{\boldsymbol{Z}}^{-1}\boldsymbol{z_i^T} = \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^2}{RSS} + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} \tag{2.33}$$

and

$$\boldsymbol{z_j}\tilde{\boldsymbol{Z}}^{-1}\boldsymbol{z^T} = \frac{(y_j - \boldsymbol{x_j}\hat{\boldsymbol{w}})(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})}{RSS} + \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} \tag{2.34}$$

Using above let's express how the determinant $\det(\boldsymbol{Z})$ and inverse of $vecZ^{-1}$ will change when we'll add one observation $\boldsymbol{z_i} = (\boldsymbol{x_i}, y_i) \in \mathbb{R}^{p+1\times 1}$ to the matrix $\boldsymbol{A}$. First lets notice that if we add this row to $\boldsymbol{A}$, then $\boldsymbol{Z}$ will change

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} & x_{1i} \\ x_{21} & x_{22} & \dots & x_{2n} & x_{2i} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \dots & x_{pn} & x_{pi} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \\ x_{i1} & x_{i2} & \dots & x_{ip} \end{bmatrix} = \boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{x_i^T}\boldsymbol{x_i} = \boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i}, \tag{2.35}$$

so determinant with appended row will be

$$\det(\boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i}) = det(\boldsymbol{Z})(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}) \tag{2.36}$$

and inversion can be obtained using Sherman-Morrison formula [18] so that

$$(\boldsymbol{Z} + \boldsymbol{x_i^T}\boldsymbol{x_i})^{-1} = \boldsymbol{Z}^{-1} - \frac{\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_i}\boldsymbol{Z}^{-1}}{1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}} \tag{2.37}$$

For following equations it'll be convenient for us to mark

$$b = \frac{-1}{(1 + \boldsymbol{x_i} \boldsymbol{Z}^{-1} \boldsymbol{x_i^T})}, b \in \mathbb{R}, \tag{2.38}$$

and

$$\boldsymbol{u} = \boldsymbol{Z}^{-1} \boldsymbol{x_i^T}, \boldsymbol{u} \in \mathbb{R}^{p \times 1}, \tag{2.39}$$

so that 2.37 can be written as

$$(\boldsymbol{Z} + \boldsymbol{x_i^T} \boldsymbol{x_i})^{-1} = \boldsymbol{Z}^{-1} + b\boldsymbol{u}\boldsymbol{u^T}. \tag{2.40}$$

Given that last piece that we're missing to express updated $\hat{\boldsymbol{w}}$ is appended $\boldsymbol{X}^T\boldsymbol{y}$ by one row, which can be simply expressed by same idea as 2.35 so that updated $\hat{\boldsymbol{w}}$ which we'll denote as $\overline{\hat{\boldsymbol{w}}}$ is

$$\overline{\hat{\boldsymbol{w}}} = (\boldsymbol{Z}^{-1} + b\boldsymbol{u}\boldsymbol{u^T})(\boldsymbol{X^T}\boldsymbol{y} + y_i \boldsymbol{x_i^T}). \tag{2.41}$$

mistake in original paper where is w + (y - xw)bu

This can be simplified so that we get

$$\overline{\hat{\boldsymbol{w}}} = \hat{\boldsymbol{w}} - (y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})b\boldsymbol{u}. \tag{2.42}$$

Last but not least we want to express updated RSS which we denote as $\overline{RSS}$. This can be done easily from 2.31 and 2.36 as

$$\overline{RSS} = RSS + \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^{\boldsymbol{2}}}{(1 + \boldsymbol{x_i} \boldsymbol{Z}^{-1} \boldsymbol{x_i^T})} \tag{2.43}$$

and again, it's convenient to mark

$$\gamma^+(\boldsymbol{z_i}) = \frac{(y_i - \boldsymbol{x_i}\hat{\boldsymbol{w}})^{\boldsymbol{2}}}{(1 + \boldsymbol{x_i} \boldsymbol{Z}^{-1} \boldsymbol{x_i^T})}, \tag{2.44}$$

so that

$$\overline{RSS} = RSS + \gamma^+(\boldsymbol{z_i}) \tag{2.45}$$

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

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

Consider that we've already included one observation z_i in our dataset and mark $\overline{\boldsymbol{Z}} = \boldsymbol{Z} + \boldsymbol{x_i}\boldsymbol{x_i^T}$ If we exclude one observation $\boldsymbol{z_j} = (\boldsymbol{x_j}, y_j) \in \mathbb{R}^{p+1\times 1}$ from already updated matrix $\boldsymbol{A}$ then determinant $\det(\overline{\boldsymbol{Z}})$ will change as

$$\det(\overline{\boldsymbol{Z}} - \boldsymbol{x_j^T}\boldsymbol{x_j}) = det(\overline{\boldsymbol{Z}})(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}) \tag{2.46}$$

and inversion will change (again according to Sherman-Morrison formula) as

$$(\overline{\boldsymbol{Z}} - \boldsymbol{x_j^T}\boldsymbol{x_j})^{-1} = \overline{\boldsymbol{Z}}^{-1} + \frac{\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}}{1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}}. \tag{2.47}$$

Once again, it's convenient to denote

$$\bar{b} = \frac{-1}{(1\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})}, \bar{v} \in \mathbb{R}, \tag{2.48}$$

and

$$\overline{\boldsymbol{u}} = \overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}, \overline{\boldsymbol{u}} \in \mathbb{R}^{p\times 1}, \tag{2.49}$$

so that we can write

$$(\overline{\boldsymbol{Z}} + \boldsymbol{x_j^T x_j})^{-1} = \boldsymbol{Z}^{-1} - \bar{b}\overline{\boldsymbol{u}}\overline{\boldsymbol{u}}^{\boldsymbol{T}}. \tag{2.50}$$

Using same approach as before we can denote express downdated estimate which we denote as $\overline{\overline{\hat{\boldsymbol{w}}}}$

$$\overline{\overline{\hat{\boldsymbol{w}}}} = +\overline{\hat{\boldsymbol{w}}}(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})\bar{b}\overline{\boldsymbol{u}}. \tag{2.51}$$

Finally lets also express updated $\overline{RSS}$ which we denote as $\overline{\overline{RSS}}$. This can be done easily from 2.31 and 2.36 and 2.47 as

$$\overline{\overline{RSS}} = \overline{RSS} - \frac{(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})^{\boldsymbol{2}}}{(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})} \tag{2.52}$$

and let's also mark

$$\gamma^{-}(\boldsymbol{z_j}) = \frac{(y_j - \boldsymbol{x_j}\overline{\hat{\boldsymbol{w}}})^{\boldsymbol{2}}}{(1 - \boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T})}, \tag{2.53}$$

so that

$$\overline{\overline{RSS}} = \overline{RSS} - \gamma^{-}(\boldsymbol{z_j}) \tag{2.54}$$

Lets now express equation for including and removing observation at once. First let's notice that from 2.47 we can express

$$\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T}) = \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}) - \frac{(\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2}{1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T})}, \tag{2.55}$$

so that

$$\det(\boldsymbol{Z} + \boldsymbol{x_i^T x_i} - \boldsymbol{x_j^T x_j}) = \tag{2.56}$$
$$\det(\boldsymbol{Z})(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} + (\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2 - \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})$$

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

$$\overline{\overline{RSS}} = RSS\rho(z_i, z_j), \tag{2.57}$$

where

$$\rho(z_i, z_j) = \frac{(1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} - \frac{e_i^2}{RSS}) + (\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} + \frac{e_i e_j}{RSS})^2}{1 + \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} - \boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T} + (\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T})^2 - \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}}, \tag{2.58}$$

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

We can see that this formula is similar to the one 2.27 but here the $\rho(z_i, z_j)$ represents multiplicative increment. Moreover $0 < \rho(z_i, z_j)$ and if $0 < \rho(z_i, z_j) < 1$ then the swap leads to improvement in terms of feasible solution.

With all this we are now able to change FSA so that we don't need to recompute $\hat{\boldsymbol{w}}$ and inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ but we can update it. That means after we find z_i, z_j that improve the current RSS the most (that means we find smallest $\rho(z_i, z_j)$ by 2.58). We can then update RSS by 2.57, update $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ by 2.40 and 2.47 and finally update $\hat{\boldsymbol{w}}$ by 2.42 and 2.51.

pseudokod

Let's now talk about time complexity of such solution. We can clearly see that core iteration takes the most time. So let's first describe time complexity of finding optimal swap and updating result with such pair. We need to go through all pairs between z_i, z_j where z_i is from h-subset and z_j is from g-subset. That is

$$h(n-h) \approx (\frac{n^2 - p^2}{4})^2 \sim \mathcal{O}(n^2 - p^2) \tag{2.59}$$

For each of this pair we need to calculate 2.27. Note that e_i and e_j can be calculated before this loop. The same goes for $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}$ and $\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$. So inside the loop there is only one vector-matrix-vector multiplication that is $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$ which is $\sqrt{}^{\in}$. Rest are scalars. The whole loop is then

$$\mathcal{O}((n^2 - p^2)p^2) \tag{2.60}$$

Before we'll move to the updating we must not to forget for quantities which we said we can calculate in advance. That is
$mathcalO(2np^2)$ for all $\boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T}$ and $\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$ and $\mathcal{O}(2np)$ for all e_j and e_j.

Finally we need to update RSS which is trivial. Next inversion by 2.40 which is $\mathcal{O}(4p^2)$ and 2.47 which is also $\mathcal{O}(4p^2)$. Finally we need to update $\hat{\boldsymbol{w}}$ by 2.42 and 2.51 which are both $\mathcal{O}(p^2 + p)$. Putting everything together we get

$$\mathcal{O}((n^2 - p^2)p^2 + 2np^2 + 2np + 8p^2 + 2p^2 + p) \sim \mathcal{O}(n^2p^2 - p^4) \tag{2.61}$$

nejak zminit kolikrat ten loop vetsinou projede atd..

Note that right now it doesn't matter if we use 2.27 with stopping criterion $\Delta S_{i,j}^{(m)} \geq 0$ or 2.58 and use $\rho(z_i, z_j) \geq 1$ stopping criterion. With both results we can update RSS. But the advantage of 2.58 is that we can use following bounding condition to improve performance.

Bounding condition improvement

$\rho(z_i, z_j)$ is expressed as ratio. We can see that in numerator we have

$$(1 + \boldsymbol{x_i Z^{-1} x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j Z^{-1} x_j^T} - \frac{e_i^2}{RSS}) + (\boldsymbol{x_i Z^{-1} x_j^T} + \frac{e_i e_j}{RSS})^2 \quad (2.62)$$

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

$$(1 + \boldsymbol{x_i Z^{-1} x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j Z^{-1} x_j^T} - \frac{e_i^2}{RSS}). \quad (2.63)$$

On the other hand we can see that denominator is

$$1 + \boldsymbol{x_i Z^{-1} x_i^T} - \boldsymbol{x_j Z^{-1} x_j^T} + (\boldsymbol{x_i Z^{-1} x_j^T})^2 - \boldsymbol{x_i Z^{-1} x_i^T x_j Z^{-1} x_j^T} \quad (2.64)$$

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

$$(\boldsymbol{x_i Z^{-1} x_j^T})^2 \leq \boldsymbol{x_i Z^{-1} x_i^T x_j} \quad (2.65)$$

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

$$1 + \boldsymbol{x_i Z^{-1} x_i^T} - \boldsymbol{x_j Z^{-1} x_j^T}. \quad (2.66)$$

Given that we can denote $\rho_b(z_i, z_j)$ so that

$$\rho_b(z_i, z_j) = \frac{(1 + \boldsymbol{x_i Z^{-1} x_i^T} + \frac{e_j^2}{RSS})(1 - \boldsymbol{x_j Z^{-1} x_j^T} - \frac{e_i^2}{RSS})}{1 + \boldsymbol{x_i Z^{-1} x_i^T} - \boldsymbol{x_j Z^{-1} x_j^T}} \leq \rho(z_i, z_j) \quad (2.67)$$

So the actual speed improvement is given by we don't need to compute $\boldsymbol{x_i Z^{-1} x_j^T}$ in each of $h(n-h)$ pairs swap comparison. Every time that algorithm starts to compare all pairs we set $\rho_{min} := 1$. Then every time we only compute $\rho_b(z_i, z_j)$ and if it is greater of equal to ρ_{min} we can continue to next pair without computing $\rho(z_i, z_j)$. On the other hand if $\rho_b(z_i, z_j)$ is less that ρ_{min} we compute $\rho(z_i, z_j)$ and set $\rho_{min} := \rho(z_i, z_j)$ This of course won't improve $\backslash^{\in}$ time required to compare all z_i, z_j pairs. Finally lets note that [3] call FSA which compute $\rho(z_i, z_j)$ 2.58 as *Optimal exchange algorithm (OEA)* and OEA using bounding condition 2.67 as *Modified optimal exchange algorithm (MOEA)*.

write the pseudocode

Different method of computing Improved FSA

zmenit to oznaceni Improved FSA na neco jineho. myslim ze to pouzivam nekonzistentne

We've described way how to modify FSA so that we can update ***hatw*** and inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$. This requires to compute inversion at the start of the algorithm (this is also the case for the FSA). In practice, however, this is not usually the way how OLS estimate $\hat{\boldsymbol{w}}$ is compute. That is primarily due to low numerical stability. In practice we usually use QR decomposition. In this subsection we describe how we can modify FSA to use QR decomposition. Let us start by describing Givens rotations algorithm in detail, because it is critical part of the modified algorithm.

Givens Rotation

In this section, we describe Givens rotations in detail. Moreover, we also show how to update QR decomposition in terms of including and excluding row from factorized matrix $\boldsymbol{X}$ which help us in our algorithm.

We compute the QR factorization of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ which has full column rank so that we apply orthogonal transformation by a matrix $\boldsymbol{Q}^T$ so that [19]

$$\boldsymbol{Q}^T\boldsymbol{A} = \boldsymbol{R} \tag{2.68}$$

where $\boldsymbol{Q}$ is product of orthogonal matrices. One such example of orthogonal matrix can be:

$$\boldsymbol{Q} = \begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \tag{2.69}$$

This matrix is indeed orthogonal since

$$\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{Q}^T\boldsymbol{Q}$$

$$= \begin{bmatrix} \cos(\varphi)^2 + \sin(\varphi)^2 & \sin(\varphi)\cos(\varphi) - \cos(\varphi)\sin(\varphi) \\ \cos(\varphi)\sin(\varphi) - \sin(\varphi)\cos(\varphi) & \cos(\varphi)^2 + \sin(\varphi)^2 \end{bmatrix} = \boldsymbol{I} \tag{2.70}$$

Moreover if we multiply this orthogonal with some column vector $\boldsymbol{x} \in \mathbb{R}^{2\times 1}$ thus $\boldsymbol{Q}\boldsymbol{x}$ as a result we get vector of same length but is rotated clockwise by φ radians. When we say that vector have same length we mean than L-2 norm of such vector stays the same.This is important property of all orthogonal matrices. We can simply verify correctness of this claim. Let us have any orthogonal matrix $\boldsymbol{Q} \in \mathbb{R}^{m\times m}$ and any column vector $\boldsymbol{x} \in \mathbb{R}^{m\times 1}$ then

$$\|\boldsymbol{Q}\boldsymbol{x}\|^2 = (\boldsymbol{Q}\boldsymbol{x})^T(\boldsymbol{Q}\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} = \|\boldsymbol{x}\|^2 \tag{2.71}$$

So we would like to create a series of orthogonal matrices which gradually rotates column vectors of $\boldsymbol{A}$, so we obtain zeros under diagonal. One such method - *Givens rotation* uses *Givens matricies* that zero one element under diagonal at a time. The example of 2.69 is not random. Let us look at this orthogonal matrix one more time and let us multiply this matrix with some vector.

$$\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ z \end{bmatrix} \tag{2.72}$$

To introduce this zeroing effect, we need to rotate this vector so that it is parallel to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Now it is only up to find $\cos(\varphi)a + \sin(\varphi) = r$. As we know rotation with $\boldsymbol{Q}$ preserve L-2 norm. That means if we want to $z = 0$ then r must be equal to L-2 norm of the vector $\begin{bmatrix} a \\ b \end{bmatrix}$ thus $r = \sqrt{a^2 + b^2}$. Then the solution seems trivial. We do not even need to calculate φ because if we put

$$\cos(\varphi) = \frac{a}{\sqrt{a^2 + b^2}} \tag{2.73}$$

and

$$\sin(\varphi) = \frac{b}{\sqrt{a^2 + b^2}} \tag{2.74}$$

then

$$r = \frac{a^2}{\sqrt{a^2 + b^2}} + \frac{b^2}{\sqrt{a^2 + b^2}} = \sqrt{a^2 + b^2} \tag{2.75}$$

$$z = \frac{-ab}{\sqrt{a^2 + b^2}} + \frac{ab}{\sqrt{a^2 + b^2}} = 0. \tag{2.76}$$

Practically used algorithm of computing $\cos\varphi$ and $\sin\varphi$ is slightly different because we want to prevent overflow. This algorithm is described by the following pseudocode:

Algorithm 5: Rotate

```
Input: a, b
Output: cos, sin
sin ← ∅;
cos ← ∅;
if b == 0 then
    sin ← 0;
    cos ← 1;
else if abs(b) ≥ abs(a) then
    cotg ← a/b;
    sin ← 1/√(1+(cotg)²);
    cos ← sin cotg;
else
    tan ← b/b;
    cos ← 1/√(1+(tan)²);
    sin ← cos tan;
end
return cos , sin;
```

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

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

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 calculate c and s by means of 2.73 and 2.74 for $a := x_i$ and $b := x_j$. Finally if we multiply $\boldsymbol{Q}(i,j)\boldsymbol{x} = \boldsymbol{p}$ we can see that $\boldsymbol{p}$ is same as vector $\boldsymbol{x}$ except p_i and p_j so that:

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

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. So if we have such matrix we need to create matrix $\boldsymbol{Q}(i,j)$ 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} then $a_{13} \dots a_{1_n}$. Then we start with second column a_{23} and so on. That means we need to create in total exactly $e = \frac{p^2-p}{2} + np - p^2$ matrices $\boldsymbol{Q}(i,j)$. We can denote this sequence of matrices as $\boldsymbol{Q_1}, \boldsymbol{Q_2}, \dots \boldsymbol{Q_e}$. The QR decomposition then looks like

$$\boldsymbol{Q_e} \dots \boldsymbol{Q_2}\boldsymbol{Q_1}\boldsymbol{A} = \boldsymbol{R} \tag{2.77}$$

where $\boldsymbol{Q_e} \dots \boldsymbol{Q_2}\boldsymbol{Q_1}$ is actually $\boldsymbol{Q}^T$. So $\boldsymbol{Q}$ is obtained by

$$\boldsymbol{Q} = \boldsymbol{Q_1^T}\boldsymbol{Q_2^T} \dots \boldsymbol{Q_e^T}. \tag{2.78}$$

We can see that for each row of the matrix we need one additional matrix $\boldsymbol{Q_i}$ and with such matrix, we are multiplying only two rows. Moreover, the rows are getting shorter after each finished column. So the time we see that time complexity is equal to

todo write the pseudocode

$$\sum_{i=1}^{n} \sum_{j=i+1}^{p} 6(n-i+1) \approx 6np^2 - 3np^2 - 3p^3 + 2p^3 = 3np^2 - p^3 \tag{2.79}$$

move all this about rotations somewhere else

So we are now in a situation when we have QR decomposition of $\boldsymbol{X}$, and we need to exchange ith row for jth row. We can simulate this by first adding jth row and consequently removing ith row.

Updating QR decomposition

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

$$\boldsymbol{R}^{+} = \boldsymbol{Q_e} \dots \boldsymbol{Q_2 Q_1 A_{new}} = \begin{bmatrix} \times & \cdots & \cdots & \cdots & \times \\ 0 & \times & \cdots & \cdots & \times \\ \vdots & 0 & \ddots & \cdots & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \vdots & 0 & \times \\ \vdots & \vdots & \vdots & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \times & \cdots & \cdots & \cdots & \times \end{bmatrix} \tag{2.80}$$

So all we need to do is create matrices $\boldsymbol{Q_{e+1}} \dots \boldsymbol{Q_{e+p}}$ to zero newly added row. So the R is then equal to $\boldsymbol{R_{new}} = \boldsymbol{R Q_{e+1} Q_e} \dots \boldsymbol{Q_2 Q_1 A}$. $\boldsymbol{Q}$ is updated in the same way thus $\boldsymbol{Q_{new}} = \boldsymbol{Q Q_e^T Q_{e+1}^T} \dots \boldsymbol{Q_{e+p}^T}$.

better notation

Algorithm 6: QR insert

Input: $\boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{x_i}$
Output: $\boldsymbol{Q}^{+}, \boldsymbol{R}^{+}$
return $\boldsymbol{Q}^{+}$, $\boldsymbol{R}^{+}$;

todo pseudokod? moc se mi nechce :-D

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

Note 25. For adding rows, the matrix $\boldsymbol{Q}$ is not needed so that we can update $\boldsymbol{R}$ to $\boldsymbol{R}^{+}$. Later we show that this is useful.

say where

When we are deleting the row $\boldsymbol{x_i}$ from matrix $\boldsymbol{A}$ we can use following trick [19]. First, we move such row as the first row of the matrix $\boldsymbol{A}$ so ve create permutation matrix $\boldsymbol{P}$ so that

$$\boldsymbol{PA} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{A(1:i-1,1:p)} \\ \boldsymbol{A(1:i+1,1:p)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_i} \\ \boldsymbol{A}^{-} \end{bmatrix} = \boldsymbol{PQR} \tag{2.81}$$

where $\boldsymbol{A(a:b,1:p)}$ means rows of matrix $\boldsymbol{A}$ from a to b. No we can se that we only need to zero first row $\boldsymbol{q_1}$ of matrix $\boldsymbol{Q}$. We can do this by $n-1$ matrixes matrixes $\boldsymbol{Q}(i,j) \in \mathbb{R}^{n \times n}$ so that

$$\boldsymbol{Q}(n-1,n) \dots \boldsymbol{Q}(1,2)\boldsymbol{q_1^T} = \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix} \tag{2.82}$$

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

$$\boldsymbol{Q}(n-1,n)\dots\boldsymbol{Q}(1,2)\boldsymbol{q}_1^T\boldsymbol{R} = \begin{bmatrix} \times \\ \boldsymbol{R}^- \\ \vdots \end{bmatrix}. \tag{2.83}$$

The result is

$$\boldsymbol{PA} = (\boldsymbol{PQQ^T}(1,2)\dots\boldsymbol{Q^T}(n-1,n))(\boldsymbol{Q}(n-1,n)\dots\boldsymbol{Q}(1,2)\boldsymbol{q}_1^T\boldsymbol{R}) = \begin{bmatrix} 1 & 0 \\ 0 & \boldsymbol{Q}^- \end{bmatrix}\begin{bmatrix} \times \\ \boldsymbol{R}^- \end{bmatrix}, \tag{2.84}$$

so that

$$\boldsymbol{A}^- = \boldsymbol{Q}^-\boldsymbol{R}^- \tag{2.85}$$

Algorithm 7: QR delete

Input: $\boldsymbol{Q}, \boldsymbol{R}, i$

~~**Output:** $\boldsymbol{Q}^-, \boldsymbol{R}^-$~~

return $\boldsymbol{Q}^-$, $\boldsymbol{R}^-$;

napsat pseudokod. opet se mi moc necche :-D

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

Calculation of imporved FSA using QR

So let's focus on our problem. As we said using decomposition is better than calculating inversion primarily due to higher numerical stability. We'll show that both FSA as well as improved FSA can be calculated using QR decomposition.

nevim jestli zminovat ze muzu delat i SVD dekompozici ktera je nejstabilnejsi ze vsech...

Let's start with 2.58. Here inversion is need to calculate $\boldsymbol{x_i Z}^{-1}\boldsymbol{x_i^T}$, $\boldsymbol{x_j Z}^{-1}\boldsymbol{x_j^T}$ and $\boldsymbol{x_i Z}^{-1}\boldsymbol{x_j^T}$. This can also be done without inversion, only using the decomposition. We can write this equation

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

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

$$\boldsymbol{R}^T\boldsymbol{v} = \boldsymbol{x_i}^T. \tag{2.87}$$

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

$$\boldsymbol{x_i Z}^{-1}\boldsymbol{x_j^T} = \boldsymbol{x_j u} \iff \boldsymbol{x_i}(\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{x_j^T} = \boldsymbol{x_j}^T\boldsymbol{u} \tag{2.88}$$

Where column vector $\boldsymbol{u}$ is defined by 2.39. We can see that

$$\boldsymbol{u} = \boldsymbol{Z}^{-1}\boldsymbol{x_i^T} \iff (\boldsymbol{R}^T\boldsymbol{R})^{-1}\boldsymbol{x_i^T} = \boldsymbol{u} \tag{2.89}$$

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

$$\boldsymbol{R}\boldsymbol{u} = \boldsymbol{v} \tag{2.90}$$

where $\boldsymbol{v}$ is solution of 2.87. Other quantities of 2.58 does not require $\boldsymbol{Z}^{-1}$.

When optimal exchange $\boldsymbol{z_i}, \boldsymbol{z_j}$ is found then we need to update RSS which can be done same way by 2.57. Updating $\hat{\boldsymbol{w}}$ to $\overline{\hat{\boldsymbol{w}}}$ by 2.42 requires $\boldsymbol{u}$ which in this case we calculate by 2.90 and b which requires $\boldsymbol{x_j}\boldsymbol{Z}^{-1}\boldsymbol{x_j^T}$ but that we've also already covered by idea 2.86. Analogous operations can be used to update $\overline{\hat{\boldsymbol{w}}}$ to $\overline{\overline{\hat{\boldsymbol{w}}}}$ by means of 2.51. Only thing wee need to realize that we can express 2.55 as $\boldsymbol{x_j}\overline{\boldsymbol{Z}}^{-1}\boldsymbol{x_j^T} = \boldsymbol{x_i}\boldsymbol{Z}^{-1}\boldsymbol{x_i^T} + b(\boldsymbol{u}\boldsymbol{x_j}^2)$.

We can't use updating inversion because we don't have one so we need to somehow update our QR decomposition. This can be done by both Householder rotation as well as by Givens rotation, but because our $\boldsymbol{R}$ is already sparse matrix, Givens rotations are ideal tool for this. We've already described how this can be done. First we can use 6 to add row $\boldsymbol{x_i}$ to the decomposition and consequently 7 to remove $\boldsymbol{x_j}$ from the decomposition. We can see that this is slower than updating inversion directly. On the other hand this solution is numerically stable and requires less time than computing factorization again from scratch.

Finally let's talk about matrix $\tilde{\boldsymbol{Z}}$ which we used 2.28 for derivation of our equations. If we use 2.19 observation then we realize that if we make QR factorization of $\boldsymbol{A} = (\boldsymbol{X}, \boldsymbol{y})$ then

$$\tilde{\boldsymbol{Z}} = \begin{bmatrix} \boldsymbol{Z} & \boldsymbol{X}^T\boldsymbol{y} \\ \boldsymbol{y}^T\boldsymbol{X} & \boldsymbol{y}^T\boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R^T} & 0 \\ \boldsymbol{\phi^T} & r \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{\phi} \\ 0 & r \end{bmatrix} \tag{2.91}$$

where

$$\tilde{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{\phi} \\ 0 & r \end{bmatrix} = \tilde{\boldsymbol{Q^T}}\boldsymbol{A} \tag{2.92}$$

is matrix from QR factorization of $\boldsymbol{A}$. Next we can realize that $\boldsymbol{R} \in \mathbb{R}^{p\times p}$ is actually 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

$$\boldsymbol{\phi} = \boldsymbol{Q^T}\boldsymbol{y} \tag{2.93}$$

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

$$\boldsymbol{R}\hat{\boldsymbol{w}} = \boldsymbol{\phi} \tag{2.94}$$

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

$$r^2 = \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\hat{\boldsymbol{w}} = RSS \tag{2.95}$$

Remark 26. We can see that all quantities we use in the algorithm can be obtained from matrix $\tilde{\boldsymbol{R}}$ moreover if we didn't need to remove rows from $\boldsymbol{X}$ then matrix $\tilde{\boldsymbol{Q}}$ would not need 25, thus speedup while inserting and rows would be possible.

Now we've described all properties of improved FSA algorithm. We can see that two different methods of computation are possible. One, using inversion $\boldsymbol{Z}^{-1}$ is faster, but numerically less stable, second using QR decomposition is slower due to updating matrix $\boldsymbol{R}$ and matrix $\boldsymbol{Q}$. We've also shown that using matrix $\tilde{\boldsymbol{R}}$ is useful because it contains all necessary quantities for the improved FSA algorithm, although it won't improve the speed of the algorithm. Later we'll introduce different algorithm where we won't need to remove rows from $\boldsymbol{X}$ so then we'll be ab1'A 'e to use this observation in our favor.

time complexity QR qwerty

2.5.1 MMEA

time complexity tildeZQR is the same as QR

2.6 BAB algorithm

no taak. to je na stranku. to das

2.7 BSA algorithm

zminit podobnost s: Hoffmann, Kontoghiorghes, 2010 (Matrix strategies SUR)

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

CHAPTER 3

Experiments

3.1 Data

3.2 Results

3.3 Outlier detection

In this chapter we describe implementation of all algorithms as well as data sets which we use for experiments. Then a bunch of experiments will be presented. Right now we are finishing implementation of algorithms so we do not have any experimental results yet. This will change this or next week... Chapter will be about 4 pages long.

Conclusion

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

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

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

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

Bibliography

[1] 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.

[2] 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.

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

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

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

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

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

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

[9] 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.

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

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

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

[13] 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).

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

[15] 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.

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

[17] 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.

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

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

Appendix A

Data sets

GUI Graphical user interface

XML Extensible markup language

Appendix B

Contents of enclosed CD

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