University of Heidelberg

Department of Mathematics and Computer Science

B.Sc. Mathematics 100%

Bachelor Thesis

Supervisor:

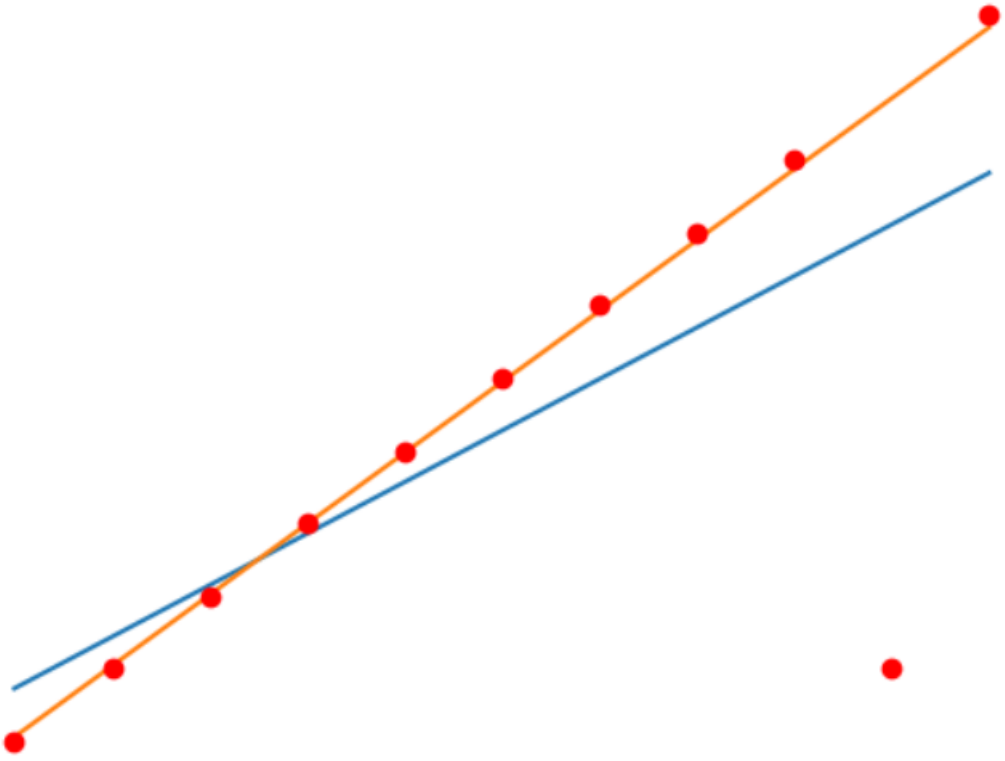
Prof. Ekaterina Kostina

Author:

Thomas Gantz

Academic Year 2022/2023

Optimisation with Different Cost Functions: Comparing Huber and Least Squares



or

Outperforming Least Squares

Abstract

The objective of this thesis is to introduce a robust one-dimensional polynomial regression model using the Huber cost function[1] via Convex Quadratic Optimisation[2] and to compare its performance against the non-robust Least Squares approach via a concrete implementation using the qpOASES[3] algorithm for solving the obtained Convex Quadratic Programs.

Contents

1 Introduction

Regression analysis is a statistical technique to estimate the relationship between a dependent variable (usually Y) and an independent variable (usually X)[4]. A widely used form of regression analysis is linear regression, where the goal is to find a line (or a more complex linear combination) that best fits, for example, experimentally taken data points based on a specific mathematical criterion (usually referred to as minimising the costs). The choice of the cost function is crucial as it significantly affects the results.

Least Squares could be considered the most widely used approach (being already developed during the eighteenth century[5], explicitly published as such by Legendre in 1805[6] and Gauß in 1809[7]). It calculates the line (or hyperplane) that minimises the sum of squared differences between the known data points and that line (or hyperplane). However, there are cases where the Least Squares method may not be suitable for the intended application.

For example, consider the following scenarios:
a) You have obtained experimental data points, but instrumental unreliability may lead to a few erroneous results.
b) You want to model economic development over a long period, and at some point, a crisis (e.g. COVID-19) influenced your economy. Therefore, not all resulting data points accurately reflect the actual economic development.

A Least Squares modelling approach would be problematic because it weights the pointwise errors quadratically. As a result, outliers (data points that differ significantly from the other observations[8][9]) will have a massive impact on the regression function.

The Huber method overcomes this issue by using a different cost function (stated in 1964 by Swiss mathematician Peter J. Huber[1]) that starts quadratically from 0 and becomes linear at a certain point. Hence, it weights outliers less and predominantly focuses on the data points that seem more connected, prospectively the relevant data. Therefore, Huber may suit data sets with outliers better than Least Squares.

But how can we make Huber explicitly applicable? And is it really better than Least Squares?

2 Overview

This thesis starts with the problem of multi-dimensional linear regression modelling as introduced in section 3. It involves choosing a suitable cost function. One-dimensional polynomial regression is a subcase of the more general problem, as demonstrated in subsection 3.1. A brief discussion of cost functions, along with their advantages and disadvantages, is presented in subsection 3.2.

Whether considering Least Squares or Huber, the problem will be reformulated in the form of an Convex Quadratic Program (CQP), as explained in section 4. For Least Squares, the result is Theorem 4.1.1, while for Huber, it is Theorem 4.2.4. These will be solved by the qpOASES[3] algorithm that solves Convex Quadratic Programs.

Regression polynomials for different application cases obtained by the programs developed in this thesis for Least Squares and Huber are shown in section 5. In this section, the reader may observe when to stick with Least Squares and when to choose Huber instead.

The implementation of Solvers_Huber.py, which you can also call through Solvers_Interface.py, can be found in Appendix B. Additionally, the classical approach of Least Squares is implemented in Solvers_LeastSquares.py and is also callable through the interface. Its use is explained in section 6.

3 General Problem

We consider a multi-dimensional linear regression modelling approach. This is a system of linear equations which, in the best case, can be solved for X analytically:

Linear Regression.

$$AX = B \tag{3.0.1}$$

where:

$$A \in \mathbb{R}^{l \times d}, X \in \mathbb{R}^{d}, B \in \mathbb{R}^{l}$$

The system might be unsolvable. For instance, if $l > d$, we cannot solve for X with any random B, even if the column vectors of A are linearly independent. In this case, one has to go for a numerical approach, trying to find a vector X that minimises the costs caused by the error between the model and the data points. In subsection 3.2, we will introduce suitable cost functions that quantify the cost of the difference between the calculated value of AX and the actual value of B for each column i:

Cost Function.

$$\rho : \mathbb{R} \to \mathbb{R} \tag{3.0.2}$$

$$(AX - B)_{i \in [|1,l|]} \mapsto \text{cost of this distance}$$

Using this column-wise cost function, we formulate a Minimisation Problem, which consists in finding a vector X that minimises the total costs given by the sum over the columnswise costs:

General Problem.

$$f_{initial} : \sum_{i=1}^{l} \rho((AX - B)_i) \to \min_{X \in \mathbb{R}^d} \tag{3.0.3}$$

3.1 Subcase: One-dimensional Polynomial Regression

The General Problem of multi-dimensional linear regression described above contains as a subcase one-dimensional polynomial regression, which is stated as follows.

Given:

Dataset of k Points.

$$((x_1, y_1), ..., (x_k, y_k)) \in (\mathbb{R}^2)^k \tag{3.1.1}$$

We want to find:

A Regression Polynomial of Degree n.

$$P : \mathbb{R} \to \mathbb{R}, x \mapsto \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n \tag{3.1.2}$$

such that the polynomial approximates the given dataset in the most precise way concerning a cost function yet-to-be-defined.

We transform this problem into the form of the General Problem by defining:

One-dimensional Polynomial Regression Input.

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ & & \vdots & & \\ 1 & x_k & x_k^2 & \dots & x_k^n \end{pmatrix}, X = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_n \end{pmatrix}, B = \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix} \tag{3.1.3}$$

3.2 Cost Functions

To solve the General Problem, we need to choose a specific Cost Function. This choice is crucial as it affects how the problem can be solved and what the solution will look like.

Here are some possible choices:

Absolute Value Cost Function.

$$\rho_{\text{absolute value}}(t) = |t| \tag{3.2.1}$$

Least Squares Cost Function.

$$\rho_{\text{Least Squares}}(t) = t^2 \tag{3.2.2}$$

Huber Cost Function[1].

$$\rho_{\text{Huber}}(t) = \begin{cases} \frac{1}{2}t^2 & |t| \leq \gamma \\ \gamma|t| - \frac{1}{2}\gamma^2 & |t| > \gamma \end{cases} \quad \text{for } \gamma \in \mathbb{R}_{>0} \tag{3.2.3}$$

The Least Squares Cost Function could be considered the most widely used cost function, perhaps because its use is straightforward. For instance, it leads to a simple Convex Quadratic Program, as shown in Theorem 4.1.1. But it has the disadvantage of not being robust, meaning it is quite sensitive to outliers.

The Absolute Value Cost Function is robust but could be expected complicated to reformulate into a solvable optimisation program for not being differentiable.

The Huber Cost Function[1] is robust and easy to work with. It results in a simple Convex Quadratic Program, as shown in Theorem 4.2.4.

4 Reformulating into a CQP

The goal of section 4 is to reformulate the General Problem, considering, on one hand, the Least Squares Cost Function, and on the other hand, the Huber Cost Function[1], in the form of a Convex Quadratic Program (CQP):

Convex Quadratic Program.

$$\frac{1}{2}X^T H X + X^T g \to \min_{X \in \mathbb{R}^n} \tag{4.0.1}$$

$$s.t.\ \mathcal{A}X \leq \mathcal{B}$$

where

$$X \in \mathbb{R}^n, H \in \mathbb{R}^{n \times n} \text{ symmetric positive semidefinite}, g \in \mathbb{R}^n$$

$$\mathcal{A} \in \mathbb{R}^{m \times n}, \mathcal{B} \in \mathbb{R}^m$$

For Least Squares, the corresponding result is Theorem 4.1.1, while for Huber, it is Theorem 4.2.4.

4.1 Considering the Least Squares Method

Theorem 4.1.1 (CQP for Least Squares)**.**
The General Problem with the Least Squares Cost Function is equivalent to an unconstrained Convex Quadratic Program:

CQP for Least Squares.

$$f_{LQ}^{CQP} : \frac{1}{2} X^T H X + X^T g \to \min_{X \in \mathbb{R}^n} \tag{4.1.1}$$

where

$$X \in \mathbb{R}^n, H \in \mathbb{R}^{n \times n} \text{ symmetric}, g \in \mathbb{R}^n$$

with

$$H = 2A^T A,\ g = -2B^T A$$

Proof.
We start from the General Problem with the Least Squares Cost Function (n = d, m = l).

$$\begin{aligned}
\sum_{i=1}^{l} (AX - B)_i^2 &\to \min_{X \in \mathbb{R}^n} \\
= \|AX - B\|_2^2 &\to \min_{X \in \mathbb{R}^n} \\
= (AX - B)^T (AX - B) &\to \min_{X \in \mathbb{R}^n} \\
= X^T A^T AX - B^T AX - X^T A^T B + B^T B &\to \min_{X \in \mathbb{R}^n} \\
= X^T A^T AX + X^T - 2B^T A + B^T B &\to \min_{X \in \mathbb{R}^n} \\
\sim \frac{1}{2} X^T \underbrace{2A^T A}_{H} X + X^T \underbrace{-2B^T A}_{g} &\to \min_{X \in \mathbb{R}^n}
\end{aligned}$$

We note that H is symmetric:

$$H^T = (2A^T A)^T = 2A^T A^{T^T} = 2A^T A = H$$

Furthermore, H is positive semidefinite:

$$x^T H x = x^T 2A^T Ax = 2x^T A^T Ax = 2(Ax)^T Ax = 2\|Ax\|_2^2 \geq 0 \tag{4.1.2}$$

Hence, we see that this indeed satisfies the criteria of being an (unconstrained) Convex Quadratic Program. □

Remark 4.1.2 (Existence and uniqueness of a solution)**.**
(i) Theorem 4.1.1 guarantees the existence of a solution since the objective function is an unconstrained convex function.
(ii) If matrix A has full column rank, then the solution to the CQP for Least Squares is unique because A is injective, and the inequality symbol $\geq$ in equation (4.1.2) can be replaced with $>$.

Remark 4.1.3 (qpOASES[3] input for Least Squares)**.**
(i) The General Problem with the Least Squares Cost Function is equivalent to a program, which is a valid input for the algorithm's QProblemB class:

qpOASES[3] Input Program for Least Squares.

$$f_{LQ}^{qpOASES} : \frac{1}{2} X^T H X + X^T g \to \min_{X \in \mathbb{R}^n} \tag{4.1.3}$$

s.t.

$$lb \leq X \leq ub$$

where

$$X \in \mathbb{R}^n, H \in \mathbb{R}^{n \times n} \text{ symmetric}, g \in \mathbb{R}^n$$
$$lb \in \mathbb{R}^n \cup \{\infty, -\infty\}^n, ub \in \mathbb{R}^n \cup \{\infty, -\infty\}^n$$

with

$$H = 2A^T A,\ g = -2B^T A$$
$$lb = \begin{pmatrix} -\infty \\ \vdots \\ -\infty \end{pmatrix},\ ub = \begin{pmatrix} \infty \\ \vdots \\ \infty \end{pmatrix}$$

(ii) Since this program is a Convex Quadratic Program (CQP), we know that the solution given is not only a critical point but an actual minimiser.

4.2 Considering the Huber Method

The proofs in this section are derived from the approach presented here[2].

In this subsection, we deal with the General Problem considering the Huber Cost Function[1]. Hence, let $\gamma > 0$ be fixed in this section.

First, we start with a lemma reformulating the Huber Cost Function[1] itself.

Lemma 4.2.1 (Reformulating the Huber cost function[1])**.**

$$\rho_{Huber}(t) = \min_{z \in \mathbb{R}} \frac{1}{2}z^2 + \gamma|t - z| \tag{4.2.1}$$

Proof.
Define:

$$\theta_t(z) : \mathbb{R} \to \mathbb{R},\ z \mapsto \theta_t(z) = \frac{1}{2}z^2 + \gamma|t - z| = \begin{cases} \frac{1}{2}z^2 + \gamma(t - z) & z < t \\ \frac{1}{2}z^2 & z = t \\ \frac{1}{2}z^2 + \gamma(z - t) & z > t \end{cases}$$

This function is convex as a positive combination of convex functions. Furthermore, it is non-differentiable in $z_0 = t$. Therefore, we calculate the Subgradient using A.0.1:

$$\delta\theta_t(z_0) = \begin{cases} \{z - \gamma\} & z_0 < t \\ \{z + \lambda\gamma | \lambda \in [-1, 1]\} & z_0 = t \\ \{z + \gamma\} & z_0 > t \end{cases}$$

We remark:

$$\exists\, 0 \in \delta\theta_t(z_0) \Leftrightarrow \begin{cases} z_0 = \gamma & z_0 < t \\ z_0 = -\lambda\gamma,\ \lambda \in [-1, 1] & z_0 = t \\ z_0 = -\gamma & z_0 > t \end{cases}$$

Using A.0.2, we know that the minimum is attained in these points and obtain:

$$\min_{z \in \mathbb{R}} \frac{1}{2}z^2 + \gamma|t - z| = \begin{cases} \frac{1}{2}\gamma^2 + \gamma(t - \gamma) = \gamma t - \frac{1}{2}\gamma^2 & \gamma < t \\ \frac{1}{2}t^2 & t \in [-\gamma, \gamma] \\ \frac{1}{2}\gamma^2 + \gamma(-t - \gamma) = -\gamma t - \frac{1}{2}\gamma^2 & -\gamma > t \end{cases}$$

This is exactly the Huber Cost Function[1]. □

Now we can use this lemma to reformulate our General Problem considering the Huber Cost Function[1] in multiple steps to end up at the CQP for Huber.

Lemma 4.2.2.
The General Problem considering the Huber Cost Function[1] is equal to:

$$f^1_{Huber} : \sum_{i=1}^{l} \min_{z\in\mathbb{R}} \frac{1}{2}z^2 + \gamma|(AX-B)_i - z| \to \min_{X\in\mathbb{R}^d} \tag{4.2.2}$$

Proof. Use 4.2.1 to substitute the Huber Cost Function[1] in the General Problem. □

Lemma 4.2.3.
(4.2.2) *is equivalent to:*

$$f^5_{Huber} : \frac{1}{2}\|Z\|_2^2 + \gamma eT \to \min_{X\in\mathbb{R}^d, Z\in\mathbb{R}^l, T\in\mathbb{R}^l} \tag{4.2.3}$$

s.t.

$$-T \leq AX - B - Z \leq T$$

Remarks:
- $e = (1, ...1) \in \mathbb{R}^l$
- *the multiplication of two vectors is meant as their scalar product*
- $\leq$ *of two vectors is meant componentwise*

Proof.

(4.2.2) can be rewritten as:

$$f^2_{Huber} : \sum_{i=1}^{l} \frac{1}{2}z_i^2 + \gamma|(AX-B)_i - z_i| \to \min_{X\in\mathbb{R}^d, z_1\in\mathbb{R}, ..., z_l\in\mathbb{R}} \tag{4.2.4}$$

(4.2.4) is equivalent to:

$$f^3_{Huber} : \sum_{i=1}^{l} \frac{1}{2}Z_i^2 + \gamma|(AX-B)_i - Z_i| \to \min_{X\in\mathbb{R}^d, Z\in\mathbb{R}^l} \tag{4.2.5}$$

via $Z = (z_1...z_l)$

(4.2.5) is equal to:

$$f^4_{Huber} : \frac{1}{2}\|Z\|_2^2 + \gamma\|AX - B - Z\|_1 \to \min_{X\in\mathbb{R}^d, Z\in\mathbb{R}^l} \tag{4.2.6}$$

(4.2.6) is equivalent to (4.2.3) via

$$\Phi : \underset{X,Z}{argmin}\, f^4_{Huber} \leftrightarrow \underset{X,Z,T}{argmin}\, f^5_{Huber}$$

$$(X, Z) \mapsto (X, Z, T) \text{ with } T_i = |(AX - B - Z)_i| \; \forall i \in \{1, ..., l\}$$

Let T_i be a random component of T for some $i \in \{1, ..., l\}$.

We remark that considering the componentwise constraints

$$-T_i \leq (AX - B - Z)_i \leq T_i$$

gives us:

$$(AX - B - Z)_i \geq 0 \Rightarrow T_i = (AX - B - Z)_i$$
$$(AX - B - Z)_i < 0 \Rightarrow T_i = -(AX - B - Z)_i$$

Therefore:

$$T_i = |(AX - B - Z)_i| \qquad (*)$$

Now we look at (4.2.6), in particular the second summand. We note:

$$\|AX - B - Z\|_1 = \sum_{i=1}^{l} |(AX - B - Z)_i| \overset{*}{=} \sum_{i=1}^{l} T_i = eT$$

□

Theorem 4.2.4 (CQP for Huber)**.**
(4.2.3) *is equivalent to a constrained Convex Quadratic Program:*

CQP for Huber.

$$f_{Huber}^{CQP} : \frac{1}{2}\mathcal{X}^T H\mathcal{X} + \mathcal{X}^T g \to \min_{\mathcal{X}\in\mathbb{R}^n} \tag{4.2.7}$$

s.t.

$$\mathcal{A}\mathcal{X} \leq \mathcal{B}$$

where

$$\mathcal{X} \in \mathbb{R}^n, H \in \mathbb{R}^{n\times n} \text{ symmetric}, g \in \mathbb{R}^n$$
$$\mathcal{A} \in \mathbb{R}^{m\times n}, \mathcal{B} \in \mathbb{R}^m$$

with

$$H = \begin{pmatrix} 0 & & & & & & & \\ & \ddots & & & & & & \\ & & 0 & & & & & \\ & & & 1 & & & & \\ & & & & \ddots & & & \\ & & & & & 1 & & \\ & & & & & & 0 & \\ & & & & & & & \ddots \\ & & & & & & & & 0 \end{pmatrix}, \; g = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ \gamma \\ \vdots \\ \gamma \end{pmatrix}$$

$$\mathcal{A} = \begin{pmatrix} A & -\mathbb{I} & -\mathbb{I} \\ -A & +\mathbb{I} & -\mathbb{I} \end{pmatrix}, \; \mathcal{B} = \begin{pmatrix} B \\ -B \end{pmatrix}$$

via

$$\Phi : \underset{X,Z}{argmin}\, f_{Huber}^5 \leftrightarrow \underset{\mathcal{X}}{argmin}\, f_{Huber}^{CQP}$$

$$(X, Z, T) \mapsto \mathcal{X} = \begin{pmatrix} X \\ Z \\ T \end{pmatrix}$$

Proof.
Combining X, Z, and T from equation (4.2.3) into a vector $\mathcal{X}$ and accordingly restating the objective function and constraints leads to a H with eigenvalues $0, ..., 0, 1, ..., 1, 0, ..., 0$. Hence, it is positive semidefinite and meets the requirements of a Convex Quadratic Program. □

Remark 4.2.5 (qpOASES[3] input for Huber)**.**
(i) The General Problem with the Huber Cost Function[1] is equivalent to a program, which is a valid input for the algorithm's QProblem class:

qpOASES[3] Input Program for Huber.

$$f_{Huber}^{qpOASES} : \frac{1}{2}\,\mathcal{X}^T H \mathcal{X} + \mathcal{X}^T g \to \min_{\mathcal{X}\in\mathbb{R}^n} \tag{4.2.8}$$

s.t.

$$lbA \leq \mathcal{A}\mathcal{X} \leq ubA$$
$$lb \leq \mathcal{X} \leq ub$$

where

$$\mathcal{X} \in \mathbb{R}^n, H \in \mathbb{R}^{n\times n} \text{ symmetric}, g \in \mathbb{R}^n$$
$$lbA \in \mathbb{R}^m \cup \{\infty, -\infty\}^m, \mathcal{A} \in \mathbb{R}^{m\times n}, ubA \in \mathbb{R}^m \cup \{\infty, -\infty\}^m$$
$$lb \in \mathbb{R}^n \cup \{\infty, -\infty\}^n, ub \in \mathbb{R}^n \cup \{\infty, -\infty\}^n$$

with

$$H = \begin{pmatrix} 0 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 0 & & & & & & \\ & & & 1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & & & \\ & & & & & & 0 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 0 \end{pmatrix}, \; g = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ \gamma \\ \vdots \\ \gamma \end{pmatrix}$$

$$lbA = \begin{pmatrix} -\infty \\ \vdots \\ -\infty \end{pmatrix}, \; \mathcal{A} = \begin{pmatrix} A & -\mathbb{I} & -\mathbb{I} \\ -A & +\mathbb{I} & -\mathbb{I} \end{pmatrix}, \; ubA = \begin{pmatrix} B \\ -B \end{pmatrix}$$

$$lb = \begin{pmatrix} -\infty \\ \vdots \\ -\infty \end{pmatrix}, \; ub = \begin{pmatrix} \infty \\ \vdots \\ \infty \end{pmatrix}$$

(ii) Since this program is a Convex Quadratic Program (CQP), we know that the solution given is not only a critical point but an actual minimiser.
(iii) The solution for X is obtained by the corresponding part of $\mathcal{X}$.

5 Experiments

5.1 Description

The experiments aim to compare the performance of the two solving methods of this thesis, Least Squares and Huber, in their application case of one-dimensional polynomial regression.

For Least Squares, the method is used in its standard version and after removing all outliers (if there are such). That is possible since they will be chosen manually, hence are known. For Huber, different γ values (0.5, 1.0, 2.0) are used. The regression polynomials are calculated for degrees (n) 1, 2, and 3.

Two experiments are conducted that differ in the way the data points on which the solving methods explained above are generated.

The first experiment generates a polynomial by assigning random coefficients within the range [0, 1). Noise (0.01, 0.03, 0.10) is introduced to the data points in a normal-distributed manner. Outliers (0, 1, or 2) are added by multiplying the y-values of randomly selected data points by a factor of 3.

The results of the experiments are plots displaying the original data points and the regression curves obtained for one run of the experiment. A table next to the plot presents their coefficients. A second table presents the ratio of the initial coefficients (of the random polynomial) to the numerical coefficients, calculated as the arithmetic mean over 10^6 iterations.

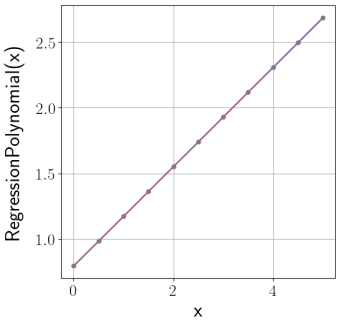
In the second experiment, random points are chosen in a uniform way.

The Experiments_GraphicalRepresentation.py and Experiments_Ratio.py files conduct the experiments described above. The actual solving methods are available in Solvers_LeastSquares.py and Solvers_Huber.py.

5.2 Results

5.2.1 Points along a Random Polynomial

Figure 1: graphical representation of one run of the experiment for $n = 1$, 0 outliers, noise = 0.01

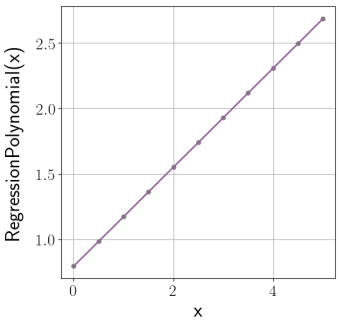


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Huber ($\gamma = 0.5$)	0.79	0.38
—	Huber ($\gamma = 1.0$)	0.79	0.38
—	Huber ($\gamma = 2.0$)	0.79	0.38
—	Least Squares (stand.)	0.79	0.38

(b) Coefficients

Figure 2: graphical representation of one run of the experiment for $n = 1$, 0 outliers, noise = 0.03

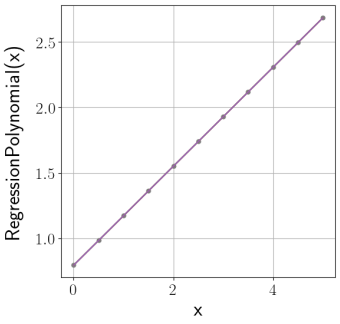


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Huber ($\gamma = 0.5$)	0.79	0.38
—	Huber ($\gamma = 1.0$)	0.79	0.38
—	Huber ($\gamma = 2.0$)	0.79	0.38
—	Least Squares (stand.)	0.79	0.38

(b) Coefficients

Figure 3: graphical representation of one run of the experiment for $n = 1$, 0 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Huber ($\gamma = 0.5$)	0.79	0.38
—	Huber ($\gamma = 1.0$)	0.79	0.38
—	Huber ($\gamma = 2.0$)	0.79	0.38
—	Least Squares (stand.)	0.79	0.38

(b) Coefficients

Table 1: ratio calculated for 10^6 runs of the experiment for $n = 1$, 0 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00
Least Squares (stand.)	1.00	1.00

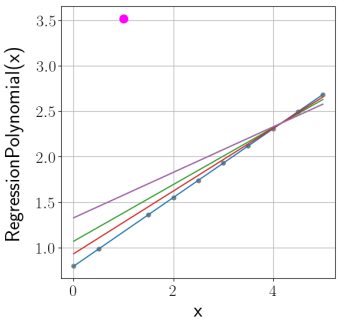
Table 2: ratio calculated for 10^6 runs of the experiment for $n = 1$, 0 outliers, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00
Least Squares (stand.)	1.00	1.00

Table 3: ratio calculated for 10^6 runs of the experiment for $n = 1$, 0 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00
Least Squares (stand.)	1.00	1.00

Figure 4: graphical representation of one run of the experiment for $n = 1$, 1 outlier, noise = 0.01

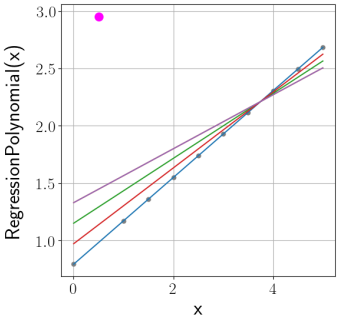


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
	Least Squares (rem. out.)	0.79	0.38
	Huber ($\gamma = 0.5$)	1.33	0.25
	Huber ($\gamma = 1.0$)	1.07	0.31
	Huber ($\gamma = 2.0$)	0.93	0.35
	Least Squares (stand.)	1.33	0.25

(b) Coefficients

Figure 5: graphical representation of one run of the experiment for $n = 1$, 1 outlier, noise = 0.03

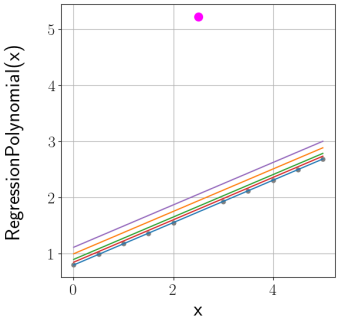


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
	Least Squares (rem. out.)	0.79	0.38
	Huber ($\gamma = 0.5$)	1.33	0.23
	Huber ($\gamma = 1.0$)	1.15	0.28
	Huber ($\gamma = 2.0$)	0.97	0.33
	Least Squares (stand.)	1.33	0.23

(b) Coefficients

Figure 6: graphical representation of one run of the experiment for $n = 1$, 1 outlier, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
	Least Squares (rem. out.)	0.79	0.38
	Huber ($\gamma = 0.5$)	0.99	0.38
	Huber ($\gamma = 1.0$)	0.89	0.38
	Huber ($\gamma = 2.0$)	0.84	0.38
	Least Squares (stand.)	1.11	0.38

(b) Coefficients

Table 4: ratio calculated for 10^6 runs of the experiment for $n = 1$, 1 outlier, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.95	0.99
Huber ($\gamma = 1.0$)	0.90	0.97
Huber ($\gamma = 2.0$)	0.82	0.95
Least Squares (stand.)	0.65	0.88

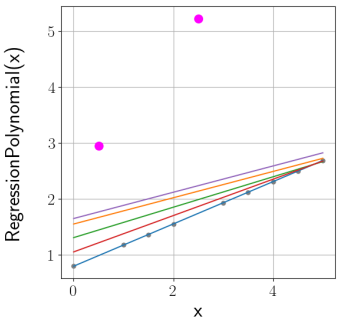
Table 5: ratio calculated for 10^6 runs of the experiment for $n = 1$, 1 outlier, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.90	1.00
Huber ($\gamma = 1.0$)	0.82	1.00
Huber ($\gamma = 2.0$)	0.70	1.00
Least Squares (stand.)	0.52	1.00

Table 6: ratio calculated for 10^6 runs of the experiment for $n = 1$, 1 outlier, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.72	1.07
Huber ($\gamma = 1.0$)	0.56	1.15
Huber ($\gamma = 2.0$)	0.50	1.20
Least Squares (stand.)	0.50	1.00

Figure 7: graphical representation of one run of the experiment for $n = 1$, 2 outliers, noise = 0.01

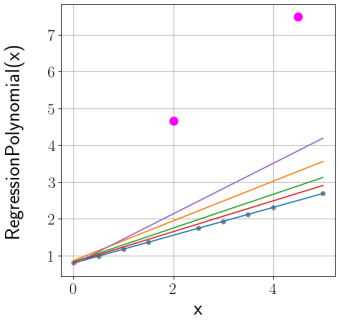


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Least Squares (rem. out.)	0.79	0.38
—	Huber ($\gamma = 0.5$)	1.55	0.23
—	Huber ($\gamma = 1.0$)	1.31	0.27
—	Huber ($\gamma = 2.0$)	1.05	0.32
—	Least Squares (stand.)	1.65	0.23

(b) Coefficients

Figure 8: graphical representation of one run of the experiment for $n = 1$, 2 outliers, noise = 0.03

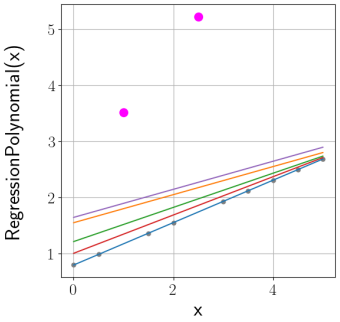


(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Least Squares (rem. out.)	0.79	0.38
—	Huber ($\gamma = 0.5$)	0.87	0.54
—	Huber ($\gamma = 1.0$)	0.83	0.46
—	Huber ($\gamma = 2.0$)	0.81	0.42
—	Least Squares (stand.)	0.76	0.68

(b) Coefficients

Figure 9: graphical representation of one run of the experiment for $n = 1$, 2 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1
	Initial	0.79	0.38
—	Least Squares (rem. out.)	0.79	0.38
—	Huber ($\gamma = 0.5$)	1.55	0.25
—	Huber ($\gamma = 1.0$)	1.21	0.30
—	Huber ($\gamma = 2.0$)	1.00	0.34
—	Least Squares (stand.)	1.64	0.25

(b) Coefficients

Table 7: ratio calculated for 10^6 runs of the experiment for $n = 1$, 2 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.90	0.97
Huber ($\gamma = 1.0$)	0.81	0.94
Huber ($\gamma = 2.0$)	0.69	0.88
Least Squares (stand.)	0.68	0.71

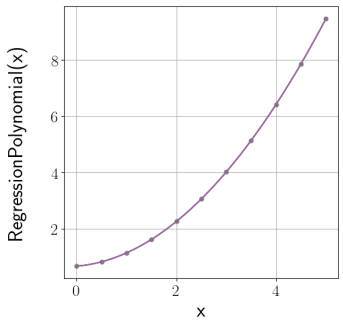
Table 8: ratio calculated for 10^6 runs of the experiment for $n = 1$, 2 outliers, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.59	1.11
Huber ($\gamma = 1.0$)	0.53	1.13
Huber ($\gamma = 2.0$)	0.48	1.13
Least Squares (stand.)	0.39	1.13

Table 9: ratio calculated for 10^6 runs of the experiment for $n = 1$, 2 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$
Least Squares (rem. out.)	1.00	1.00
Huber ($\gamma = 0.5$)	0.81	1.00
Huber ($\gamma = 1.0$)	0.67	1.00
Huber ($\gamma = 2.0$)	0.51	1.00
Least Squares (stand.)	0.66	0.75

Figure 10: graphical representation of one run of the experiment for $n = 2$, 0 outliers, noise = 0.01

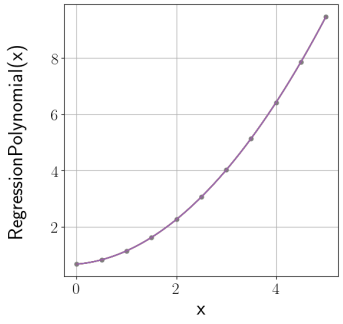


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	0.67	0.15	0.32
—	Huber ($\gamma = 1.0$)	0.67	0.15	0.32
—	Huber ($\gamma = 2.0$)	0.67	0.15	0.32
—	Least Squares (stand.)	0.67	0.15	0.32

(b) Coefficients

Figure 11: graphical representation of one run of the experiment for $n = 2$, 0 outliers, noise = 0.03

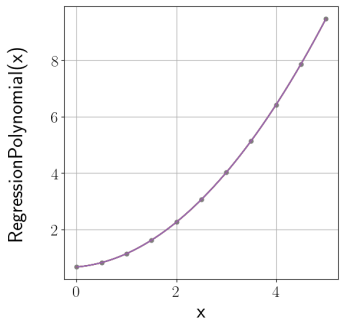


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	0.67	0.15	0.32
—	Huber ($\gamma = 1.0$)	0.67	0.15	0.32
—	Huber ($\gamma = 2.0$)	0.67	0.15	0.32
—	Least Squares (stand.)	0.67	0.15	0.32

(b) Coefficients

Figure 12: graphical representation of one run of the experiment for $n = 2$, 0 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	0.67	0.15	0.32
—	Huber ($\gamma = 1.0$)	0.67	0.15	0.32
—	Huber ($\gamma = 2.0$)	0.67	0.15	0.32
—	Least Squares (stand.)	0.67	0.15	0.32

(b) Coefficients

Table 10: ratio calculated for 10^6 runs of the experiment for $n = 2$, 0 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00

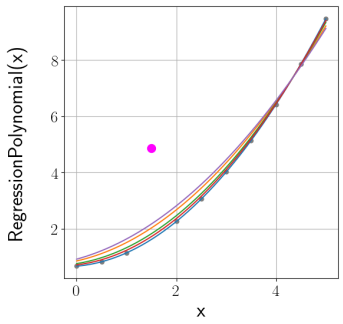
Table 11: ratio calculated for 10^6 runs of the experiment for $n = 2$, 0 outliers, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00

Table 12: ratio calculated for 10^6 runs of the experiment for $n = 2$, 0 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00

Figure 13: graphical representation of one run of the experiment for $n = 2$, 1 outlier, noise = 0.01

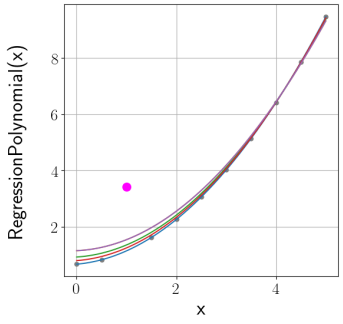


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Least Squares (rem. out.)	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	0.86	0.40	0.25
—	Huber ($\gamma = 1.0$)	0.76	0.28	0.29
—	Huber ($\gamma = 2.0$)	0.72	0.21	0.30
—	Least Squares (stand.)	0.92	0.49	0.23

(b) Coefficients

Figure 14: graphical representation of one run of the experiment for $n = 2$, 1 outlier, noise = 0.03

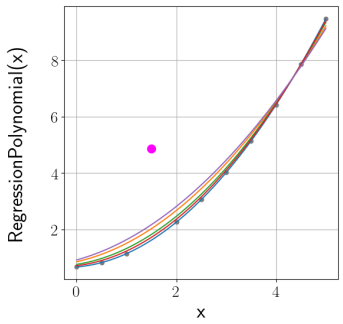


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Least Squares (rem. out.)	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	1.15	0.08	0.31
—	Huber ($\gamma = 1.0$)	0.92	0.11	0.32
—	Huber ($\gamma = 2.0$)	0.8	0.13	0.32
—	Least Squares (stand.)	1.15	0.08	0.31

(b) Coefficients

Figure 15: graphical representation of one run of the experiment for $n = 2$, 1 outlier, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
—	Least Squares (rem. out.)	0.67	0.15	0.32
—	Huber ($\gamma = 0.5$)	0.86	0.40	0.25
—	Huber ($\gamma = 1.0$)	0.76	0.28	0.29
—	Huber ($\gamma = 2.0$)	0.72	0.21	0.30
—	Least Squares (stand.)	0.92	0.49	0.23

(b) Coefficients

Table 13: ratio calculated for 10^6 runs of the experiment for $n = 2$, 1 outlier, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	1.05	0.36	1.06
Huber ($\gamma = 1.0$)	1.11	0.22	1.12
Huber ($\gamma = 2.0$)	1.26	0.12	1.28
Least Squares (stand.)	1.59	0.07	1.68

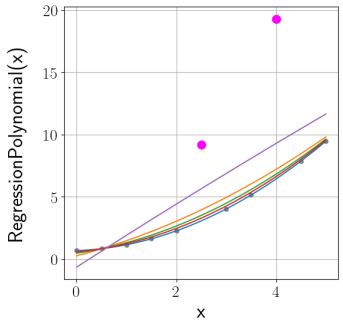
Table 14: ratio calculated for 10^6 runs of the experiment for $n = 2$, 1 outlier, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.49	0.80	0.96
Huber ($\gamma = 1.0$)	0.45	0.61	0.95
Huber ($\gamma = 2.0$)	0.45	0.61	0.95
Least Squares (stand.)	0.45	0.61	0.95

Table 15: ratio calculated for 10^6 runs of the experiment for $n = 2$, 1 outlier, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.85	0.51	1.04
Huber ($\gamma = 1.0$)	0.73	0.35	1.08
Huber ($\gamma = 2.0$)	0.58	0.21	1.17
Least Squares (stand.)	0.54	0.18	1.20

Figure 16: graphical representation of one run of the experiment for $n = 2$, 2 outliers, noise = 0.01

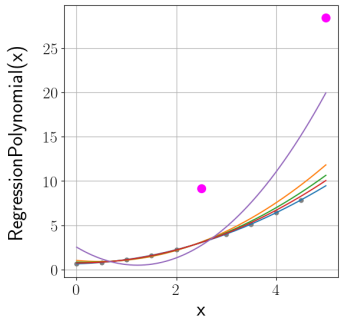


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
	Least Squares (rem. out.)	0.67	0.15	0.32
	Huber ($\gamma = 0.5$)	0.25	1.04	0.17
	Huber ($\gamma = 1.0$)	0.46	0.60	0.25
	Huber ($\gamma = 2.0$)	0.57	0.37	0.28
	Least Squares (stand.)	-0.65	2.58	-0.02

(b) Coefficients

Figure 17: graphical representation of one run of the experiment for $n = 2$, 2 outliers, noise = 0.03

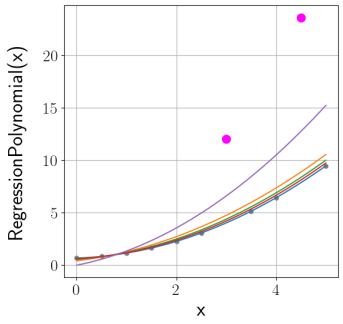


(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
	Least Squares (rem. out.)	0.67	0.15	0.32
	Huber ($\gamma = 0.5$)	1.05	-0.49	0.53
	Huber ($\gamma = 1.0$)	0.86	-0.17	0.43
	Huber ($\gamma = 2.0$)	0.76	-0.01	0.37
	Least Squares (stand.)	2.54	-3.32	1.36

(b) Coefficients

Figure 18: graphical representation of one run of the experiment for $n = 2$, 2 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2
	Initial	0.67	0.15	0.32
	Least Squares (rem. out.)	0.67	0.15	0.32
	Huber ($\gamma = 0.5$)	0.41	0.57	0.29
	Huber ($\gamma = 1.0$)	0.54	0.36	0.31
	Huber ($\gamma = 2.0$)	0.60	0.26	0.31
	Least Squares (stand.)	-0.01	0.93	0.42

(b) Coefficients

Table 16: ratio calculated for 10^6 runs of the experiment for $n = 2$, 2 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	2.73	0.15	1.18
Huber ($\gamma = 1.0$)	3.75	0.08	1.45
Huber ($\gamma = 2.0$)	0.65	0.04	2.65
Least Squares (stand.)	0.19	0.02	1.99

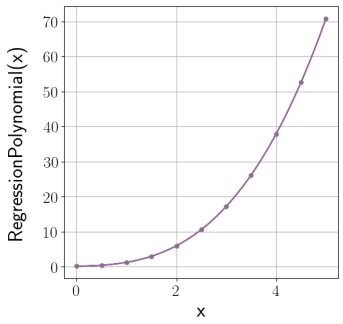
Table 17: ratio calculated for 10^6 runs of the experiment for $n = 2$, 2 outliers, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	1.34	0.39	1.02
Huber ($\gamma = 1.0$)	2.05	0.24	1.03
Huber ($\gamma = 2.0$)	39.62	0.14	1.07
Least Squares (stand.)	0.50	0.08	0.73

Table 18: ratio calculated for 10^6 runs of the experiment for $n = 2$, 2 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	1.56	0.20	1.12
Huber ($\gamma = 1.0$)	3.50	0.11	1.28
Huber ($\gamma = 2.0$)	2.34	0.06	1.77
Least Squares (stand.)	0.22	0.02	6.78

Figure 19: graphical representation of one run of the experiment for $n = 3$, 0 outliers, noise = 0.01

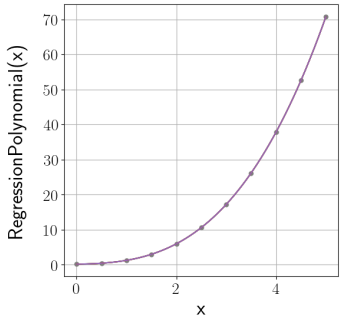


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
	Huber ($\gamma = 0.5$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 1.0$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 2.0$)	0.18	0.24	0.37	0.48
	Least Squares (stand.)	0.18	0.24	0.37	0.48

(b) Coefficients

Figure 20: graphical representation of one run of the experiment for $n = 3$, 0 outliers, noise = 0.03

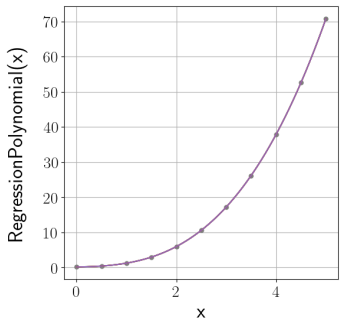


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
	Huber ($\gamma = 0.5$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 1.0$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 2.0$)	0.18	0.24	0.37	0.48
	Least Squares (stand.)	0.18	0.24	0.37	0.48

(b) Coefficients

Figure 21: graphical representation of one run of the experiment for $n = 3$, 0 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
	Huber ($\gamma = 0.5$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 1.0$)	0.18	0.24	0.37	0.48
	Huber ($\gamma = 2.0$)	0.18	0.24	0.37	0.48
	Least Squares (stand.)	0.18	0.24	0.37	0.48

(b) Coefficients

Table 19: ratio calculated for 10^6 runs of the experiment for $n = 3$, 0 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00	1.00

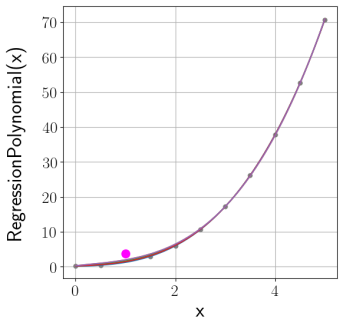
Table 20: ratio calculated for 10^6 runs of the experiment for $n = 3$, 0 outliers, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00	1.00

Table 21: ratio calculated for 10^6 runs of the experiment for $n = 3$, 0 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Huber ($\gamma = 0.5$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 1.0$)	1.00	1.00	1.00	1.00
Huber ($\gamma = 2.0$)	1.00	1.00	1.00	1.00
Least Squares (stand.)	1.00	1.00	1.00	1.00

Figure 22: graphical representation of one run of the experiment for $n = 3$, 1 outlier, noise = 0.01

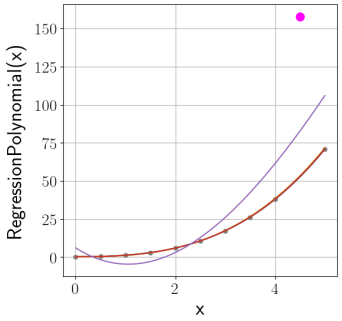


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.32	1.40	-0.29	0.57
—	Huber ($\gamma = 1.0$)	0.25	0.88	0.01	0.53
—	Huber ($\gamma = 2.0$)	0.21	0.56	0.19	0.50
—	Least Squares (stand.)	0.32	1.40	-0.29	0.57

(b) Coefficients

Figure 23: graphical representation of one run of the experiment for $n = 3$, 1 outlier, noise = 0.03

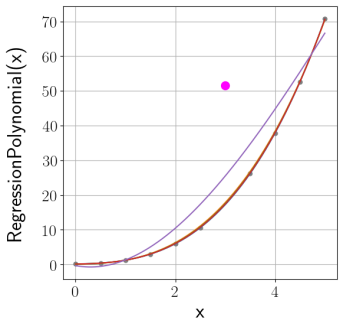


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.33	-0.32	0.65	0.45
—	Huber ($\gamma = 1.0$)	0.25	-0.04	0.51	0.47
—	Huber ($\gamma = 2.0$)	0.21	0.10	0.44	0.47
—	Least Squares (stand.)	6.06	-20.83	10.66	-0.50

(b) Coefficients

Figure 24: graphical representation of one run of the experiment for $n = 3$, 1 outlier, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.14	0.02	0.69	0.42
—	Huber ($\gamma = 1.0$)	0.16	0.13	0.53	0.45
—	Huber ($\gamma = 2.0$)	0.17	0.19	0.45	0.47
—	Least Squares (stand.)	-0.31	-2.62	4.55	-0.27

(b) Coefficients

Table 22: ratio calculated for 10^6 runs of the experiment for $n = 3$, 1 outlier, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	1.12	1.13	0.74	0.76
Huber ($\gamma = 1.0$)	1.28	1.30	0.58	0.28
Huber ($\gamma = 2.0$)	1.77	1.85	0.41	0.12
Least Squares (stand.)	4.66	3.55	0.20	0.04

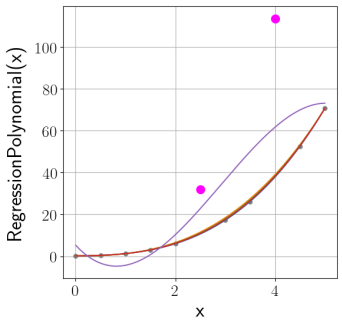
Table 23: ratio calculated for 10^6 runs of the experiment for $n = 3$, 1 outlier, noise = 0.03

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.39	1.65	0.76	3.66
Huber ($\gamma = 1.0$)	0.39	1.65	0.76	3.66
Huber ($\gamma = 2.0$)	0.39	1.65	0.76	3.66
Least Squares (stand.)	0.39	1.65	0.76	3.66

Table 24: ratio calculated for 10^6 runs of the experiment for $n = 3$, 1 outlier, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.68	1.98	0.57	0.33
Huber ($\gamma = 1.0$)	0.52	123.73	0.40	0.14
Huber ($\gamma = 2.0$)	0.35	1.02	0.25	0.07
Least Squares (stand.)	0.14	0.18	0.09	0.02

Figure 25: graphical representation of one run of the experiment for $n = 3$, 2 outliers, noise = 0.01

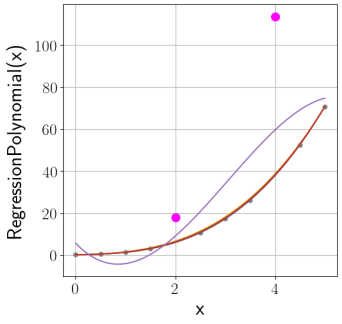


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.22	-0.40	1.05	0.37
—	Huber ($\gamma = 1.0$)	0.20	-0.08	0.71	0.42
—	Huber ($\gamma = 2.0$)	0.19	0.08	0.54	0.45
—	Least Squares (stand.)	5.27	-25.92	18.43	-2.11

(b) Coefficients

Figure 26: graphical representation of one run of the experiment for $n = 3$, 2 outliers, noise = 0.03

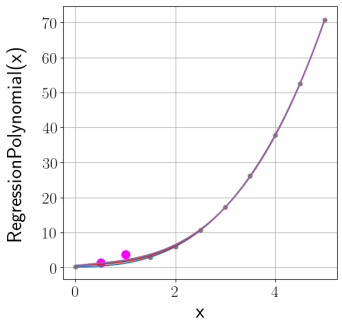


(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.12	0.41	0.54	0.44
—	Huber ($\gamma = 1.0$)	0.15	0.33	0.46	0.46
—	Huber ($\gamma = 2.0$)	0.16	0.28	0.41	0.47
—	Least Squares (stand.)	5.63	-24.84	16.96	-1.85

(b) Coefficients

Figure 27: graphical representation of one run of the experiment for $n = 3$, 2 outliers, noise = 0.10



(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
	Initial	0.18	0.24	0.37	0.48
—	Least Squares (rem. out.)	0.18	0.24	0.37	0.48
—	Huber ($\gamma = 0.5$)	0.62	1.33	-0.33	0.58
—	Huber ($\gamma = 1.0$)	0.57	0.94	-0.1	0.55
—	Huber ($\gamma = 2.0$)	0.53	0.62	0.08	0.52
—	Least Squares (stand.)	0.62	1.33	-0.33	0.58

(b) Coefficients

Table 25: ratio calculated for 10^6 runs of the experiment for $n = 3$, 2 outliers, noise = 0.01

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.24	1.66	0.85	1.00
Huber ($\gamma = 1.0$)	0.20	3.40	0.68	14.91
Huber ($\gamma = 2.0$)	0.17	3.40	0.48	0.49
Least Squares (stand.)	0.07	0.19	0.14	0.05

Table 26: ratio calculated for 10^6 runs of the experiment for $n = 3$, 2 outliers, noise = 0.03

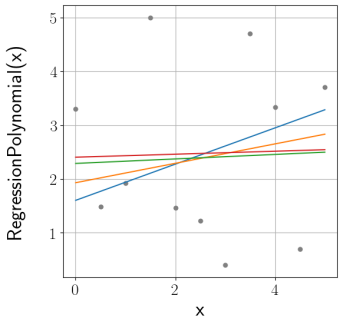
Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	0.23	0.07	0.05	0.01
Huber ($\gamma = 1.0$)	0.11	0.06	0.05	0.01
Huber ($\gamma = 2.0$)	0.06	0.06	0.04	0.01
Least Squares (stand.)	0.05	0.05	0.04	0.01

Table 27: ratio calculated for 10^6 runs of the experiment for $n = 3$, 2 outliers, noise = 0.10

Method	$\overline{\alpha_0^{init}/\alpha_0^{num}}$	$\overline{\alpha_1^{init}/\alpha_1^{num}}$	$\overline{\alpha_2^{init}/\alpha_2^{num}}$	$\overline{\alpha_3^{init}/\alpha_3^{num}}$
Least Squares (rem. out.)	1.00	1.00	1.00	1.00
Huber ($\gamma = 0.5$)	2.01	0.82	0.86	1.11
Huber ($\gamma = 1.0$)	256.20	0.70	0.75	0.36
Huber ($\gamma = 2.0$)	0.99	0.53	0.60	0.15
Least Squares (stand.)	0.58	0.67	0.13	0.03

5.2.2 Random Points

Figure 28: graphical representation of one run of the experiment for $n = 1$

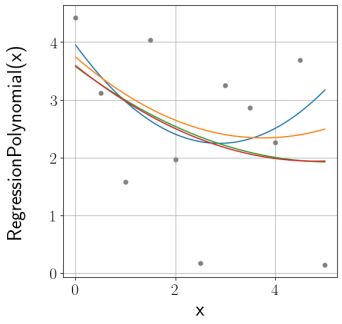


(a) Plot

Colour	Method	α_0	α_1
—	Huber ($\gamma = 0.5$)	1.60	0.34
—	Huber ($\gamma = 1.0$)	1.93	0.18
—	Huber ($\gamma = 2.0$)	2.29	0.04
—	Least Squares (stand.)	2.40	0.03

(b) Coefficients

Figure 29: graphical representation of one run of the experiment for $n = 2$

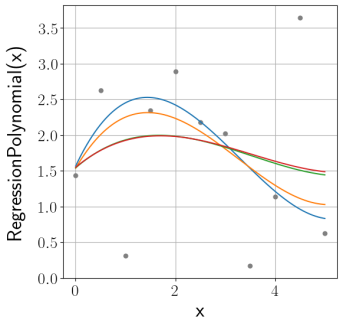


(a) Plot

Colour	Method	α_0	α_1	α_2
—	Huber ($\gamma = 0.5$)	3.95	-1.18	0.20
—	Huber ($\gamma = 1.0$)	3.74	-0.74	0.10
—	Huber ($\gamma = 2.0$)	3.57	-0.64	0.06
—	Least Squares (stand.)	3.59	-0.69	0.07

(b) Coefficients

Figure 30: graphical representation of one run of the experiment for $n = 3$



(a) Plot

Colour	Method	α_0	α_1	α_2	α_3
—	Huber ($\gamma = 0.5$)	1.57	1.48	-0.66	0.07
—	Huber ($\gamma = 1.0$)	1.54	1.19	-0.53	0.05
—	Huber ($\gamma = 2.0$)	1.54	0.61	-0.24	0.02
—	Least Squares (stand.)	1.55	0.58	-0.22	0.02

(b) Coefficients

5.3 Interpretation

First, we look at the case of data points from a random polynomial without outliers. We observe that all methods produce the same results, regardless of the degree of the regression polynomial or the amount of noise.

For the cases of 1 and 2 outliers, we observe that the regression polynomial obtained using Least Squares (without removing the outlier(s)) is more affected by them than those obtained using Huber. Least Squares, while removing the outliers, fits the initial coefficients best. But this also means not considering the outlier(s) (at all), which might be suitable if they do not carry information. To reduce their impact to a proportionate level regarding their carried information, Huber seems suitable in being a compromise between using Least Squares with the full, maybe even disproportionate, impact of the outliers or none.

Within the different Huber approaches, we notice that larger γ values lead to polynomials that are more similar to Least Squares (the two may become indistinguishable in the plots when γ is large enough). It occurs because, in the case of increasing gamma values, the Huber Cost Function[1] (with a factor of 2) converges pointwise to the Least Squares Cost Function. Hence, also the resulting objective function converges pointwise. Smaller γ values lead to coefficients closer to the initial ones. How close they should be, needs to be chosen while choosing γ.

But choosing γ remains an unsolved question. A solution may be knowing the data and its range and choosing γ by that knowledge. An outlier can not be defined in general, only within the context.

Looking at the regression polynomials calculated for the random data points, one can see that γ does have a noticeable impact. So in cases where points may lay in a huge range, and none of them should be considered an outlier, Huber may not be suitable. Choosing a huge γ and using Huber would lead to basically applying Least Squares. Doing so would not lead to a benefit while taking the risk of choosing γ too small.

In summary, as was expected by the formulae and underlined by the experiments, Huber seems like a suitable regression approach while dealing with outliers (not random data points) if knowing the data and choosing a suitable γ by its means is possible.

6 API

In this section, it is demonstrated how to use the solving methods defined in Solvers_LeastSquares.py and Solvers_Huber.py through the interface defined in Solvers_Interface.py. An example file is presented that explains the most commonly used functionalities.

The solving methods used in this example rely on the qpOASES[3] algorithm, which must be installed to proceed. It functions through an iterative process, which necessitates the specification of a parameter indicating the maximum number of iterations required when invoking `solve`. No information concerning this, and the execution of qpOASES[3] in general, will be displayed to the user. In particular, the user cannot see if the set limit proves adequate for the algorithm to derive a solution. However, if you need information regarding the operation of qpOASES[3], it is necessary to remove the `options.printLevel = PrintLevel.NONE` command from the Solvers_LeastSquares.py and Solvers_Huber.py files.

When using the `RegressionProblem_Huber` class the second parameter that has to be chosen is γ. This parameter defines the extent to which the Huber cost function takes outliers into account or, more precisely, the range in which it considers errors with a quadratic weight. The algorithm does not provide a γ; therefore, it has to be chosen by the user.

The regression polynomial and its coefficients are member variables within the Regression Problem classes. They change after each time `solve` is called and are returned by the method. Before the first call of `solve`, the regression polynomial is 0, and the coefficients are an empty array.

The Regression Problem classes include a `plot`, `show`, and `show_and_save` method. These methods use matplotlib in the expected way. The graphical output shows all parameters used while solving the last time. However, the user should consider whether this integrated graphic demonstration functionality is sufficient for them.

6.1 Solvers_example.py

```
# Import the numpy library.
import numpy as np

# Import the Regression Problem classes from the Solvers_Interface module.
from Solvers.Solvers_Interface import RegressionProblem_LeastSquare
from Solvers.Solvers_Interface import RegressionProblem_Huber

# Initialise a point cloud with evenly spaced points along the x- and y-axes.
X_points = np.linspace(0, 10, 11)
Y_points = np.linspace(0, 10, 11)

# Add an outlier to Y_points to demonstrate robust regression.
Y_points[10] = 1

# Initialise a Regression Problem to be solved with Least Square method.
RP_LQ = RegressionProblem_LeastSquare(X_points, Y_points)

# Initialise a Regression Problem to be solved with Huber method.
RP_Hu = RegressionProblem_Huber(X_points, Y_points)

# Solve the Regression Problems with their respective methods.
# RP_LQ.solve(degree of the regression polynomial, maximum number of iterations of qpOASES)
poly_LQ, poly_LQ_coefficients = RP_LQ.solve(1, 100)

# RP_Hu.solve(degree of the regression polynomial, maximum number of iterations of qpOASES, gamma)
poly_Hu, poly_Hu_coefficients = RP_Hu.solve(1, 100, 1)

# Plot the regression curves for both Regression Problems.
RP_LQ.plot()
RP_Hu.plot()

# Show and save the plot as an image file with the specified name.
RP_LQ.show_and_save("some_nice_regression_polynomials")  # or RP_Hu.show_and_save()
```

Listing 1: code snippet of Solvers_example.py to show the use of Solvers_Interface.py

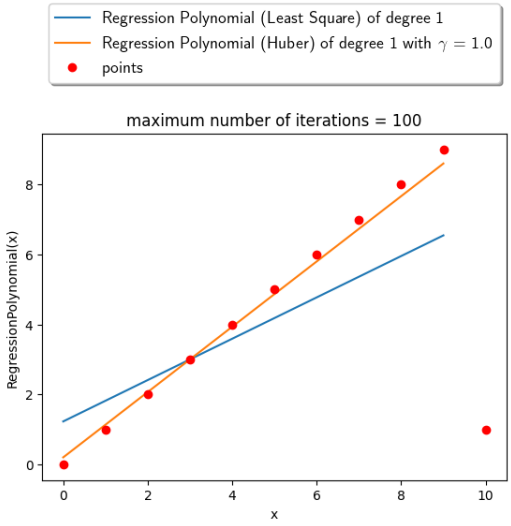


Figure 31: Output of Solvers_example.py

7 Conclusion

In conclusion, while Least Squares regression could be considered the most widely used approach for modelling data in many situations, it may not always be the best choice. When confronted with outliers, the Least Squares method finds its limitations due to its cost function, which weights all errors quadratically.

In such cases, an approach using the Huber cost function[1] can improve the accuracy of the regression model in weighting the errors caused by outliers less. The shown alternative of removing the outliers from the data set and then applying Least Squares does not convince in every case because it would mean not weighting their error, respectively, their carried information at all.

It is due to the data set, if and how outliers should be taken into account. The method needs to be chosen considering these thoughts. Unfortunately, it remains an open question what an outlier is in each context. These effects both scenarios: whether it is removed or taken less into account by choosing γ.

A Mathematical Methods

Definition A.0.1 (Minimisation Problem)**.**
We write a Minimisation Problem as follows:

$$f(x_1, ..., x_n) \rightarrow \min_{x_1,...,x_n}$$

s.t.

$$x_1 \in M_1, ..., x_n \in M_n$$

The goal is to determine:

$$\underset{x_1,...,x_n}{argmin} f = \{(\hat{x_1}, ..., \hat{x_n}) \in \prod_{i=1}^{n} M_i | \forall (x_1, ..., x_n) \in \prod_{i=1}^{n} M_i : f(\hat{x_1}, ..., \hat{x_n}) \leq f(x_1, ..., x_n)\}$$

Therefore:

$$\min_{x_1,...,x_n} f = f((\hat{x_1}, ..., \hat{x_n})) \quad \forall (\hat{x_1}, ..., \hat{x_n}) \in \underset{x_1,...,x_n}{argmin} f$$

Definition A.0.2 (Equal Minimisation Problems)**.**
Two Minimisation Problems

$$f(x_1, ..., x_n) \rightarrow \min_{x_1,...,x_n}$$

$$g(y_1, ..., y_m) \rightarrow \min_{y_1,...,y_m}$$

are equal if and only if:

$$\underset{x_1,...,x_n}{argmin} f = \underset{y_1,...,y_m}{argmin}\, g$$

We remark that this is an equivalence relation and note it with $=$.

Definition A.0.3 (Equivalent Minimisation Problems)**.**
Two Minimisation Problems

$$f(x_1, ..., x_n) \rightarrow \min_{x_1,...,x_n}$$

$$g(y_1, ..., y_m) \rightarrow \min_{y_1,...,y_m}$$

are equivalent if and only if there exists a bijection:

$$\Phi : \underset{x_1,...,x_n}{argmin} f \leftrightarrow \underset{y_1,...,y_m}{argmin}\, g$$

We remark that this is an equivalence relation and note it with $\sim$.

The following definitions and lemmata concerning the Subgradient are derived from here[10].

Definition A.0.4 (Subgradient)**.**
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function (not necessarily differentiable).
Then we call $g \in \mathbb{R}^n$ a subgradient of f in x_0 if:

$$\forall x \in \mathbb{R}^n : f(x) \geq f(x_0) + < g, x - x_0 >$$

We denote $\delta f(x_0)$=$\{$g $\in \mathbb{R}^n$|g is subgradient of f in $x_0\}$.

Lemma A.0.1 (Calculation of specific subgradients)**.**
(i) f differentiable in $x_0 \in$ *int dom*f*:* $\delta f(x_0) = \{\nabla f(x_0)\}$

(ii) $f : \mathbb{R} \to \mathbb{R}, x \mapsto |x|$*:* $\delta f(x_0) = \begin{cases} \{-1\} & x_0 < 0 \\ [-1, 1] & x_0 = 0 \\ \{1\} & x_0 > 0 \end{cases}$

Lemma A.0.2 (The zero-subgradient necessary and sufficient optimality condition)**.**
Let $f : \mathbb{R}^n \to \mathbb{R}$ *be a convex function (not necessarily differentiable), then:*

$$x_0 \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f \Leftrightarrow \exists 0 \in \delta f(x_0)$$

B Code

The code can also be found on GitHub:
https://github.com/gantz-thomas-gantz/BachelorThesisMathematics

The programming language chosen is Python, for its ease of reading and writing, flexibility and simplicity in visually representing.

B.1 Experiments_GraphicalRepresentation.py

```
import numpy as np
import random
import matplotlib.pyplot as plt
from Solvers.Solvers_Interface import
    RegressionProblem_LeastSquare
from Solvers.Solvers_Interface import RegressionProblem_Huber

# Maximum degree of regression polynomial.
for n in [1, 2, 3]:

    # Points along a polynomial with a different amount of
    outliers (1st experiment).
    # Random polynomial of given maximum degree.
    coefficients = []
    for i in range(n+1):
        coefficients.append(random.random())

    def Polynomial(x,coefficients):
        y = 0
        for i in range(len(coefficients)):
            y += coefficients[i]*x**i
        return y

    # Different levels of noise.
    for noise_percentage in [0.01, 0.03, 0.1]:

        # Making the data points.
        X_points = np.linspace(0, 5, 11)
        X = np.linspace(np.min(X_points), np.max(X_points),
    1000)
        Y_points_initial = np.array(Polynomial(X_points,
    coefficients))

        # Adding noise.
        noise = np.random.normal(0, noise_percentage, len(
    Y_points_initial))
```

```
32          Y_points_noisy = Y_points_initial + noise
33
34          # Different amount of outliers and plotting the
    points.
35          for number_of_outliers in [0,1,2]:
36              fig = plt.figure(figsize=(6, 6))
37              ax1 = fig.add_subplot(111)
38              fig_table = plt.figure(figsize=(15, 5))
39              ax2 = fig_table.add_subplot(111)
40              plt.rc('text', usetex=True)
41              Y_points = Y_points_initial.copy()
42
43              if (number_of_outliers==0):
44
45                  # "Removing" the outlier for one data set.
46                  X_points_without_outliers = X_points
47                  Y_points_without_outliers = Y_points
48
49                  # Plotting the points.
50                  ax1.plot(X_points, Y_points, marker='o',
    markersize=5, color='gray', linestyle='None', label="
    points")
51
52              if (number_of_outliers==1):
53
54                  # Adding an outlier.
55                  random_index_1 = random.randint(0, len(
    X_points)-1)
56                  Y_points[random_index_1] *= 3
57
58                  # Removing the outlier for one data set.
59                  X_points_without_outliers = np.delete(
    X_points, random_index_1)
60                  Y_points_without_outliers = np.delete(
    Y_points, random_index_1)
61
62                  # Plotting the points.
63                  ax1.plot(X_points, Y_points, marker='o',
    markersize=5, color='gray', linestyle='None', label="
    points")
64                  ax1.plot([X_points[random_index_1]], [
    Y_points[random_index_1]], marker='o', markersize=10,
    color='magenta', linestyle='None', label="outliers")
65
66              if (number_of_outliers==2):
67
68                  # Adding 2 outliers
69                  random_index_1 = random.randint(0, len(
    X_points)-1)
```

```
70                  Y_points[random_index_1] *= 3
71                  random_index_2 = random.randint(0, len(
    X_points)-1)
72                  while random_index_2 == random_index_1:
73                      random_index_2 = random.randint(0, len(
    X_points)-1)
74                  Y_points[random_index_2] *= 3
75
76                  # Removing the outliers for one data set.
77                  X_points_without_outliers = np.delete(
    X_points, [random_index_1, random_index_2])
78                  Y_points_without_outliers = np.delete(
    Y_points, [random_index_1, random_index_2])
79
80                  # Plotting the points
81                  ax1.plot(X_points, Y_points, marker='o',
    markersize=5, color='gray', linestyle='None', label="
    points")
82                  ax1.plot([X_points[random_index_1], X_points[
    random_index_2]], [Y_points[random_index_1], Y_points[
    random_index_2]], marker='o', markersize=10, color='
    magenta', linestyle='None', label="outliers")
83
84          # Making the table.
85          data = []
86          data.append(["Initial", [round(x, 2) for x in
    coefficients]])
87
88          # Solving with Least Squares and removed outliers
    .
89          RP = RegressionProblem_LeastSquare(
    X_points_without_outliers,Y_points_without_outliers)
90          RegressionPolynomial, RegressionCoefficients = RP
    .solve(n,1000)
91
92          data.append(["Least Squares (rem. out.)", [round(
    x, 2) for x in RegressionCoefficients]])
93          Y = RegressionPolynomial(X)
94          ax1.plot(X,Y, label="Least Squares (rem. out.)")
95
96          # Solving with Huber and different gammas.
97          for gamma in [2,1,0.5]:
98
99              RP = RegressionProblem_Huber(X_points,
    Y_points)
100             RegressionPolynomial, RegressionCoefficients
    = RP.solve(n,1000,gamma)
101
```

```
102                 data.append([r'Huber ($\gamma$ = %1.1f)' %
    gamma, [round(x, 2) for x in RegressionCoefficients]])
103
104                 Y = RegressionPolynomial(X)
105                 ax1.plot(X, Y, label=r'Huber ($\gamma$ = %1.1
    f)' % gamma)
106
107             # Solving with Least Squares.
108             RP = RegressionProblem_LeastSquare(X_points,
    Y_points)
109             RegressionPolynomial, RegressionCoefficients = RP
    .solve(n,1000)
110
111             data.append(["Least Squares (stand.)", [round(x,
    2) for x in RegressionCoefficients]])
112             Y = RegressionPolynomial(X)
113             ax1.plot(X,Y, label="Least Squares (stand.)")
114
115             # Setting options for the table.
116             ax2.axis('off')
117             table = ax2.table(cellText=data, colLabels=['
    Label', 'Coefficients'], loc='center')
118             plt.tight_layout()
119
120             # Setting options for the graph.
121             ax1.set_xlabel('x', fontsize=25)
122             ax1.set_ylabel('RegressionPolynomial(x)',
    fontsize=25)
123             ax1.tick_params(axis='x', labelsize=20)
124             ax1.tick_params(axis='y', labelsize=20)
125             ax1.yaxis.set_label_coords(-0.135, 0.5)
126             ax1.grid(True)
127
128             # Setting options for the figure and saving it.
129             plt.tight_layout()
130             table_filename = "n=%i,outliers:%s,
    noise_percentage=%f" % (n, number_of_outliers,
    noise_percentage) + "_table.png"
131             graph_filename = "n=%i,outliers:%s,
    noise_percentage=%f" % (n, number_of_outliers,
    noise_percentage) + "_graph.png"
132             fig.savefig(graph_filename, bbox_inches='tight')
133             fig_table.savefig(table_filename, bbox_inches='
    tight')
134
135             # Closing the figures to free memory.
136             plt.close(fig)
137             plt.close(fig_table)
138
```

```
139     # Making random points and plotting them (2nd experiment)
140     fig = plt.figure(figsize=(6, 6))
141     ax1 = fig.add_subplot(111)
142     fig_table = plt.figure(figsize=(15, 5))
143     ax2 = fig_table.add_subplot(111)
144     plt.rc('text', usetex=True)
145 
146     for i in range(len(X_points)):
147         Y_points[i] = random.uniform(0, 5)
148 
149     ax1.plot(X_points, Y_points, marker='o', markersize=5,
    color='gray', linestyle='None', label="points")
150 
151     # Making the table.
152     data = []
153 
154     # Solving with Huber and different gammas.
155     for gamma in [0.5, 1.0, 2.0]:
156 
157         RP = RegressionProblem_Huber(X_points,Y_points)
158         RegressionPolynomial, RegressionCoefficients = RP.
    solve(n,1000,gamma)
159 
160         data.append([r'Huber ($\gamma$ = %1.1f)' % gamma, [
    round(x, 2) for x in RegressionCoefficients]])
161 
162         Y = RegressionPolynomial(X)
163         ax1.plot(X, Y, label=r'Huber ($\gamma$ = %1.1f)' %
    gamma)
164 
165     # Solving with Least Squares.
166     RP = RegressionProblem_LeastSquare(X_points,Y_points)
167     RegressionPolynomial, RegressionCoefficients = RP.solve(n
    ,1000)
168 
169     data.append(["Least Squares (stand.)", [round(x, 2) for x
     in RegressionCoefficients]])
170     Y = RegressionPolynomial(X)
171     ax1.plot(X,Y, label="Least Squares (stand.)")
172 
173     # Setting options for the table.
174     ax2.axis('off')
175     table = ax2.table(cellText=data, colLabels=['Label', '
    Coefficients'], loc='center')
176     plt.tight_layout()
177 
178     # Setting options for the graph.
179     ax1.set_xlabel('x', fontsize=25)
180     ax1.set_ylabel('RegressionPolynomial(x)', fontsize=25)
```

```
181     ax1.tick_params(axis='x', labelsize=20)
182     ax1.tick_params(axis='y', labelsize=20)
183     ax1.yaxis.set_label_coords(-0.135, 0.5)
184     ax1.grid(True)
185 
186     # Setting options for the figure and saving it.
187     plt.tight_layout()
188     table_filename = "random,n=%i" % n + "_table.png"
189     graph_filename = "random,n=%i" % n + "_graph.png"
190     fig.savefig(graph_filename, bbox_inches='tight')
191     fig_table.savefig(table_filename, bbox_inches='tight')
192 
193     # Closing the figures to free memory.
194     plt.close(fig)
195     plt.close(fig_table)
```

Listing 2: code snippet of Experiments_GraphicalRepresentation.py where all the sample calculations that concern the graphical representation of one run of the experiment shown in subsection 5.2 are obtained from

B.2 Experiments_Ratio.py

```
import numpy as np
import random
import matplotlib.pyplot as plt
from Solvers.Solvers_Interface import
    RegressionProblem_LeastSquare
from Solvers.Solvers_Interface import RegressionProblem_Huber

number_of_iterations = 1000000

# Maximum degree of regression polynomial.
for n in [1,2,3]:

    # Points along a polynomial with a different amount of
    outliers (1st experiment).
    # Random polynomial of given maximum degree.
    coefficients = []
    for i in range(n+1):
        coefficients.append(random.random())

    def Polynomial(x,coefficients):
        y = 0
        for i in range(len(coefficients)):
            y += coefficients[i]*x**i
        return y

    # Different levels of noise.
    for noise_percentage in [0.01,0.03,0.1]:
        # Making the data points.
        X_points = np.linspace(0, 5, 11)
        X = np.linspace(np.min(X_points), np.max(X_points),
    1000)
        Y_points_initial = np.array(Polynomial(X_points,
    coefficients))

        # Adding noise.
        noise = np.random.normal(0, noise_percentage, len(
    Y_points_initial))
        Y_points_noisy = Y_points_initial + noise

        # Different amount of outliers and plotting the
    points.
        for number_of_outliers in [0,1,2]:
            fig = plt.figure(figsize=(10, 6))
            plt.rc('text', usetex=True)
            Y_points = Y_points_initial.copy()

            if (number_of_outliers==0):
```

```
42
43                  # "Removing" the outlier for one data set.
44                  X_points_without_outliers = X_points
45                  Y_points_without_outliers = Y_points
46
47              if (number_of_outliers==1):
48
49                  # Adding an outlier.
50                  random_index_1 = random.randint(0, len(
    X_points)-1)
51                  Y_points[random_index_1] *= 3
52
53                  # Removing the outlier for one data set.
54                  X_points_without_outliers = np.delete(
    X_points, random_index_1)
55                  Y_points_without_outliers = np.delete(
    Y_points, random_index_1)
56
57              if (number_of_outliers==2):
58
59                  # Adding 2 outliers
60                  random_index_1 = random.randint(0, len(
    X_points)-1)
61                  Y_points[random_index_1] *= 3
62                  random_index_2 = random.randint(0, len(
    X_points)-1)
63                  while random_index_2 == random_index_1:
64                      random_index_2 = random.randint(0, len(
    X_points)-1)
65                  Y_points[random_index_2] *= 3
66
67                  # Removing the outliers for one data set.
68                  X_points_without_outliers = np.delete(
    X_points, [random_index_1, random_index_2])
69                  Y_points_without_outliers = np.delete(
    Y_points, [random_index_1, random_index_2])
70
71              # Making the table.
72              data = []
73
74              # Solving with Least Squares and removed outliers
    .
75              data_temp = np.zeros(n+1)
76              for i in range(number_of_iterations):
77                  RP = RegressionProblem_LeastSquare(
    X_points_without_outliers,Y_points_without_outliers)
78                  RegressionPolynomial, RegressionCoefficients
    = RP.solve(n,1000)
79                  data_temp += np.abs(coefficients/
```

```
    RegressionCoefficients)
80
81              data_temp = data_temp/number_of_iterations
82              data.append(["Least Squares (rem. out.)", [round(
    x, 2) for x in data_temp]])
83
84              # Solving with Huber and different gammas.
85              for gamma in [0.5, 1.0, 2.0]:
86
87                  data_temp = np.zeros(n+1)
88                  for i in range(number_of_iterations):
89
90                      RP = RegressionProblem_Huber(X_points,
    Y_points)
91                      RegressionPolynomial,
    RegressionCoefficients = RP.solve(n,1000,gamma)
92                      data_temp += np.abs(coefficients/
    RegressionCoefficients)
93
94                  data_temp = data_temp/number_of_iterations
95                  data.append([r'Huber ($\gamma$ = %1.1f)' %
    gamma, [round(x, 2) for x in data_temp]])
96
97              # Solving with Least Squares.
98              data_temp = np.zeros(n+1)
99              for i in range(number_of_iterations):
100
101                 RP = RegressionProblem_LeastSquare(X_points,
    Y_points)
102                 RegressionPolynomial, RegressionCoefficients
    = RP.solve(n,1000)
103                 data_temp += np.abs(coefficients/
    RegressionCoefficients)
104
105             data_temp = data_temp/number_of_iterations
106             data.append(["Least Squares (stand.)", [round(x,
    2) for x in data_temp]])
107
108             # Setting options for the table.
109             table = plt.table(cellText=data, colLabels=["
    Method", "InitialCoefficients/NumericalCoefficients"], loc
    ='center')
110             table.scale(2, 3.86)
111             table.auto_set_font_size(False)
112             table.set_fontsize(22)
113             table.auto_set_column_width([0, 1])
114             plt.axis('off')
115
116             # Setting options for the figure and saving it.
```

```
117             plt.tight_layout()
118             plt.savefig("
    InitialCoefficients_ratio_NumericalCoefficients_n=%i,
    outliers:%s,noise_percentage=%f" % (n, number_of_outliers,
     noise_percentage) + ".png", bbox_inches='tight')
119             plt.close(fig)
```

Listing 3: code snippet of Experiments_Ratio.py where all the sample calculations that concern the ratio calculated for 10^6 runs of the experiment shown in subsection 5.2 are obtained from

B.3 Solvers_Interface.py

```
from abc import ABCMeta, abstractmethod
import matplotlib.pyplot as plt
import numpy as np
from Solvers.Solvers_LeastSquare import
    Solver_LeastSquare_PolynomialRegression
from Solvers.Solvers_Huber import
    Solver_Huber_PolynomialRegression

class RegressionProblem:
    __metaclass__ = ABCMeta

    # member variables.
    X_points = []
    Y_points = []
    ProblemClass = ""

    used_n = -1
    used_max_number_of_iterations = -1

    def f(x):
        return 0
    RegressionPolynomial = f

    PolynmialCoefficients = []

    # constructor.
    def __init__(self, X_points, Y_points):
        self.X_points = X_points
        self.Y_points = Y_points

    # member methods.
    @abstractmethod
    def solve(self,max_number_of_iterations):
        pass

    @abstractmethod
    def plot(self):
        pass

    def show(self):
        plt.plot(self.X_points, self.Y_points, 'ro', label="
    points")

        # naming the x axis.
        plt.xlabel('x')
        # naming the y axis.
        plt.ylabel('RegressionPolynomial(x)')
```

```
45
46          # giving a title to my graph.
47          plt.title("maximum number of iterations = %i" % self.
    used_max_number_of_iterations)
48          # showing the legend.
49          plt.legend(loc='upper center', bbox_to_anchor=(0.5,
    1.4), fancybox=True, shadow=True, ncol=1, fontsize=13)
50
51          # adding grid.
52          plt.grid(True)
53
54          # function to show the plot.
55          plt.show()
56
57      def show_and_save(self,picturename):
58          plt.plot(self.X_points, self.Y_points, 'ro', label="
    points")
59
60          # naming the x axis.
61          plt.xlabel('x')
62          # naming the y axis.
63          plt.ylabel('RegressionPolynomial(x)')
64
65          # giving a title to my graph.
66          plt.title("maximum number of iterations = %i" % self.
    used_max_number_of_iterations)
67
68          # showing the legend.
69          plt.legend(loc='upper center', bbox_to_anchor=(0.5,
    1.4), fancybox=True, shadow=True, ncol=1, fontsize=13)
70          plt.savefig(picturename+".png", bbox_inches='tight')
71
72          # adding grid.
73          plt.grid(True)
74
75          # function to show the plot.
76          plt.show()
77
78
79  class RegressionProblem_LeastSquare(RegressionProblem):
80
81      # member methods.
82      def solve(self,n,max_number_of_iterations):
83          self.used_n = n
84          self.used_max_number_of_iterations =
    max_number_of_iterations
85          self.RegressionPolynomial, self.PolynmialCoefficients
     = Solver_LeastSquare_PolynomialRegression(self.X_points,
    self.Y_points,n,max_number_of_iterations)
```

```
86          return self.RegressionPolynomial, self.
    PolynmialCoefficients
87
88      def plot(self):
89
90          # x axis values.
91          X = np.linspace(np.min(self.X_points),np.max(self.
    Y_points),1000)
92          # corresponding y axis values.
93          Y = self.RegressionPolynomial(X)
94
95          # plotting the regression polynomial.
96          plt.plot(X,Y, label="Regression Polynomial (Least
    Square) of degree %i" % self.used_n)
97
98  class RegressionProblem_Huber(RegressionProblem):
99
100     #member variables.
101     used_gamma = -1.
102
103     #member methods.
104     def solve(self,n,max_number_of_iterations,gamma):
105         self.used_n = n
106         self.used_max_number_of_iterations =
    max_number_of_iterations
107         self.used_gamma = gamma
108         self.RegressionPolynomial, self.PolynmialCoefficients
     = Solver_Huber_PolynomialRegression(self.X_points,self.
    Y_points,n,max_number_of_iterations,gamma)
109         return self.RegressionPolynomial, self.
    PolynmialCoefficients
110
111     def plot(self):
112
113         # x axis values.
114         X = np.linspace(np.min(self.X_points),np.max(self.
    Y_points),1000)
115         # corresponding y axis values.
116         Y = self.RegressionPolynomial(X)
117
118         # plotting the regression polynomial.
119         plt.rc('text', usetex=True)
120         plt.plot(X,Y, label=r'Regression Polynomial (Huber)
    of degree %i with $\gamma$ = %1.1f' % (self.used_n,self.
    used_gamma))
```

Listing 4: code snippet of Solvers_Interface.py which is used to call the solving methods defined in Solvers_LeastSquares.py and Solvers_Huber.py

B.4 Solvers_LeastSquares.py

```
# This file includes two solvers using the Least Squares method via qpOASES.
# The first one finds a most approximate solution for AX=B.
# The second one finds an approximative polynomial function for a given dataset.

# Import.
import numpy as np
from qpoases import PyQProblemB as QProblemB
from qpoases import PyOptions as Options
from qpoases import PyPrintLevel as PrintLevel

def Solver_LeastSquare_LSE(A, B, max_number_of_iterations):

    # Setup data of QP.

    H   = 2*np.transpose(A)@A
    nV = len(H)

    g   = (-2*B@A).ravel()
    lb  = np.full(nV,-np.inf)
    ub  = np.full(nV,np.inf)

    # Initializing QProblem object.

    QP = QProblemB(nV)

    # Setting and printing Options.

    options = Options()
    options.printLevel = PrintLevel.NONE
    QP.setOptions(options)

    # Setting up and solving QProblem object.

    nWSR = np.array([max_number_of_iterations]) # maximum iterations
    QP.init(H, g, lb, ub, nWSR)

    # Returning solution.

    xOpt = np.zeros(nV)
    QP.getPrimalSolution(xOpt)
    return xOpt

def Solver_LeastSquare_PolynomialRegression(X,Y,n, max_number_of_iterations):
```

```
44
45      B = Y
46      A = np.empty((len(X),n+1))
47      for i in range(0,len(X)):
48          X_row = np.empty(n+1)
49          for j in range(0,n+1):
50              X_row[j] = X[i]**j
51          A[i] = X_row
52
53      PolynomialCoefficients = Solver_LeastSquare_LSE(A,B,
    max_number_of_iterations)
54
55      def RegressionPolynomial(x):
56          value = 0
57          for i in range(0,len(PolynomialCoefficients)):
58              value += PolynomialCoefficients[i]*x**i
59          return value
60
61      return RegressionPolynomial, PolynomialCoefficients
```

Listing 5: code snippet of Solvers_LeastSquares.py that implements the solving methods for a linear system of equations and the regression problem using the Least Squares method

B.5 Solvers_Huber.py

```
# This file includes two solvers using the Huber cost function via qpOASES.
# The first one finds a most approximate solution for AX=B.
# The second one finds an approximative polynomial function for a given dataset.

# Import.
import numpy as np
from qpoases import PyQProblem as QProblem
from qpoases import PyOptions as Options
from qpoases import PyPrintLevel as PrintLevel

def Solver_Huber_LSE(A,B,max_number_of_iterations,gamma):

    d = len(A[1]) # d = X_size
    l = len(A)    # l = Z_size = T_size = B_size

    n = d + l + l
    m = l + l

    # Setup data of QP.

    H   = np.zeros((n,n))
    for i in range(d,d+l):
        H[i][i] = 1
    nV = len(H) # nV = n

    A_QP = np.concatenate((np.concatenate((A,-1*A),axis=0),np.concatenate((-1*np.identity(l),np.identity(l)),axis=0),np.concatenate((-1*np.identity(l),-1*np.identity(l)),axis=0)),axis=1)
    nC = len(A_QP) # nC = l + l

    g = np.zeros(n)
    for i in range(d+l,n):
        g[i] = gamma

    lb = np.full(nV,-np.inf)
    ub = np.full(nV,np.inf)

    lbA_QP = np.full(nV,-np.inf)
    ubA_QP = np.concatenate((B,-1*B),axis=None)

    # Initializing QProblem object.

    QP = QProblem(nV,nC)

```

```
43      # Setting and printing Options.
44
45      options = Options()
46      options.printLevel = PrintLevel.NONE
47      QP.setOptions(options)
48
49      # Setting up and solving QProblem object.
50
51      nWSR = np.array([max_number_of_iterations]) # maximum
    iterations
52      QP.init(H, g, A_QP, lb, ub, lbA_QP, ubA_QP, nWSR)
53
54      # Obtaining solution of the QP.
55
56      xOpt = np.zeros(nV)
57      QP.getPrimalSolution(xOpt)
58
59      # Returning actual searched X.
60      X_solution = xOpt[0:d]
61      return X_solution
62
63
64  def Solver_Huber_PolynomialRegression(X,Y,n,
    max_number_of_iterations,gamma):
65
66      B = Y
67      A = np.empty((len(X),n+1))
68      for i in range(0,len(X)):
69          X_row = np.empty(n+1)
70          for j in range(0,n+1):
71              X_row[j] = X[i]**j
72          A[i] = X_row
73
74      PolynomialCoefficients = Solver_Huber_LSE(A,B,
    max_number_of_iterations,gamma)
75
76      def RegressionPolynomial(x):
77          value = 0
78          for i in range(0,len(PolynomialCoefficients)):
79              value += PolynomialCoefficients[i]*x**i
80          return value
81
82      return RegressionPolynomial, PolynomialCoefficients
```

Listing 6: code snippet of Solvers_Huber.py that implements the solving methods for a linear system of equations and the regression problem using the Huber method

References

[1] P. J. Huber, "Robust estimation of a location parameter," *Annals of Statistics*, vol. 53, no. 1, pp. 73–101, 1964.

[2] O. L. Mangasarian and D. R. Musicant, "Robust linear and support vector regression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 9, pp. 1055–1060, 2000.

[3] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[4] K. Backhaus, *Multivariate Analysemethoden: Eine anwendungsorientierte Einführung*. Berlin: Springer, 2006.

[5] S. M. Stigler, *The History of Statistics: The Measurement of Uncertainty Before 1900*. Cambridge, MA: Belknap Press of Harvard University Press, 1986.

[6] A.-M. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes*. Paris: Firmin-Didot, 1805. Appendice sur la méthode des moindres quarrés, S. 72-80.

[7] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium auctore Carolo Friderico Gauss*. sumtibus Frid. Perthes et IH Besser, 1809.

[8] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, pp. 1–21, February 1969.

[9] G. S. Maddala, *Introduction to Econometrics*. New York: MacMillan, 2nd ed., 1992.

[10] Y. Singer, "Advanced optimization," *Lecture Notes, AM221 Lecture7. pdf*, 2016.

Danksagung

Zu aller erst möchte ich meiner Betreuerin, Prof. Ekaterina Kostina, meinen herzlichen Dank aussprechen. Durch die Bereitstellung dieses erstklassigen Themas und die vielen neuen Denkanstöße während der Bearbeitung konnte ich mich fachlich weiterentwickeln.

Ein besonderer Dank gebührt auch meiner Familie, die es mir überhaupt erst ermöglicht hat, mein Studium zu absolvieren. Darüber hinaus ist ihr persönlicher, durch viel Erfahrung geleiteter Rat für mich stets eine große Hilfe.

Meinen Freunden in Heidelberg möchte ich ebenso von Herzen danken. Sie waren während dieser Zeit eine unersetzliche Stütze, sowohl auf persönlicher als auch auf fachlicher Ebene. Leonard Benkendorff und Sebastian Preuß möchte ich dabei gan(t)z besonders hervorheben. Euch gebührt mein Dank für unzählige gemeinsame Tage und Nächte in Heidelberg und ganz Europa, sowie für die unvergesslichen Gespräche in der Sauna.

Zudem gilt herzlicher Dank Lars Jungerberg. Auf dich kann ich immer zählen.

ll meglio alla fine, un grazie anche a Elena. Il tuo amore mi rende migliore.

Eure Unterstützung hat mich stets motiviert und mir geholfen, meine Ziele zu erreichen. Ohne euch wäre dieser Abschnitt meines Lebens nicht so erfolgreich und erfüllend gewesen. Dafür bin ich von Herzen dankbar und werde diese Zeit immer in sehr guter Erinnerung behalten.

Danke an euch alle!

Eigenständigkeitserklärung

Hiermit versichere ich schriftlich mit eigenhändiger Unterschrift, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe. Ich erkläre, dass die übermittelte elektronische Version in Inhalt und Wortlaut der gedruckten Fassung entspricht und dass ich mit einer universitätsinternen Prüfung anhand einer Plagiatssoftware einverstanden bin.

Heidelberg, den 25.07.2023
Ort, Datum


Unterschrift