



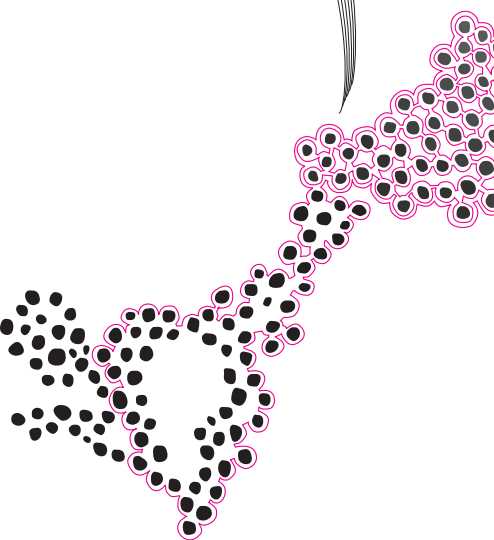
BSc Thesis Applied Mathematics

Local Optimization Algorithm for minimizing the Least Squares Criterion over Lipschitz Functions

Nadine Waninge

Supervisor:
Johannes Schmidt-Hieber
Petr Zamolodtchikov

September 6, 2023



Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

Local Optimization Algorithm on Lipschitz Least Squares

Nadine Waninge*

September 6, 2023

Abstract

In this article, we consider the basic regression model and the least squares estimator over the Lipschitz-1 functions. Two algorithms that approximate a function based on an i.i.d. sample are compared. The first algorithm utilizes a quadratic programming (QP) approach, where the objective function corresponds to the least squares estimator (LSE), and the constraints are determined by the Lipschitz condition. Beliakov's algorithm, which reduces the time complexity of the QP, is implemented and used as a benchmark. The second algorithm is based on a local optimality characterization for Lipschitz least squares estimators. We call this the Local Optimization algorithm. Both algorithms are tested for different data distributions, sample sizes, and test functions. The analysis presented in this article focuses on two fundamental metrics: the running time and the maximum error (supremum norm).

We can conclude that the local optimization algorithm is able to output comparable results as the Beliakov algorithm in a fraction of the time (especially for large sample sizes). However, it is important to note that when dealing with non-uniform distributed data, especially where a small number of data points cover a larger portion of the entire interval, the algorithm needs to be carefully tuned to ensure comparable and reliable output.

Keywords: Least squares, LSE, Lipschitz, Scattered data approximation

Contents

1	Introduction	3
2	Model	4
3	Preliminaries	4
4	Solving the constrained Least Squares problem via quadratic programming	6
4.1	The brute force approach	6
4.2	Pruning the constraints (Beliakov)	6
4.3	Testing procedure	7
4.4	Performance of the Beliakov Algorithm	9

*Email: l.n.waninge@student.utwente.nl

5	Local Optimization algorithm	10
5.1	Breakdown of the algorithm	11
5.1.1	Picking a random interval	11
5.1.2	Optimizing the interval	12
5.1.3	Minimal change condition	13
5.2	Performance of the Local Optimization algorithm	14
6	Comparison of the Beliakov and Local Optimization algorithm	17
6.1	Running time	17
6.2	Approximation quality	17
7	Discussion	19
	References	20
A	Appendix	21
A.1	Data sets of the running time and error per category	21
A.2	Pseudocode	23

1 Introduction

Regression models are one of the main topics of research in statistics. In a regression model, it is assumed that data can be interpreted as a function with noise,

$$Y = f(X) + \epsilon, \tag{1}$$

where the error ϵ is normally distributed ($\epsilon \sim \mathcal{N}(0, \sigma)$) and X is a random variable with values in $[0, 1]$. The goal is to estimate the function f based on an independent and identically distributed (i.i.d.) sample $(X_1, Y_1), \dots, (X_n, Y_n)$, which can be done in multiple ways. The simplest method is to fit a linear function to the data. Although linear least squares is a well-studied regression estimator, in some applications the regression function might behave in a highly non-linear manner and therefore relaxations of the linear least squares are studied. This can be achieved by interpreting regression as an optimization problem. It is necessary to set up constraints for the class of non-parametric regression functions because we do not want to fit the noise. The latter is called overfitting, it is an error that occurs in a regression model that results from aligning a particular function too closely to the set of data points. When a model is overfitted the results are misleading and the model performs poorly as a predictive tool.

The approach that is considered in this article is to select a minimizer of the mean squared error among the class of Lipschitz functions. These are functions where the distance of the inputs of the function is closely related to the distance of the output of the function [5].

For a given sample size n , minimizing the least squares criterion over Lipschitz functions can be seen as a quadratic programming problem with $n \times (n - 1)$ constraints. Beliakov proposes an algorithm to prune redundant constraints to reduce the time complexity of this fit [2]. The number of non-redundant inequalities produced by this algorithm is significantly smaller than the original number of constraints, but its effectivity depends greatly on the dimension of the input data. A larger dimension equals fewer redundant inequalities. For small dimensions, the algorithm is very efficient in eliminating redundant constraints.

Recently, the Lipschitz least square estimator was proven to be minimax optimal with regard to a weighted supremum norm [4]. This behaviour is taken here as an optimality constraint and gives rise to an algorithm to approximate the Lipschitz Least Square Estimator. This local optimization property is analyzed and the Local Optimization Algorithm is implemented for the one-dimensional case. Both the Beliakov algorithm and the Local Optimization algorithm will be tested and their performance will be evaluated.

How do Beliakov's algorithm and the Local Optimization algorithm compare in terms of time complexity and their approximation quality?

2 Model

In this article, the basic regression model is used, which is described in Equation 1. We consider the Least Squares Estimator (LSE) over the Lipschitz-1 functions, which is the function \hat{f} defined as

$$\hat{f} \in \arg \min_{f \in \text{Lip}(1)} \frac{1}{n} \sum_{i=1}^n (Y_i - f(X_i))^2, \quad (2)$$

where $\text{Lip}(1)$ is the set of Lipschitz-1 functions. Lipschitz functions are functions where the distance of the output of the two function values is related to the distance of the input of the function [5]. The set of Lipschitz-1 functions considered in this work is

$$\text{Lip}(1) := \left\{ f: \mathbb{R} \rightarrow \mathbb{R} : |f(X_i) - f(X_j)| \leq |X_i - X_j| \right\}. \quad (3)$$

In order to measure the quality or correctness of the LSE, we introduce the L_∞ norm or also called the supremum norm;

$$\|f\|_\infty = \sup_{x \in \mathbb{R}} |f(x)|. \quad (4)$$

First, the infinite-dimensional optimization problem will be reduced to a finite-dimensional optimization problem, then, the quadratic programming (QP) approach will be explained, implemented, and tested. Finally, a local optimality characterization of the Lipschitz LSE will be derived, which will help derive an algorithm to evaluate the LSE.

3 Preliminaries

In this section, it is shown how the infinite-dimensional optimization problem in Equation 2 can be reduced to a finite-dimensional optimization problem. To do so we leverage the Lipschitz property of our hypothesis space. Everything will be proven for the one-dimensional case.

The first lemma shows that the optimization can be conducted within a n -dimensional vector space.

Lemma 3.1. *Let \mathcal{F}_l^n be the set of $(n - 1)$ -piecewise linear and Lipschitz-1 functions on $[0, 1]$. For any $X_1 < \dots < X_n \in [0, 1]$ and any $Y_1, \dots, Y_n \in \mathbb{R}$*

$$\mathcal{F}_l^n \cap \arg \min_{g \in \text{Lip}(1)} \left[\frac{1}{n} \sum_{i=1}^n (g(X_i) - Y_i)^2 \right] \neq \emptyset.$$

Proof. Let $\epsilon_i(X_i) = g(X_i) - Y_i$ be the residuals at point X_i . Then

$$\arg \min_{g \in \text{Lip}(1)} \left[\frac{1}{n} \sum_{i=1}^n (g(X_i) - Y_i)^2 \right] = \arg \min_{g \in \text{Lip}(1)} \left[\frac{1}{n} \sum_{i=1}^n \epsilon_i(X_i)^2 \right]$$

is the set of Lipschitz-1 functions that minimizes the residuals of all data points, regardless of what happens in between these points. Therefore in between these points, it could be linear, which results in a $(n - 1)$ -piecewise linear Lipschitz-1 function. Hence the intersection is non-empty and as a consequence, the infinite-dimensional optimization problem can be reduced to an optimization over a n -dimensional vector space. \square

Having established the proof that the Lipschitz function can be represented as a piecewise linear function, our objective now is to define an optimal piecewise linear function for the Lipschitz interpolation. Therefore the data set needs to be compatible.

Definition 3.2. A set $\{(X_i, Y_i)\}_{i=1}^n$ is called compatible if there exists a Lipschitz-1 interpolation. That is if

$$\min_{f \in \text{Lip}(1)} \frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2 = 0.$$

In between two data points a cone shape is formed by the constraints of the Lipschitz-1 function, where the upper bound is defined as $f_{max}(x) := \min_i(Y_i + |x - X_i|)$ and the lower bound as $f_{min}(x) := \max_i(Y_i - |x - X_i|)$. These are the bounds of the Lipschitz-1 interpolation. It has already been shown that $\hat{f}(x) = \frac{1}{2}(f_{min}(x) + f_{max}(x))$ gives the best approximation in the worst case [6]. Now we proceed to prove that this function indeed is an optimal approximation for the Lipschitz-1 interpolation.

Lemma 3.3. Let f_{max} and f_{min} be defined as above. If the set $D := \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ is compatible, then

$$\hat{f} = \frac{1}{2}(f_{max} + f_{min}) \in \arg \min_{f \in \text{Lip}(1)} \frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2.$$

Proof. Since the set is compatible, by Definition 3.2 it must be shown that $\frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2 = 0$ holds for $\hat{f}(x) = \frac{1}{2}(f_{max}(x) + f_{min}(x))$. Substituting this equation into the compatibility equation, yields

$$\frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2 = \frac{1}{n} \sum_{i=1}^n \left[\frac{1}{2} (f_{max}(X_i) + f_{min}(X_i)) - Y_i \right]^2 \quad (5)$$

$$= \frac{1}{n} \sum_{i=1}^n \left[\frac{1}{2} \left(\min_j (Y_j + |X_j - X_i|) + \max_j (Y_j - |X_j - X_i|) \right) - Y_i \right]^2. \quad (6)$$

Now we prove by contradiction that for all $i \in \{1, \dots, n\}$, $\min_j (Y_j + |X_j - X_i|) = Y_i + |X_i - X_i|$. First, assume the minimum is at some $\tilde{j} \neq i$. Then

$$Y_{\tilde{j}} + |X_{\tilde{j}} - X_i| < Y_i + |X_i - X_i|.$$

The distance between X_i and X_i is zero and rewriting the equation results in

$$Y_i - Y_{\tilde{j}} > |X_{\tilde{j}} - X_i|.$$

This contradicts the fact that the function f is in $\text{Lip}(1)$, see Equation 3. Therefore the minimum must be obtained at i . The same argument holds for the maximum. Substituting these points of the minimum and maximum into Equation 5 yields

$$\frac{1}{n} \sum_{i=1}^n (f(X_i) - Y_i)^2 = \frac{1}{n} \sum_{i=1}^n \left(\frac{1}{2}(Y_i + Y_i) - Y_i \right)^2 = 0.$$

Hence it is proven that \hat{f} is a minimizer of the squared error. \square

Lemma 3.3 establishes that if a set is compatible, then \hat{f} is the minimizer of the least squared error. Not every data set is compatible, therefore it is necessary to employ alternative methodologies to find this minimizer. Such an alternative is described in the next section.

4 Solving the constrained Least Squares problem via quadratic programming

In this section, the quadratic programming approach for solving the constrained least squares problem is discussed.

4.1 The brute force approach

The brute force approach simply defines the Lipschitz Least Squares problem as a quadratic program, further referred to as the QP. First, the set's compatibility with the Lipschitz condition is ensured by smoothening the data. The residuals are the differences between the approximation and the original data: $r_i = \hat{f}(X_i) - Y_i$. Here r_i is the residual and $\hat{f}(X_i)$ is the approximation. Substituting the residual into the compatibility definition in Definition 3.2 gives the objective function for the QP. The constraints are determined by the Lipschitz condition in Equation 3. This results in the following QP

$$\begin{aligned} \min \quad & \sum_{i=1}^n |r_i|^2 \\ \text{s.t.} \quad & r_i - r_j \leq Y_j - Y_i + |X_i - X_j| \\ & \text{for all } i, j \in 1, \dots, n. \end{aligned} \tag{7}$$

The QP has a total of $n(n-1)$ constraints.

Once the data is smoothened, the optimal approximation is computed as described in Lemma 3.3. There are various ways to reduce the time complexity of this QP. Beliakov proposes such an algorithm.

4.2 Pruning the constraints (Beliakov)

Beliakov proposes an algorithm to reduce the number of constraints of the quadratic program in Equation 7 [2]. It is based on describing the Lipschitz compatibility in Definition 3.2 with the simplicial distance instead of a norm. The use of a different distance function only affects the Lipschitz constant of f . The distance function that is of interest is one based on the Minkowski gauge.

Let P be a simplex defined as the intersection of $d+1$ halfspaces (where d is the dimension of x)

$$P = \bigcap_{i=1}^{d+1} \{x : x \cdot h_i \leq 1\},$$

for the directional vectors

$$\begin{aligned} h_1 &= (-v_1, 0, \dots, 0), \\ h_2 &= (0, -v_2, \dots, 0), \\ &\vdots \\ h_d &= (0, \dots, 0, -v_d), \\ h_{d+1} &= (v_{d+1}, \dots, v_{d+1}) \end{aligned}$$

with $v_i > 0$. Let us introduce the slack variable x_{d+1} , then the simplicial distance is defined in the following way.

Definition 4.1. The simplicial distance between the points $x, y \in \mathbb{R}^d$ is

$$d_P(x, y) = \max_{i=1, \dots, d+1} v_i(y_i - x_i),$$

$$\text{where } x_{d+1} = 1 - \sum_{i=1}^d x_i.$$

With the use of the slack variable x_{d+1} , the space \mathbb{R}^d becomes a hyperplane $HP \subset \mathbb{R}^{d+1}$. Once the distance between the points is described in a simplicial distance, the hyperplane HP can be divided into sets based on their distance. For $i \in \{1, \dots, d+1\}$ and $k = 1, \dots, n$ we define the following sets

$$S_i^k = \{x \in HP : v_i(x_i - x_i^k) \geq v_j(x_j - x_j^k), \text{ for all } j \neq i\}$$

for all data x_k . By definition, it is given that if $x_k \in S_i^j$, and $x_p \in S_i^k$, then $x_p \in S_i^j$ and $d_P(x_p, x_j) = d_P(x_p, x_k) + d_P(x_k, x_j)$. As a consequence, the inequality from (p, j) is redundant. Adopting this approach, the redundant constraints can be eliminated from the QP in Equation 7. This QP with a significantly smaller number of constraints can be solved and the approximation can be computed as described in Section 4.1.

A more thorough explanation of the algorithm and the pseudocode can be found in Beliakov's article [2]. Next, the algorithm will be implemented in Python and its performance will be tested. But first, the formal testing procedure will be discussed.

4.3 Testing procedure

This section outlines the formal testing procedure for the Beliakov algorithm.

The algorithm will be evaluated by conducting tests under various scenarios, including different data distributions, diverse test functions and varying sample sizes. The following data distributions are considered:

- X_0 is uniformly distributed,
- X_1 has probability density function $p_1(x) = 2x$,
- X_2 has probability density function $p_3(x) = -6(x - 0.5)^2 + \frac{3}{2}$,
- X_3 has probability density function $p_2(x) = 12(x - 0.5)^2$.

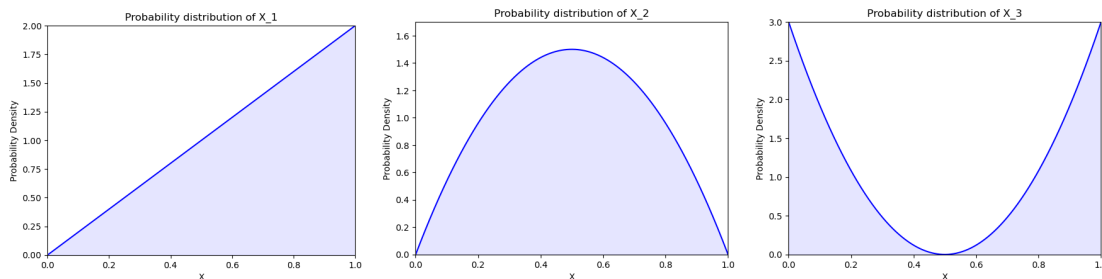


FIGURE 1: Plots of the Probability Density Functions of the different distributions

In Figure 1, the distributions of X_1 , X_2 , and X_3 are displayed. The method that is used to generate samples from these distributions is Inverse Transform Sampling. This method

allows one to generate random samples from any probability distribution function with an invertible cumulative distribution function (CDF). Let U be uniform on $[0, 1]$ and let $p(x)$ be the probability distribution function of random variable X with $\int_{[0,1]} p(t)dt = 1$. Let $F(X)$ be the CDF of $p(x)$, i.e. $F(x) = \int_0^x p(z)dz$. Then $U = F(X)$ and $F^{-1}(U) = X$, where F^{-1} is the inverse of the CDF [3]. Hence applying the inverse CDF to an uniform sample results in a sample that follows the distribution of the random variable X .

Additionally, the algorithm will be tested on four different Lipschitz-1 functions, where the first test function is the zero function, which means that the response variables are pure noise. The second test function is continuous, the third test function is a piece-wise linear function and the fourth is a combination of the second and third. The functions are

- $f_0(x) = 0$,
- $f_1(x) = 0.5x^2 + 0.25$,
- $f_2(x) = |x - 0.5|$,
-

$$f_3(x) = \begin{cases} 0.5 + x & \text{if } 0 \leq x < 0.2 \\ 0.9 - x & \text{if } 0.2 \leq x \leq 0.3 \\ 1.25(x - 0.7)^2 + 0.4 & \text{if } 0.3 < x \leq 1, \end{cases}$$

and are displayed in Figure 2. We sample the noise from the normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.1$. A combination of an X -distribution and a test function is referred to as a *category*. Each category is tested for increasing sample sizes, which are 50, 100, 250, 500, 1 000, 5 000, and 10 000. Every category is run 50 times for each sample size.

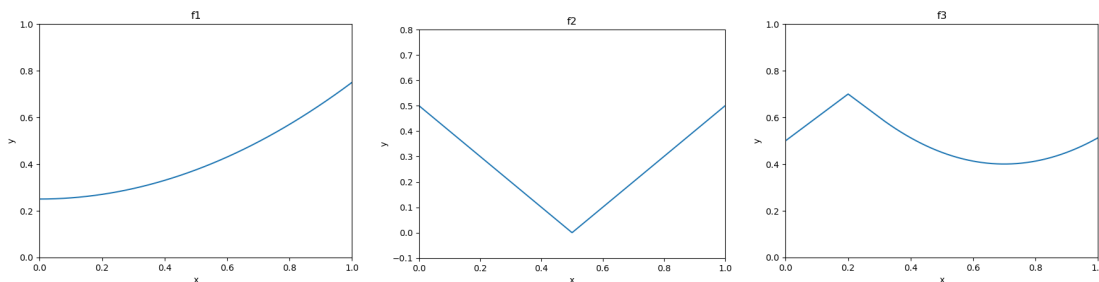


FIGURE 2: Plots of the test functions

The performance of each run is determined by the total running time of the algorithm and the error. The error that is considered is the supremum norm in Equation 4. This is the maximum difference between the approximation and the original test function. Per category and sample size the mean of these errors will be calculated. Throughout this article, this mean will be referred to as the mean max(imum) error. Given the independence and identical distribution of the mean maximum error and running time originating from two distinct runs within a category, the application of the central limit theorem allows us to assume that they are normally distributed. Leveraging this property allows us to

calculate 95%-confidence intervals for said errors and times. This is done in the following way

$$P\left(\bar{x} - z\frac{s}{\sqrt{n}} \leq X \leq \bar{x} + z\frac{s}{\sqrt{n}}\right) = 0.95, \quad (8)$$

where \bar{x} is the sample mean, s is the standard deviation, n is the sample size and $z = 1.96$.

4.4 Performance of the Beliakov Algorithm

First, the error is considered. As anticipated, a larger number of data points corresponds to a more accurate approximation, thereby resulting in a smaller maximum error. Table 1 presents the mean maximum error across all categories per number of data points along with the corresponding standard deviation (sd) and the 95%-confidence interval (95%-CI). The confidence interval is calculated using Equation 8. From the table, it is clear that when the number of data points increases, the mean maximum error and the standard deviation both decrease. In Appendix A.1 there is a more extensive table, Table 10, which shows the mean maximum error per category.

TABLE 1: Summary of the mean maximum error of the Beliakov algorithm per sample size

n	Mean maximum error	sd	95%-CI
50	0.0859	0.0301	[0.0838, 0.0880]
100	0.0754	0.0272	[0.0735, 0.0773]
250	0.0607	0.0242	[0.0590, 0.0624]
500	0.0533	0.0209	[0.0519, 0.0548]
1 000	0.0451	0.0190	[0.0438, 0.0464]
5 000	0.0315	0.0157	[0.0305, 0.0326]

Another important aspect of the algorithm is the total running time of the algorithm. The running time of the algorithm increases as the number of data points increases. This is supported by Table 2, which displays the mean running time for each sample size. Next to that, it can be seen that the standard deviation (sd) increases as well and that the majority of the running time is accounted for by the QP-solver. On average 97.6% of the total running time is occupied by the QP-solver. The QP-solver that was used in the implementation is *cvxopt* [1]. In Appendix A.1 in Table 12 the running time per category is shown. Due to the long running time of the algorithm for a larger number of data points, on average 82.8 seconds for 5 000 points, testing the algorithm on larger sample sizes was not conducted.

TABLE 2: Summary of the running time of the Beliakov algorithm per sample size

n	Mean running time [s]	sd [s]	95%-CI [s]	% QP- solver
50	0.0143	0.00845	[0.0137, 0.0149]	95.2
100	0.0297	0.00883	[0.0291, 0.0303]	98.9
250	0.0616	0.0141	[0.0606, 0.0626]	98.3
500	0.195	0.0302	[0.193, 0.197]	96.8
1 000	1.05	0.133	[1.04, 1.06]	97.2
5 000	82.8	16.4	[81.6, 83.9]	99.2

This algorithm is considered to be the true LSE and is used as a benchmark for the Local Optimization algorithm. The latter will be introduced in the next section.

5 Local Optimization algorithm

The following local behaviour for Lipschitz least squares estimators is observed.

Lemma 5.1. *Let $\hat{y} = (\hat{y}_1, \dots, \hat{y}_n)$ with $\hat{y}_i = \hat{f}(X_i)$ be a Lipschitz least square estimator. For any partition of $[n]$ into P non-empty sets I_1, \dots, I_P such that*

$$\forall (i, j) \in I_l \times I_m, l < m \Rightarrow X_i < X_j.$$

Let $(i_m, i_M) = (\min I_i, \max I_i)$, then for any $p \in [P]$, either one or both of the following statements holds

$$\begin{aligned} 1) \quad & \sum_{i \in I_p} \hat{y}_i = \sum_{i \in I_p} Y_i, \\ 2) \quad & \frac{\hat{y}_{p_m} - \hat{y}_{(p-1)_M}}{(x_{p_m} - x_{(p-1)_M})} = -\sigma \text{ or } \frac{\hat{y}_{(p+1)_m} - \hat{y}_{p_M}}{(x_{(p+1)_m} - x_{p_M})} = \sigma, \end{aligned}$$

where

$$\sigma = \text{sign}\left(\sum_{i \in I_p} \hat{y}_i - \sum_{i \in I_p} Y_i\right).$$

If $p \in \{1, n\}$ then 2) consists of only the well-defined equality and admits no 'or' expression.

Proof. Let I_1, \dots, I_P be such a partition and suppose that there is a $p \in [P]$ such that both 1) and 2) do not hold. Suppose w.l.o.g that $\sigma = 1$. Since \hat{y} is Lipschitz, we have

$$\begin{aligned} \hat{y}_{p_m} - \hat{y}_{(p-1)_M} &> -\sigma(x_{p_m} - x_{(p-1)_M}), \\ \hat{y}_{(p+1)_m} - \hat{y}_{p_M} &< \sigma(x_{(p+1)_m} - x_{p_M}). \end{aligned}$$

By continuity, there exists $\eta_0 > 0$ such that for any $\eta \in [0, \eta_0]$

$$\begin{aligned} \hat{y}_{p_m} - \hat{y}_{(p-1)_M} - \eta &\geq -\sigma(x_{p_m} - x_{(p-1)_M}), \\ \hat{y}_{(p+1)_m} - \hat{y}_{p_M} + \eta &\leq \sigma(x_{(p+1)_m} - x_{p_M}). \end{aligned}$$

Then $\tilde{y} = (\hat{y}_1, \dots, \hat{y}_{(p-1)_M}, \hat{y}_{p_m} - \eta, \hat{y}_{p_m+1} - \eta, \dots, \hat{y}_{p_M} - \eta, \hat{y}_{(p+1)_m}, \dots, y_n)$ is a Lipschitz vector. Expanding the LSE problem with \tilde{y} yield

$$\sum_{i=1}^K (Y_i - \tilde{y}_i)^2 = \sum_{i=1}^{(p-1)_M} (Y_i - \hat{y}_i)^2 + \sum_{(p+1)_m}^n (Y_i - \hat{y}_i)^2 + \sum_{p_m}^{p_M} (Y_i - \hat{y}_i + \eta)^2.$$

Expanding that last part gives $\sum_{p_m}^{p_M} (Y_i - \hat{y}_i + \eta)^2 = \sum_{p_m}^{p_M} (Y_i^2 + \hat{y}_i^2 + \eta^2 - 2Y_i\hat{y}_i + 2\eta Y_i - 2\eta\hat{y}_i)$. Substituting this into the previous equation and rewriting this yields

$$\sum_{i=1}^K (Y_i - \tilde{y}_i)^2 = \sum_{i=1}^K (Y_i - \hat{y}_i)^2 + \sum_{i \in I_p} (\eta^2 + 2\eta Y_i - 2\eta\hat{y}_i)$$

$$= \sum_{i=1}^K (Y_i - \hat{y}_i)^2 + \underbrace{|I_p|\eta^2 + 2\eta \sum_{i \in I_p} (Y_i - \tilde{y}_i)}_{P(\eta)}.$$

So the LSE with \tilde{y} can be written as the LSE with \hat{y} and a second-degree polynomial $P(\eta)$ where $|I_p|$ denotes the cardinal of I_p . One root of the polynomial is 0. Since $\omega = \sum_{i \in I_p} (Y_i - \tilde{y}_i) < 0$ by assumption ($\sigma = 1$), the second root of P is $\eta_1 = -2\omega/|I_p| > 0$. Finally, since P has a positive dominating coefficient $|I_p|$, it is deduced that for $\eta \in (0, \eta_1)$, $P(\eta) < 0$ and that

$$\sum_{i=1}^K (Y_i - \tilde{y}_i)^2 < \sum_{i=1}^K (Y_i - \hat{y}_i)^2.$$

This is a contradiction with the fact that \hat{y} is an LSE. So either one or both of 1) and 2) must hold. If 1) is true, then $\sum_{i=1}^K (Y_i - \tilde{y}_i)^2 = \sum_{i=1}^K (Y_i - \hat{y}_i)^2 + \eta^2(p_M - p_m)$. And if 2) is true, then $\tilde{y} = \hat{y}$. □

This behaviour can be taken as a local optimization condition and gives rise to an algorithm.

5.1 Breakdown of the algorithm

The goal of the algorithm is to construct a Lipschitz vector $\hat{y} = [\hat{y}_1, \dots, \hat{y}_n]$ which satisfies the conclusion of Lemma 5.1 as closely as possible. The input of the algorithm is data $[X_1, \dots, X_n]$ and their corresponding response variables of the Lipschitz regression estimate $[Y_1, \dots, Y_n]$.

We initialize \hat{y} as a constant vector, where each entry is the average of Y , so $\hat{y} = \left[\frac{1}{n} \sum_{i=1}^n Y_i \right]_{i=1}^n$. The base of the algorithm is a loop. At each iteration, we uniformly pick an element J from the set of all boundaries S , where

$$S = \{(a, a + 1, \dots, b - 1, b) \mid a, b \in [0, \dots, n], a \neq b, a < b\}$$

Then the first element of J is the lower bound of the interval and the second element is the upper bound of the interval. Let I be the set of all indices in the interval based on J . If this interval adheres to the first condition of Lemma 5.1, so $\bar{Y}_I = \bar{y}_I$ where $\bar{Y}_I = \frac{1}{m} \sum_{i \in I} Y_i$ and $\bar{y}_I = \frac{1}{m} \sum_{i \in I} y_i$ and m is the cardinal of I , then this interval is optimal. If it does not, \hat{y} can be updated by shifting its entries with indexes in I in the direction of the local average Y_I while preserving the Lipschitz property of \hat{y} .

5.1.1 Picking a random interval

To ensure that every point is optimized evenly, during every iteration, the entire range of points is optimized. The indices in the set I split the interval $[0, 1]$ into one, two, or three intervals as displayed in Figure 3. All these intervals will be optimized consecutively in the same iteration. An interval that contains the zeroth data point is referred to as a *left-interval*, an interval ending at point data point n is referred to as a *right-interval* and an interval including neither zero nor n , is referred to as a *middle-interval*.

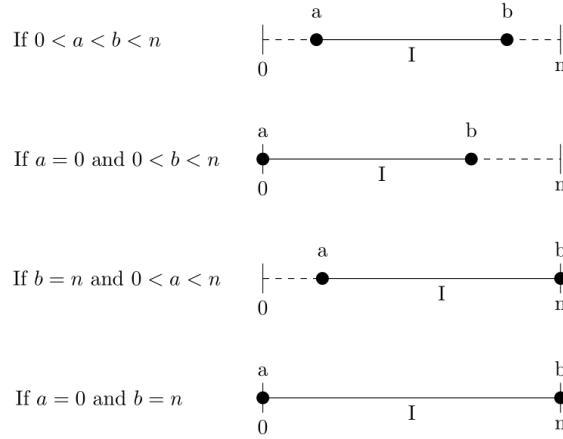


FIGURE 3: Options for the four types of intervals

5.1.2 Optimizing the interval

If the mean of the approximation over the interval \bar{y}_I is not equal to the mean of the data over the interval \bar{Y}_I , then the approximation can be improved.

The optimal shift of the interval δ_{opt} is the difference between these two means, so $\delta_{opt} = \bar{Y}_I - \bar{y}_I$. Let indices ℓb and ub be the lower and upper bound of an interval I respectively, so $\ell b = \min_{i \in I} X_i$ and $ub = \max_{i \in I} X_i$.

In case of left-intervals, the optimization starts from the upper boundary, whereas for right-intervals, it initiates from the lower boundary. This is because the movement of the first and last data point is not restricted by the Lipschitz condition. Middle intervals are on both sides constrained by the Lipschitz condition. Hence it is crucial to carefully select the starting boundary of the optimization. It starts from the side where the boundary point possesses the least freedom to move. Let $u_{\ell b}$ and u_{ub} be the maximum movement due to the Lipschitz condition of the lower and upper bound respectively, so

$$u_{\ell b} = \hat{y}_{\ell b-1} + \sigma |X_{\ell b-1} - X_{\ell b}| \quad u_{ub} = \hat{y}_{ub+1} + \sigma |X_{ub+1} - X_{ub}|.$$

The most restricted point is $\min\{|\hat{y}_{\ell b} - u_{\ell b}|, |\hat{y}_{ub} - u_{ub}|\}$ and the optimization start from that boundary. An example of this decision-making can be seen in Figure 4.

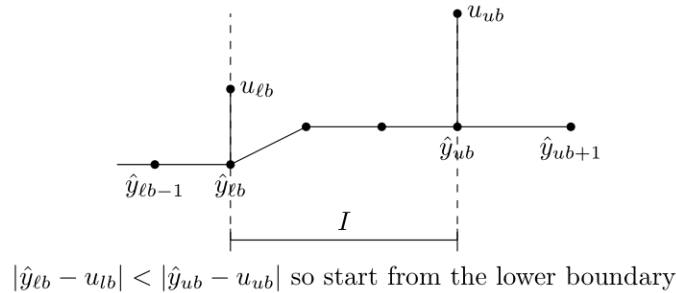


FIGURE 4: Example of the decision making on where to start the optimization of a middle interval.

Figure 5 illustrates an interval scenario where the optimization starts from the upper boundary. This serves as an example to further illustrate the optimization. For the optimization, there are two suitable options, v_1 and v_2 , where v_1 is the optimal shift and v_2

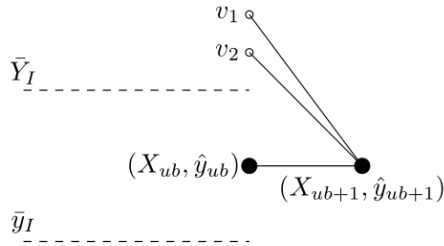


FIGURE 5: Example of the upper boundary of the left interval with the two optimization options

is the maximum shift in the direction of the actual mean without exceeding the Lipschitz condition:

$$v_1 = \hat{y}_{ub} + \delta_{opt} \quad v_2 = \hat{y}_{ub} + \sigma |X_{ub} - X_{ub+1}|.$$

The point with minimal distance to \hat{y}_{ub} is chosen as the new value for \hat{y}_{ub} , so

$$\hat{y}_{ub} = \arg \min_{i=1,2} |\hat{y}_{ub} - v_i|.$$

All data points in the interval are shifted equally to the data mean.

5.1.3 Minimal change condition

The base of the algorithm is a loop, which optimizes the entire interval during every iteration. We break the loop once the approximation is close enough to satisfy the conditions in Lemma 5.1. Let e_{T-i} be the magnitude of the shift operated at iteration $T-i$, where T is the total number of iterations. Let k be a positive integer, and let $\epsilon > 0$ be a threshold, then the algorithm stops once the minimal change condition is exceeded:

$$\frac{1}{k} \sum_{i=1}^k e_{T-i} \leq \epsilon \tag{9}$$

During every iteration, the entire interval is subdivided into one, two or three sub-intervals, with the latter being the most probable outcome. Furthermore, if there are no modifications observed during the last 20 iterations, we consider the approximation to be optimal. Hence, approximately $3 \times 20 = 60$ optimization steps must go by with only slight updates. On the interval $[0, 1]$, there are n data points. Under a uniform distribution, the average distance between these points is $\frac{1}{n}$. By the Lipschitz condition in Equation 3 the maximum average shift per optimization is $\frac{1}{n}$. If we permit at most one update in the last 20 iterations (so 60 optimization steps), then the algorithm should stop if the mean over the last 60 optimizations is smaller than $\frac{1}{60n}$. So in this case the minimal change condition in Equation 9 has values $k = 60$ and $\epsilon = \frac{1}{60n}$. This boundary condition is later referred to as the original boundary or the original Local Optimization algorithm.

A relaxation of this minimal change condition could greatly improve the running time of the algorithm. However, it can also have an impact on the approximation quality. Conversely, if the minimal change condition is stricter, it can lead to improved approximation accuracy but it may increase its running time. In addition to the original boundary, the algorithm is tested using four stricter minimal change conditions. The values of all minimal change conditions are displayed in Table 3.

The pseudocode of the Local Optimization algorithm can be found in Appendix A.2. In the next section, the performance of the Local Optimization is considered.

TABLE 3: Values of k and ε for different minimal change conditions

Name of minimal change condition	k	ε
Original	60	$\frac{1}{60n}$
Boundary 2	2×60	$\frac{1}{2 \times 60n}$
Boundary 4	4×60	$\frac{1}{4 \times 60n}$
Boundary 6	6×60	$\frac{1}{6 \times 60n}$
Boundary 50	6×60	$\frac{1}{50 \times 60n}$

5.2 Performance of the Local Optimization algorithm

The algorithm is tested using the same test procedure as the Beliakov algorithm, as outlined in Section 4.3.

TABLE 4: Summary of maximum error data for the Local Optimization algorithm per sample size

n	Mean maximum error	sd	95%-CI
50	0.0841	0.0289	[0.0821, 0.0861]
100	0.0744	0.0254	[0.0727, 0.0762]
250	0.0608	0.0253	[0.0591, 0.0626]
500	0.0537	0.0236	[0.0520, 0.0553]
1 000	0.0484	0.0259	[0.0466, 0.0501]
5 000	0.0402	0.0293	[0.0382, 0.0422]
10 000	0.0377	0.0313	[0.0356, 0.0399]

In Table 4, the mean maximum error, the standard deviation, and the 95%-confidence interval per sample size are displayed. It is evident that the (mean) maximum error decreases as the number of data points increases. On the other hand, the standard deviation remains relatively constant and does not exhibit a clear trend of increase or decrease with respect to the number of data points. Upon closer examination of the data, by looking at the extended version of this table, Table 11 in Appendix A.1, it becomes apparent that one category exhibits poor performance compared to the others. For the combination of X_3 -distributed data and the test function f_2 , the mean maximum error increases as the number of data points increases. The mean maximum error for 50 and 10 000 points is 0.0954 and 0.1329 respectively. In Figure 6 a plot of the approximation of the function for one of the test runs is shown for sample sizes 100, 1 000, and 10 000.

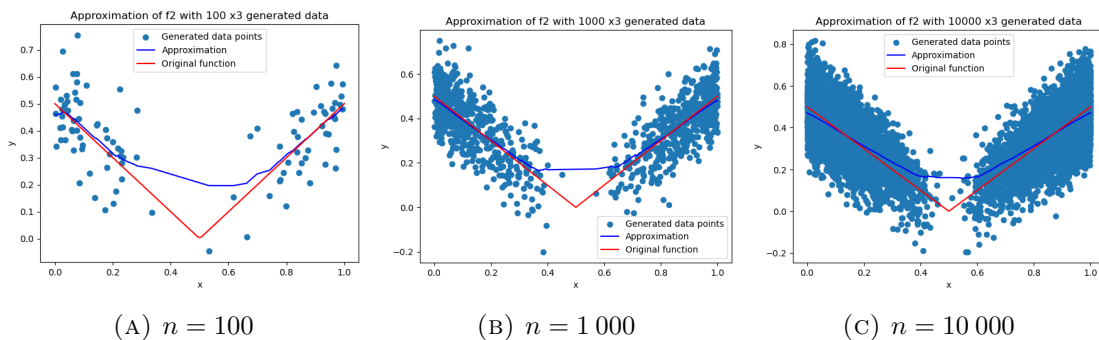


FIGURE 6: Approximation of f_2 with X_3 -distributed data for different sample sizes

The diminished approximation quality in this category can be attributed to two factors. First, the minimal change condition is based on the assumption that the data is uniformly distributed where the data points have a mean distance of $\frac{1}{n}$. When the data is X_3 -distributed, that distance is significantly smaller for the majority of the points. Therefore the maximum shift per iteration is smaller and the loop is stopped too early. Secondly, the data points laying between 0.4 and 0.6 are chosen too little as a boundary point. Consequently, this part is not optimized often enough and the quality of the approximation is lacking. This one particularly bad-performing category significantly increases the overall mean of the maximum error. Removing this category from the data, the mean maximum error is considerably smaller, especially for larger sample sizes. The plot of the mean maximum errors is displayed in Figure 7.

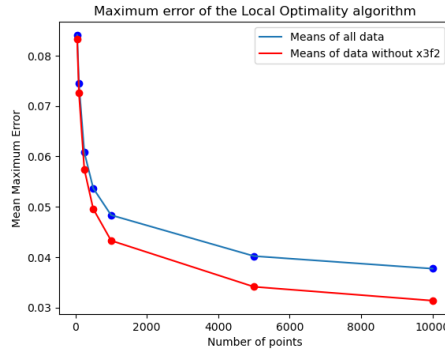


FIGURE 7: Plot of the mean maximum error of the Local Optimization algorithm with and without X_3 -distributed data for test function f_2

While removing the poor-performing category from the data shows that all other categories perform decently, improving the performance of this specific category must be investigated. To address the issue, stricter minimal change conditions are implemented. By doing so, the number of optimization steps of the algorithm increase, leading to a more accurate approximation. The stricter boundaries that are tested are presented in Table 3 and they will be applied to data depicted in Figure 6. In Figure 8 the approximations with these stricter boundaries are displayed as well as the approximation of the Beliakov algorithm.

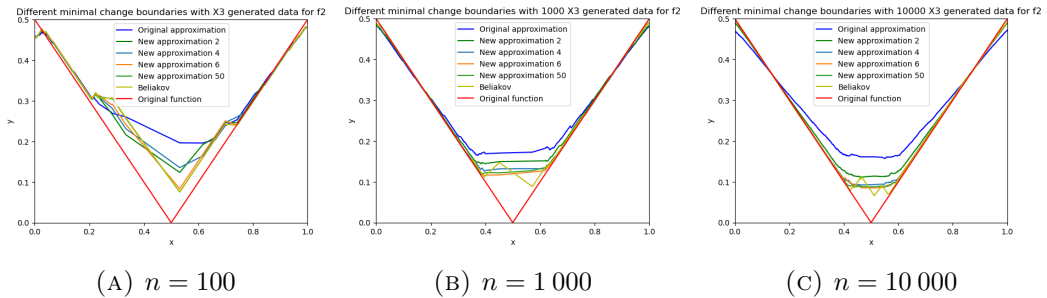


FIGURE 8: Plots of approximations with different minimal change boundaries for the Local Optimality algorithm for different sample sizes

From the plots, it is evident that the approximation improves when the minimal change condition is stricter. In Table 5 and 6, the improved maximum error and the increased running times are displayed respectively. For all sample sizes, the approximation from the Local Optimality algorithm with boundary 6 and boundary 50 closely resemble the

'true' approximation from the Beliakov algorithm. Although it does not perfectly align, the maximum error of the Local Optimization algorithm with boundary 6 and boundary 50 is smaller for each sample size than the maximum error of the Beliakov approximation. While the running time of the algorithm is significantly increased by the stricter boundary, it is still considerably faster than the Beliakov algorithm for large sample sizes. Upon closer examination of Figure 8, it becomes apparent that there is no significant difference between the approximation with boundary 6 and boundary 50. Therefore, taking into account both running time and improvement of the algorithm, we consider boundary 6 to be the optimal minimal change condition for this poor-performing category.

TABLE 5: Maximum error for the Local Optimization algorithm with different minimal change conditions and the true approximation

Maximum error	$n = 100$	$n = 1\ 000$	$n = 10\ 000$
Original	0.165	0.121	0.150
Boundary 2	0.0924	0.100	0.102
Boundary 4	0.104	0.0827	0.815
Boundary 6	0.0758	0.0677	0.0735
Boundary 50	0.0904	0.0729	0.0759
Beliakov	0.0920	0.0981	0.0775

TABLE 6: Running time in seconds for the Local Optimization algorithm with different minimal change conditions and the true approximation

Running time	$n = 100$	$n = 1\ 000$	$n = 10\ 000$
Original	0.0312	0.298	11.5
Boundary 2	0.185	0.708	21.9
Boundary 4	0.277	1.13	19.8
Boundary 6	0.850	1.63	23.0
Boundary 50	1.72	2.02	26.6
Beliakov	0.0667	1.93	1039

Next, we consider the running time of the algorithm with the original boundary. The algorithm's running time increases as the number of data points increases. The same applies to the standard deviation (sd). The foregoing, as well as the 95%-confidence interval, are displayed in Table 7.

TABLE 7: Summary of the running time of the Local Optimization algorithm

n	Mean running time [s]	sd [s]	95%-CI [s]
50	0.0381	0.0139	[0.0371, 0.0390]
100	0.0686	0.0263	[0.0668, 0.704]
250	0.135	0.0389	[0.132, 0.137]
500	0.253	0.0825	[0.248, 0.259]
1 000	0.502	0.113	[0.494, 0.510]
5 000	4.19	0.562	[4.15, 4.23]
10 000	13.9	1.81	[13.8, 14.0]

With the completion of testing of the Local Optimization algorithm, a comparative analysis can be done for both algorithms. This is presented in the next section.

6 Comparison of the Beliakov and Local Optimization algorithm

In this section, the Beliakov algorithm and the Local Optimization algorithm will be compared.

6.1 Running time

First, the running times of both algorithms are considered. Figure 9 displays the respective running times of both algorithms. Figure 9b demonstrates a slight advantage in terms of speed for the Beliakov Algorithm over the Local Optimization algorithm for small sample sizes ($n \leq 500$). For larger sample sizes, the magnitude of the running time of the Beliakov algorithm increases enormously, whereas the Local Optimization algorithm demonstrates superior speed. Since the running time of the Beliakov algorithm is greatly dependent on the running time of the QP-solver (on average the QP-solver accounts for 97.6% of the running time), the difference between the two algorithms can be significantly influenced by the presence of a faster QP-solver. To ensure a notable impact of the difference between the running times of the two algorithms, the QP-solvers must be at least 20 times faster than the current implementation, as evidenced by the Beliakov algorithm being 20 times slower than the Local Optimization algorithm for 5 000 data points.

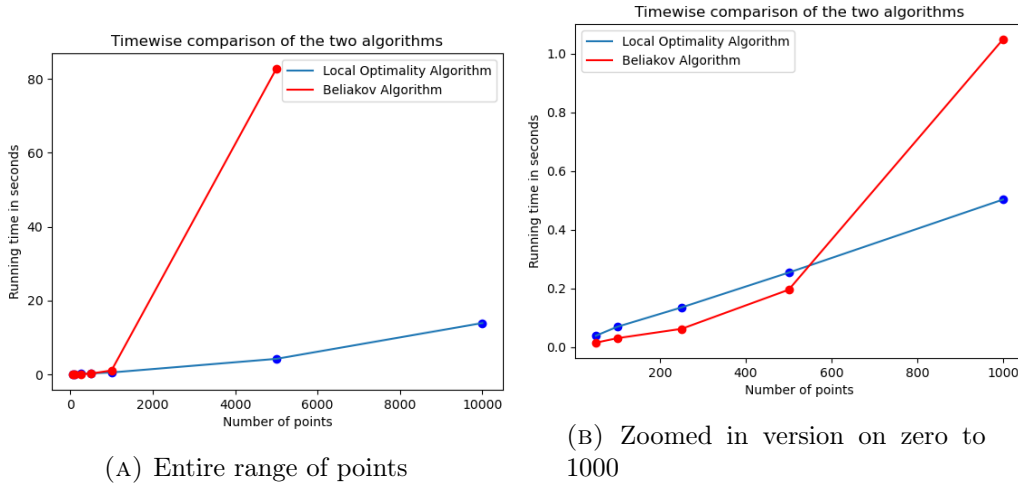
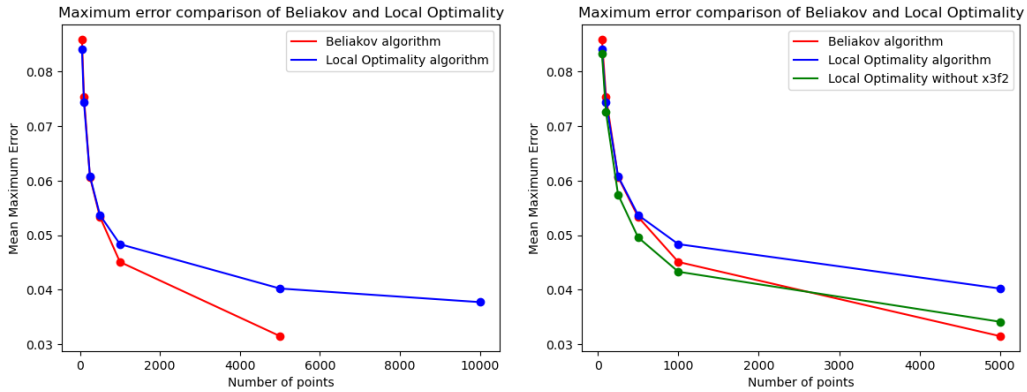


FIGURE 9: Plots of the running times of both algorithms

6.2 Approximation quality

In evaluating the approximation quality of both algorithms, an important question arises: can the Optimality algorithm replicate the true approximation of the Beliakov algorithm? Therefore the maximum errors for both algorithms are compared. The mean maximum errors of the Beliakov and Local Optimization algorithm are illustrated in Figure 10a. It is apparent that, when dealing with small sample sizes ($n \leq 500$), the differences between the mean maximum errors of the two algorithms are not significant. However, for sample sizes larger than 500, the Beliakov algorithm has a noticeably accelerated decline in maximum error compared to the Local Optimization algorithm. If the decreasing trend persists, it is anticipated that the maximum error will diminish further as the number of data points increases.



(A) Mean maximum errors as a function of the sample size of the two algorithms

(B) A zoomed-in version without the worst category for Local Optimization

FIGURE 10: Plots of the mean maximum error for both algorithms

In Section 5.2 it was determined that the Local Optimization algorithm performs inadequately when approximating the function f_2 when the data points are X_3 -distributed. The presence of this 'poor' category significantly influences the maximum error, as the mean maximum error of this category increases with an increasing number of data points. In Figure 10b, the mean maximum error of the Local optimization algorithm excluding the worst category, is plotted alongside the original plot of the mean maximum data in Figure 10a. The results indicate that the Local Optimization algorithm exhibits superior performance for 1 000 or fewer data points, while the Beliakov algorithm still demonstrates a smaller mean maximum error beyond this range.

Previously, the analysis solely focused on noise samples from a normal distribution with a mean $\mu = 0$ and a standard deviation $\sigma = 0.1$. However, we will now broaden our research to include larger noise with the same mean but a larger standard deviation $\sigma = 0.3$.

To evaluate the algorithms' performance under the influence of larger noise, we will generate sample data from a uniform distribution (X_0) and assess their ability to approximate function f_3 (see Figure 2). The experiments will be conducted using sample sizes of 100, 1 000, and 10 000, with each configuration run 20 times. For the Local Optimization algorithm, the performance will be evaluated for both the original minimal change condition as well as the stricter condition: boundary 6.

TABLE 8: Mean maximum error for $\mathcal{N}(0, 0.3^2)$ -distributed noise for the Beliakov and Local Optimization algorithm

Maximum error	$n = 100$	$n = 1\ 000$	$n = 10\ 000$
Original	0.138	0.0683	0.0341
Boundary 6	0.139	0.0687	0.0352
Beliakov	0.139	0.691	0.0353

Table 8 presents the mean maximum error per sample size for both algorithms. It is evident that the mean maximum error is nearly identical for the different algorithms, which is further supported by the corresponding plots in Figure 11.

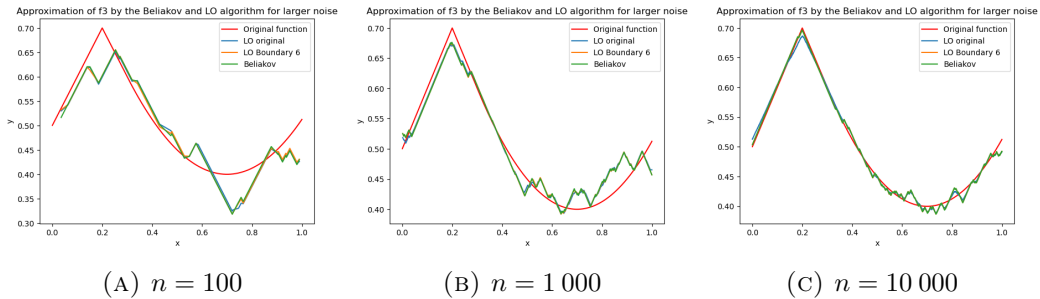


FIGURE 11: Plots of the approximation of f_3 by the Beliakov algorithm and the Local Optimization algorithm with the original boundary and boundary 6

This plot also demonstrates that the implementation of a stricter boundary condition (boundary 6) for the Local Optimization algorithm does not lead to an improvement in the maximum error when considering uniformly distributed data. The stricter boundary does increase the running time, which is shown in Table 9. This Table also reveals a clear distinction between the running times of the algorithms. For large sample sizes, the Beliakov algorithm exhibits a significantly longer running time compared to the Local Optimization algorithm regardless of which minimal change condition is implemented.

TABLE 9: Running time in seconds for $\mathcal{N}(0, 0.3^2)$ distributed noise for the Beliakov and Local Optimization algorithm

Running time	$n = 100$	$n = 1000$	$n = 10000$
Original	0.0737	0.471	17.0
Boundary 6	0.211	1.33	25.3
Beliakov	0.0245	1.03	1010

It can be concluded that the Local Optimization algorithm can output similar approximations to the Beliakov algorithm. While for very small sample sizes the Beliakov algorithm demonstrates superior speed, for larger sample sizes the Local Optimization algorithm is significantly faster.

7 Discussion

In this section, we delve into the evaluation of the testing and the results of the Beliakov and the Local Optimization algorithm.

For sample sizes less or equal to 1000, the Local Optimization algorithm exhibits a smaller maximum error compared to Beliakov's algorithm. Given that the approximation of Beliakov's algorithm is considered to be the "true" approximation, this observation raises the question of whether the supremum norm is the most appropriate measure to quantify its performance. Perhaps the mean error over the data points provides a better assessment of its accuracy. Another plausible explanation for this observation is that the algorithms were tested with different data sets. For the testing of the Beliakov algorithm data was generated and for the testing of the Local Optimization algorithm new data was generated. Especially for small sample sizes, this can influence the outcome. It would have been better to generate data and use that to test both algorithms. However, since every category was run 50 times this difference is small. For the testing with larger noise, the two algorithms

were tested with the same data sets.

In the case of non-uniform distributions where a small number of data points cover a significant portion of the entire interval of $[0, 1]$, the Local Optimization algorithm demonstrates poor performance with the original boundary condition. However, by implementing a stricter minimal change condition, this issue is addressed effectively. On the other hand, for uniform distributed data, the approximation does not show any notable improvement with the stricter boundary, suggesting that the original boundary condition suffices. This prompts the question of whether an alternative method of selecting the random interval should be considered. One idea could be to weigh the data points based on the space they cover or the number of data points that are in their vicinity. This approach aims to ensure both accuracy and efficiency in terms of computational time.

Right now the Local Optimization algorithm looks quite promising, as it can output a similar result as the Beliakov algorithm in a fraction of the time. Exploring the performance of the Local Optimization algorithm in multiple dimensions would be extremely interesting.

References

- [1] Cvxopt: Python software for convex optimization.
- [2] Gleb Beliakov. Smoothing lipschitz functions. *Optimization Methods and Software*, 22(6):901 – 916, 2007.
- [3] Luc Devroye. *Non-Uniform Random Variate Generation*, chapter 2. Springer-Verlag, 1986.
- [4] Johannes Schmidt-Hieber and Petr Zamolodtchikov. Local convergence rates of the least squares estimator with applications to transfer learning, 2022.
- [5] Mícheál Ó Searcóid. *Metric Spaces*, chapter 9. Springer, 2007.
- [6] A.G. Sukharev. Optimal method of constructing best uniform approximations for functions of a certain class. *USSR Computational Mathematics and Mathematical Physics*, 18(2):21–31, 1978.

A Appendix

A.1 Data sets of the running time and error per category

TABLE 10: Mean maximum error of the Beliakov algorithm per category and sample size

X-distribution	Function	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 1\ 000$	$n = 5\ 000$
X_0	f_0	0.0860	0.0714	0.0547	0.0422	0.0369	0.0222
X_0	f_1	0.0816	0.0667	0.0477	0.0428	0.0332	0.0199
X_0	f_2	0.0762	0.0665	0.0527	0.0413	0.0332	0.0180
X_0	f_3	0.0890	0.0741	0.0530	0.0423	0.0340	0.0214
X_1	f_0	0.0957	0.0827	0.0634	0.0535	0.0497	0.0302
X_1	f_1	0.0872	0.0723	0.0639	0.0562	0.0481	0.0325
X_1	f_2	0.0794	0.0698	0.0521	0.0542	0.0429	0.0292
X_1	f_3	0.0859	0.0724	0.0605	0.0552	0.0435	0.0327
X_2	f_0	0.0909	0.0786	0.0607	0.0554	0.0471	0.0299
X_2	f_1	0.0818	0.0757	0.0622	0.0547	0.0435	0.0337
X_2	f_2	0.0938	0.0704	0.0623	0.0575	0.0492	0.0395
X_2	f_3	0.0878	0.0758	0.0671	0.0555	0.0465	0.0327
X_3	f_0	0.0847	0.0811	0.0679	0.0649	0.0547	0.0406
X_3	f_1	0.0806	0.0814	0.0637	0.0556	0.0498	0.0351
X_3	f_2	0.0871	0.0889	0.0765	0.0623	0.0593	0.0470
X_3	f_3	0.0864	0.0788	0.0627	0.0597	0.0499	0.0393

TABLE 11: Mean maximum error of the Local Optimization algorithm per category and sample size

X-distribution	Function	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 1\ 000$	$n = 5\ 000$	$n = 10\ 000$
X_0	f_0	0.0829	0.0680	0.0498	0.0428	0.0341	0.0205	0.0166
X_0	f_1	0.0775	0.0627	0.0507	0.0408	0.0327	0.0193	0.0152
X_0	f_2	0.0819	0.0701	0.0509	0.0446	0.0409	0.0493	0.0516
X_0	f_3	0.0883	0.0705	0.0511	0.0425	0.0354	0.0202	0.0156
X_1	f_0	0.0788	0.0693	0.0617	0.0493	0.0391	0.0309	0.0253
X_1	f_1	0.0809	0.0693	0.0557	0.0494	0.0450	0.0302	0.0263
X_1	f_2	0.0974	0.0832	0.0693	0.0601	0.0598	0.0663	0.0722
X_1	f_3	0.0829	0.0842	0.0686	0.0546	0.0569	0.0365	0.0353
X_2	f_0	0.0825	0.0726	0.0538	0.0490	0.0405	0.0278	0.0244
X_2	f_1	0.0848	0.0739	0.0578	0.0497	0.0414	0.0279	0.0254
X_2	f_2	0.0913	0.0785	0.0709	0.0621	0.0524	0.0643	0.0686
X_2	f_3	0.0871	0.0753	0.0626	0.0533	0.0493	0.0330	0.0261
X_3	f_0	0.0761	0.0614	0.0460	0.0385	0.0325	0.0203	0.0165
X_3	f_1	0.0714	0.0704	0.0504	0.0471	0.0394	0.0283	0.0218
X_3	f_2	0.0954	0.1022	0.1114	0.1151	0.1242	0.1316	0.1329
X_3	f_3	0.0861	0.0793	0.0627	0.0601	0.0503	0.0370	0.0298

TABLE 12: Mean running time in seconds of the Beliakov algorithm per category and sample size

X-distribution	Function	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 1\ 000$	$n = 5\ 000$
X_0	f_0	0.0108	0.0247	0.0536	0.162	0.91	62.5
X_0	f_1	0.0121	0.0269	0.0566	0.195	1.20	77.5
X_0	f_2	0.0138	0.0315	0.0729	0.246	1.21	101.2
X_0	f_3	0.0128	0.0288	0.0614	0.191	1.03	81.2
X_1	f_0	0.0125	0.0287	0.0576	0.176	0.91	62.8
X_1	f_1	0.0141	0.0323	0.0686	0.197	1.14	83.6
X_1	f_2	0.0141	0.0319	0.0667	0.205	1.12	105.9
X_1	f_3	0.0119	0.0238	0.0562	0.201	1.00	76.8
X_2	f_0	0.0126	0.0282	0.0519	0.202	0.95	64.0
X_2	f_1	0.0166	0.0298	0.0596	0.182	0.99	75.7
X_2	f_2	0.0166	0.0328	0.0724	0.197	1.09	104.4
X_2	f_3	0.0143	0.0316	0.0623	0.185	1.01	81.7
X_3	f_0	0.0219	0.0333	0.0632	0.177	0.90	64.3
X_3	f_1	0.0169	0.0309	0.0629	0.186	1.01	84.0
X_3	f_2	0.0157	0.0323	0.0641	0.190	1.10	104.9
X_3	f_3	0.0121	0.0281	0.0558	0.224	1.20	93.7

TABLE 13: Mean running time in seconds of the Local Optimization algorithm per category and sample size

X-distribution	Function	$n = 50$	$n = 100$	$n = 250$	$n = 500$	$n = 1\ 000$	$n = 5\ 000$	$n = 10\ 000$
X_0	f_0	0.0432	0.0772	0.1563	0.2962	0.6160	4.704	15.43
X_0	f_1	0.0457	0.0837	0.1647	0.2875	0.6165	4.721	15.36
X_0	f_2	0.0299	0.0600	0.1156	0.1856	0.3871	3.547	12.86
X_0	f_3	0.0504	0.0883	0.1654	0.3480	0.5816	4.523	15.38
X_1	f_0	0.0394	0.0836	0.1454	0.2667	0.5355	4.463	15.97
X_1	f_1	0.0348	0.0711	0.1360	0.2658	0.5294	4.361	14.25
X_1	f_2	0.0304	0.0503	0.0956	0.1876	0.3941	3.868	12.45
X_1	f_3	0.0446	0.0777	0.1516	0.3085	0.5495	4.445	14.08
X_2	f_0	0.0352	0.0667	0.1469	0.2663	0.5431	4.461	14.71
X_2	f_1	0.0411	0.0669	0.1398	0.2643	0.5308	4.338	14.02
X_2	f_2	0.0296	0.0477	0.0964	0.1714	0.3407	3.402	11.73
X_2	f_3	0.0375	0.0660	0.1323	0.2509	0.5281	4.341	13.74
X_3	f_0	0.0341	0.0726	0.1415	0.2666	0.5370	4.339	13.73
X_3	f_1	0.0395	0.0668	0.1289	0.2916	0.5355	4.283	13.70
X_3	f_2	0.0348	0.0492	0.0947	0.1625	0.3427	3.321	11.71
X_3	f_3	0.0393	0.0700	0.1430	0.2340	0.4683	3.910	13.03

A.2 Pseudocode

Input: $[X_1, \dots, X_n]$ and $[Y_1, \dots, Y_n]$
Initialize $y \leftarrow \left[\frac{1}{n} \sum_{i=1}^n Y_i \right]^n$, $\epsilon \leftarrow []$
Let $S = \{(a, b) \mid a, b \in [0, \dots, n], a \neq b, a < b\}$

while $\text{length}(\epsilon < 20)$ or $\frac{1}{60} \sum \epsilon_i > 1/(60n)$ **do**
 Pick J uniformly from S
 $lb \leftarrow \min J$, $ub \leftarrow \max J$
 $i_- \leftarrow lb - 1$, $i_+ \leftarrow ub + 1$
 $I \leftarrow [lb, lb + 1, \dots, ub - 1, ub]$
 $m \leftarrow \text{card}(I)$
 $\bar{y}_I \leftarrow \frac{1}{m} \sum_{i \in I} y_i$, $\bar{Y}_I \leftarrow \frac{1}{m} \sum_{i \in I} Y_i$
 if $\bar{y}_I = \bar{Y}_I$ **then** (it can't be optimized)
 $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, 0]$
 pass
 else(can be optimized)
 if $lb > 0$ and $ub < n$ **then**
 $u_- \leftarrow y_{i_-} + \sigma |X_{lb} - X_{i_-}|$
 $u_+ \leftarrow y_{i_+} + \sigma |X_{ub} - X_{i_+}|$
 if $|y_{lb} - u_-| < |y_{ub} - u_+|$ **then** (start left)
 $v_1 \leftarrow y_{lb} + \bar{Y}_I - \bar{y}_I$
 $v_{opt} \leftarrow \arg \min_{v \in \{v_1, u_-\}} |y_{lb} - v|$
 $\delta \leftarrow y_{lb} - v_{opt}$
 $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$
 $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$
 else (start right)
 $v_1 \leftarrow y_{ub} + \bar{Y}_I - \bar{y}_I$
 $v_{opt} \leftarrow \arg \min_{v \in \{v_1, u_+\}} |y_{ub} - v|$
 $\delta \leftarrow y_{ub} - v_{opt}$
 $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$
 $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$
 end if
 $ub, lb \leftarrow lb, ub$
 $I \leftarrow [0, \dots, ub]$ (left interval)
 $i_+ \leftarrow ub + 1$
 $m \leftarrow \text{card}(I)$, $\bar{y}_I \leftarrow \frac{1}{m} \sum_{i \in I} y_i$, $\bar{Y}_I \leftarrow \frac{1}{m} \sum_{i \in I} Y_i$
 $v_{opt} \leftarrow \arg \min_{v \in \{y_{ub} + \bar{Y}_I - \bar{y}_I, y_{i_+} - \sigma |X_{ub} - X_{i_+}|\}} |y_{ub} - v|$
 $\delta \leftarrow y_{lb} - v_{opt}$
 $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$
 $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$
 $I \leftarrow [lb, \dots, n]$ (right interval)
 $i_- \leftarrow lb - 1$
 $m \leftarrow \text{card}(I)$, $\bar{y}_I \leftarrow \frac{1}{m} \sum_{i \in I} y_i$, $\bar{Y}_I \leftarrow \frac{1}{m} \sum_{i \in I} Y_i$
 $v_{opt} \leftarrow \arg \min_{v \in \{y_{lb} + \bar{Y}_I - \bar{y}_I, y_{i_-} - \sigma |X_{lb} - X_{i_-}|\}} |y_{lb} - v|$
 $\delta \leftarrow y_{lb} - v_{opt}$
 $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$
 $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$
 end if


```

if  $\ell b = 0$  and  $ub < n$  then (left interval)
   $v_{opt} \leftarrow \arg \min_{v \in \{y_{ub} + \bar{Y}_I - \bar{y}_I, y_{i_+} - \sigma | X_{ub} - X_{i_+}\}} |y_{ub} - v|$ 
   $\delta \leftarrow y_{ub} - v_{opt}$ 
   $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$ 
   $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$ 
   $I \leftarrow [ub, \dots, n]$ ,  $i_- \leftarrow ub - 1$  (right interval)
   $m \leftarrow \text{card}(I)$ ,  $\bar{y}_I \leftarrow \frac{1}{m} \sum_{i \in I} y_i$ ,  $\bar{Y}_I \leftarrow \frac{1}{m} \sum_{i \in I} Y_i$ 
   $v_{opt} \leftarrow \arg \min_{v \in \{y_{ub} + \bar{Y}_I - \bar{y}_I, y_{i_-} - \sigma | X_{ub} - X_{i_-}\}} |y_{ub} - v|$ 
   $\delta \leftarrow y_{ub} - v_{opt}$ 
   $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$ 
   $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$ 
end if
if  $\ell b > 0$  and  $ub = n$  then (right interval)
   $v_{opt} \leftarrow \arg \min_{v \in \{y_{\ell b} + \bar{Y}_I - \bar{y}_I, y_{i_-} - \sigma | X_{\ell b} - X_{i_-}\}} |y_{\ell b} - v|$ 
   $\delta \leftarrow y_{\ell b} - v_{opt}$ 
   $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$ 
   $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$ 
   $I \leftarrow [0, \dots, \ell b]$ ,  $i_+ \leftarrow \ell b + 1$  (left interval)
   $m \leftarrow \text{card}(I)$ ,  $\bar{y}_I \leftarrow \frac{1}{m} \sum_{i \in I} y_i$ ,  $\bar{Y}_I \leftarrow \frac{1}{m} \sum_{i \in I} Y_i$ 
   $v_{opt} \leftarrow \arg \min_{v \in \{y_{\ell b} + \bar{Y}_I - \bar{y}_I, y_{i_+} - \sigma | X_{\ell b} - X_{i_+}\}} |y_{\ell b} - v|$ 
   $\delta \leftarrow y_{\ell b} - v_{opt}$ 
   $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$ 
   $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$ 
end if
if  $\ell b = 0$  and  $ub = n$  then (entire interval)
   $\delta \leftarrow \bar{Y}_I - \bar{y}_I$ 
   $[y_i]_{i \in I} \leftarrow [y_i + \delta]_{i \in I}$ 
   $\epsilon \leftarrow [\epsilon_{-59}, \dots, \epsilon_1, |\delta|]$ 
end if
end if
end while

```