A robust method based on LOVO functions for solving least squares problems*

E. V. Castelani† R. Lopes† W. V. I. Shirabayashi†
F. N. C. Sobral †‡

November 16, 2020

Abstract

The robust adjustment of nonlinear models to data is considered in this paper. When data comes from real experiments, it is possible that measurement errors cause the appearance of discrepant values, which should be ignored when adjusting models to them. This work presents a Low Order-value Optimization (LOVO) version of the Levenberg-Marquardt algorithm, which is well suited to deal with outliers in fitting problems. A general algorithm is presented and convergence to stationary points is demonstrated. Numerical results show that the algorithm is successfully able to detect and ignore outliers without too many specific parameters. Parallel and distributed executions of the algorithm are also possible, allowing the use of larger datasets. Comparison against publicly available robust algorithms shows that the present approach is able to find better adjustments in well known statistical models.

AMS: 47N10, 65Y05, 90C26, 93E24
Keywords: Low Order-Value Optimization, Levenberg-Marquardt, Outlier Detection, Robust Least Squares

1 Introduction

In this work we are interested in studying the following problem: given a dataset $\mathcal{R} = \{(t_i, y_i), i = 1, ..., r\}$ of points in $\mathbb{R}^m \times \mathbb{R}$, resulting from some experiment, we want to find a model $\varphi : \mathbb{R}^m \to \mathbb{R}$ for fitting this dataset free from influence of possible outliers. In a more precise way, given a model $\varphi(t)$ depending on n parameters ($x \in \mathbb{R}^n$), that is, $\varphi(t) = \phi(x, t)$, we want to find a set $\mathcal{P} \subset \mathcal{R}$ with p elements and parameters $\overline{x} \in \mathbb{R}^n$, such that $\phi(\overline{x}, t_i) \approx y_i$, $\forall (t_i, y_i) \in \mathcal{P}$ (in the least squares sense). The $r - p$ remaining elements in $\mathcal{R} - \mathcal{P}$ are the possible outliers.

There are several definitions of what an outlier is. The definition that best suits the present work concerns to errors in y_i, that is, grotesque errors in evaluation of some measure for a given and reasonably precise t_i. This is somewhat different from the geometric interpretation of outliers, in the sense that the

[*] This project was supported by Fundação Araucária, proc. number 002/2017 – 47223

[†] Department of Mathematics, State University of Maringá, Paraná, Brazil

[‡] ✉ Corresponding author, fncsobral@uem.br

point (t_i, y_i) is (geometrically) very far from the graph of a function that one wants to find. Typically in our tests, outliers are present when there are errors resulting from the measurement of some experiment. As a consequence, their presence may contaminate the obtained model and, therefore, deteriorate or limit its use. There are several strategies to handle the presence of outliers in datasets [9, 12, 22, 23]. In a more recent approach, as highlighted by [13] and references therein, techniques based on machine learning are exploited in the context of deal with a large amount of data, lack of models and categorical variables.

In order to get a fitting model free from influence of outliers, we use an approach based on *Low Order-Value Optimization* (LOVO) [5] which is defined as follows. Consider $R_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, ..., r$. Given $x \in \mathbb{R}^n$, we can sort $\{R_i(x), i = 1, ..., r\}$ in ascending order:

$$R_{i_1(x)}(x) \leq R_{i_2(x)}(x) \leq ... \leq R_{i_k(x)}(x) \leq \cdots \leq R_{i_r(x)}(x), \tag{1}$$

where $i_k(x)$ is the i_k-th smallest element in that set, for the given value of x. Given $0 < p \leq r$, the LOVO function is defined by

$$S_p(x) = \sum_{k=1}^{p} R_{i_k(x)}(x) \tag{2}$$

and the LOVO problem is

$$\min S_p(x). \tag{3}$$

Essentially, this problem can be seen as a generalization of nonlinear least squares, as elucidated in [5]. To reiterate this affirmation, we can consider $\varphi(t) = \phi(x, t)$ as the model selected for fitting, and define $R_i(x) = \frac{1}{2}(F_i(x))^2$, where $F_i(x) = y_i - \phi(x, t_i), i = 1, ..., r$. Thus, we have the particular LOVO problem

$$\min S_p(x) = \min \sum_{k=1}^{p} R_{i_k(x)}(x) = \min \sum_{k=1}^{p} \frac{1}{2}(F_{i_k(x)}(x))^2. \tag{4}$$

Each R_i is a residual function. Consequently, if we assume $p = r$ the LOVO problem is the classical least squares problem. When $p < r$ the parameter $\bar{x} \in \mathbb{R}^n$ that solves (4) defines a model $\phi(\bar{x}, t)$ free from the influence of the worst $r - p$ deviations. Throughout this work, p is also know as the number of *trusted* points.

Several applications can be modeled in the LOVO context, as illustrated in [5, 4, 7, 16, 18]. LOVO problems originated in the studies of Order Value Optimization (OVO) problems [2, 3]. For more details on the relationship between these problems, see reference [5]. An excellent survey about LOVO problems and variations is given in [17]. Although it is well known that LOVO deals with detection of outliers, there is a limitation: the mandatory definition of the value p, which is associated to the number of possible outliers. This is the main gap that this paper intends to fill. We present a new method that combines a voting schema and an adaptation of the Levenberg-Marquardt algorithm in context of LOVO problems.

Levenberg-Marquardt algorithms can be viewed as a particular case of trust-region algorithms, using specific models to solve nonlinear equations. In [4], a

LOVO trust-region algorithm is presented with global and local convergence properties and an application to protein alignment problems. Second-order derivatives were needed in the algorithm for the local convergence analysis. In least-squares problems, as the objective function has a well known structure, Levenberg-Marquardt algorithms use a linear model for the adjustment function, instead of a quadratic model for the general nonlinear function. This approach eliminates the necessity of using second-order information for the model while still having second-order information about the function to be minimized. More details of Levenberg-Marquardt methods can be found in [21].

Another approach that also uses first-order information of models to obtain second-order approximation in least-squares functions is the Gauss-Newton method. Gauss-Newton in the context of LOVO functions for image recognition was discussed in [1]. The authors presented a LOVO approach to the detection of lines and circles with fixed radii. Line-search was used for obtaining global convergence. The main drawbacks of this approach were that near-singularity of the Gauss-Newton system had to be fixed in a heuristic way and the number p of trusted points had to be previously estimated and fixed for each problem. The algorithm was shown to be very efficient against state-of-art algorithms, when the number of parameters to be estimated started to increase.

In [24], outlier detection techniques are classified in 7 groups for problems of data streaming: Statistic-based, depth-based, deviation-based, distance-based, clustering-based, sliding-window-based and autoregression-based. In [26], classification is divided only between geometric and algebraic algorithms for robust curve and surface fitting. The approach used in this work is clearly algebraic, strongly based in the fact that the user knows what kind of model is to be used. Although models are used, we make no assumption on the distribution of the points, so we do not fit clearly in any of the types described in [24]. We also make the assumption that the values t_i are given exactly, what is called as *fixed-regressor model* in [21].

This work deals with the robust adjustment of models to data. A new version of the Levenberg-Marquardt algorithm for LOVO problems is developed, so the necessity of second-order information of function R_i is avoided. In addition, the number of possible outliers is estimated by a voting schema. The main difference of the proposed voting schema is that it is based in the values of p which has, by definition, a discrete domain. In other techniques, such as the Hough Transform [14, 10, 15, 25], continuous intervals of the model's parameters are discretized. Also, the increase in the number of parameters to adjust does not impact the proposed voting system. The main improvements of this work can be stated as follows

- a Levenberg-Marquardt algorithm with global convergence for LOVO problems is developed, which avoids the use of second-order information such as in [4] or heuristic strategies to improve conditioning as in [1];

- a voting schema based on the values of p is developed, whose size does not increase with the size or discretization of the parameters of the model, such that the number of trusted points does not have to be previously estimated as in [1];

- extensive numerical results are presented, which show the behavior of the proposed method and are also freely available for download.

This work is organized as follows. In Section 2 we describe the Levenberg-Marquardt algorithm in the LOVO context and demonstrate its convergence properties. In Section 3 the voting schema is discussed, which will make the LOVO algorithm independent of the choice of p and will be the basis of the robust fitting. Section 4 is devoted to the discussion of the implementation details and comparison against other algorithms for robust fitting. Finally, in Section 5 we draw some conclusions on the presented strategy. Throughout this paper we use the notation $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x \geq 0\}$.

2 The Levenberg-Marquardt method for LOVO problems

Following [5], let us start this section by pointing out an alternative definition of LOVO problems for theoretical purposes. Denoting $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_q\}$ the set of all combinations of the elements $\{1, 2, ..., r\}$ taken p at a time, we can define for each $i \in \{1, ..., q\}$ the following functions

$$f_i(x) = \sum_{k \in \mathcal{C}_i} R_k(x) \tag{5}$$

and

$$f_{min}(x) = \min\{f_i(x), i = 1, ..., q\}. \tag{6}$$

It is simple to note that $S_p(x) = f_{min}(x)$ which is a useful notation in our context. Moreover, it is possible to note that S_p is a continuous function if f_i is a continuous function for all $i = 1, ..., q$, but, even assuming differentiability over f_i, we cannot guarantee the same for S_p. In addition, since $R_k(x) = \frac{1}{2}(F_k(x))^2, k \in \mathcal{C}_i, i = 1, ..., q$ we can write

$$f_i(x) = \frac{1}{2} \sum_{k \in \mathcal{C}_i} F_k(x)^2 = \frac{1}{2} \|F_{\mathcal{C}_i}(x)\|_2^2. \tag{7}$$

Throughout this work, following (7), given a set $\mathcal{C}_i \in \mathcal{C}$, $F_{\mathcal{C}_i}(x) : \mathbb{R}^n \to \mathbb{R}^p$ will always refer to the map that takes x to the p-sized vector composed by the functions $F_k(x)$ defined by (4), for $k \in \mathcal{C}_i$ in any fixed order. Similarly, $J_{\mathcal{C}_i}(x)$ is defined as the Jacobian of this map. Additionally, we assume the continuous differentiability for F_i, $i = 1, ..., r$.

The goal of this section is to define a version of Levenberg-Marquardt method (LM) to solve the specific problem (4), for a given p, as well as a result on global convergence. The new version will be called by simplicity `LM-LOVO`. It is well known that the Levenberg-Marquardt method proposed in [19] is closely related to trust-region methods and our approach is based on it. Consequently, some definitions and remarks are necessary.

Definition 2.1. Given $x \in \mathbb{R}^n$ we define the *minimal function set of f_{min} in x* by

$$I_{min}(x) = \{i \in \{1, \ldots, q\} \mid f_{min}(x) = f_i(x)\}.$$

In order to define a search direction for `LM-LOVO` at the current point x_k, we choose an index $i \in I_{min}(x_k)$ and compute the direction defined by the classical

Levenberg-Marquardt method using $f_i(x)$, that is, the search direction $d_k \in \mathbb{R}^n$ is defined as the solution of

$$\min_{d\in\mathbb{R}^n} m_{k,i}(d) = \frac{1}{2}\|F_{\mathcal{C}_i}(x_k) + J_{\mathcal{C}_i}(x_k)d\|_2^2 + \frac{\gamma_k}{2}\|d\|_2^2, \tag{8}$$

where $\gamma_k \in \mathbb{R}_+$ is the *damping parameter.* Equivalently, the direction d can be obtained by

$$(J_{\mathcal{C}_i}(x_k)^T J_{\mathcal{C}_i}(x_k) + \gamma_k I)d = -\nabla f_i(x_k), \tag{9}$$

where $\nabla f_i(x_k) = J_{\mathcal{C}_i}(x_k)^T F_{\mathcal{C}_i}(x_k)$ and $I \in \mathbb{R}^{n\times n}$ is the identity matrix.

To ensure sufficient decrease in the defined search direction, we can consider a similar strategy of trust-region methods, which involves monitoring the actual decrease (given by f_{min}) and the predicted decrease (given by $m_{k,i}$) at direction d_k:

$$\rho_{k,i} = \frac{f_{min}(x_k) - f_{min}(x_k + d_k)}{m_{k,i}(0) - m_{k,i}(d_k)}. \tag{10}$$

We formalize the conceptual algorithm `LM-LOVO` in the Algorithm 1.

A noteworthy property of Levenberg-Marquardt (and also Gauss-Newton) coupled with the LOVO approach is that, assuming that the exact model was chosen, the number of trusted points p was correctly identified, $i \in I_{min}(x_k)$ and there is no relevant noise in observations y_j, $j \in \mathcal{C}_i$, then the Hessian of the model, $\nabla^2 m_{k,i}(d) = J_{\mathcal{C}_i}(x_k)^T J_{\mathcal{C}_i}(x_k) + \gamma_k I$ approximates the Hessian of f_i, $\nabla^2 f_i(x_k) = J_{\mathcal{C}_i}(x_k)^T J_{\mathcal{C}_i}(x_k) + \sum_{j\in\mathcal{C}_i} \nabla^2 F_j(x_k)F_j(x_k)$, as the algorithm converges. This occurs because $F_j(x_k) \to 0$, $j \in \mathcal{C}_i$, and $\gamma_k \to 0$ (see the definition of γ_k and Theorem 2.6). This property would not occur if a traditional Levenberg-Marquardt algorithm is applied to data with outliers.

In what follows, we show that Algorithm 1 is well defined and converges to stationary points of the LOVO problem. We begin with some basic assumptions on the boundedness of the points generated by the algorithm and on the smoothness of the involved functions.

Assumption 2.2. *The level set*

$$C(x_0) = \{x \in \mathbb{R}^n \mid f_{min}(x) \le f_{min}(x_0)\}$$

is a bounded set of $\mathbb{R}^n$ *and the functions* f_i*,* $i = 1, \dots, q$*, have Lipschitz continuous gradients with Lipschitz constants* $L_i > 0$ *in an open set containing* $C(x_0)$*.*

The next proposition is classical in the literature of trust-region methods and ensures decrease of $m_{k,i_k}(.)$ on the Cauchy direction. It was adapted to the LOVO context.

Proposition 2.3. *Given* $x_k \in \mathbb{R}^n$*,* $\gamma_k \in \mathbb{R}_+$ *and* $i_k \in \{1, \dots, q\}$*, the Cauchy step obtained from*

$$\widehat{t} = \operatorname*{arg\,min}_{t\in\mathbb{R}} \quad \{m_{k,i_k}(-t\nabla f_{i_k}(x_k))\}$$

and expressed by $d^C(x_k) = -\widehat{t}\nabla f_{i_k}(x_k) \in \mathbb{R}^n$*, satisfies*

$$m_{k,i_k}(0) - m_{k,i_k}(d^C(x_k)) \ge \frac{\theta\|\nabla f_{i_k}(x_k)\|_2^2}{2(\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2 + \gamma_k)}, \tag{11}$$

for some $\theta > 0$*, independent of* k*.*

Algorithm 1: `LM-LOVO` – Levenberg-Marquardt for the LOVO problem.

```
  Input: $x_0 \in \mathbb{R}^n$, $0 < \lambda_{min} \leq \lambda_0$, $\varepsilon \geq 0$, $\overline{\lambda} > 1, \mu \in (0,1)$ and $p \in \mathbb{N}$
  Output: $x_k$
  Set $k \leftarrow 0$;
1 Select $i_k \in I_{min}(x_k)$;
  $\lambda \leftarrow \lambda_k$;
2 if $\|\nabla f_{i_k}(x_k)\|_2 \leq \varepsilon$ then
      Stop the algorithm, $x_k$ is an approximate solution for the LOVO
        problem;
3 $\gamma_k \leftarrow \lambda \|\nabla f_{i_k}(x_k)\|_2^2$;
  Compute $d_k$ the solution of the linear system (9);
  Calculate $\rho_{k,i_k}$ as described in (10);
4 if $\rho_{k,i_k} < \mu$ then
      $\lambda \leftarrow \overline{\lambda}\lambda$;
      Go back to the Step 3;
  else
      Go to the Step 5;
5 $\lambda_{k+1} \in [\max\{\lambda_{min}, \lambda/\overline{\lambda}\}, \lambda]$;
  $x_{k+1} \leftarrow x_k + d_k$;
  $k \leftarrow k + 1$ and go back to the Step 1 ;
```

Proof. The Cauchy step is explicitly given by

$$d^C(x_k) = -\frac{\|\nabla f_{i_k}(x_k)\|_2^2}{\|J_{\mathcal{C}_{i_k}}(x_k)\nabla f_{i_k}(x_k)\|_2^2 + \gamma_k \|\nabla f_{i_k}(x_k)\|_2^2} \nabla f_{i_k}(x_k).$$

By simple substitution in m_{k,i_k} (defined by (8)), and observing that $\nabla f_{i_k}(x_k) = J_{\mathcal{C}_{i_k}}(x_k)^T F_{\mathcal{C}_{i_k}}(x_k)$ and $\|J_{\mathcal{C}_{i_k}}(x_k)\nabla f_{i_k}(x_k)\|_2 \leq \|J_{\mathcal{C}_{i_k}}(x_k)\|_2 \|\nabla f_{i_k}(x_k)\|_2$, it is not hard to show that

$$m_{k,i_k}(0) - m_{k,i_k}(d^C(x_k)) \geq \frac{1}{2} \frac{\|\nabla f_{i_k}(x_k)\|_2^2}{\left(\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2 + \gamma_k\right)},$$

and (11) holds if we define $\theta \in (0,1)$. □

Since the Cauchy step is obtained by the constant that minimizes the model $m_{k,i_k}(.)$ on the direction of the gradient vector, by Proposition 2.3 we can conclude that there exists $\theta > 0$ such that

$$m_{k,i_k}(0) - m_{k,i_k}(d_k) \geq \frac{\theta \|\nabla f_{i_k}(x_k)\|_2^2}{2(\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2 + \gamma_k)}, \tag{12}$$

since $d_k \in \mathbb{R}^n$ from (9) is the global minimizer of m_{k,i_k}.

Inspired by [6], we present Lemma 2.4 that shows that Step 5 is always executed by Algorithm 1 if λ is chosen big enough.

Lemma 2.4. *Let $x_k \in \mathbb{R}^n$ and $i_k \in I_{min}(x_k)$ be a vector and an index, respectively, both fixed in the Step 1 of the Algorithm 1. Then, the Step 3 of the Algorithm 1 will be executed a finite number of times.*

Proof. To achieve this goal, we will show that

$$\lim_{\lambda\to\infty} \rho_{k,i_k} \geq 2.$$

For each λ fixed in the Step 1 of the Algorithm 1, we have that

$$\begin{aligned} 1-\frac{\rho_{k,i_k}}{2} &= 1-\frac{f_{min}(x_k)-f_{min}(x_k+d_k)}{2(m_{k,i_k}(0)-m_{k,i_k}(d_k))} \\ &= \frac{2m_{k,i_k}(0)-2m_{k,i_k}(d_k)-f_{min}(x_k)+f_{min}(x_k+d_k)}{2(m_{k,i_k}(0)-m_{k,i_k}(d_k))} \\ &= \frac{f_{min}(x_k+d_k)+f_{min}(x_k)-2m_{k,i_k}(d_k)}{2(m_{k,i_k}(0)-m_{k,i_k}(d_k))}. \end{aligned} \tag{13}$$

From Taylor series expansion and the Lipschitz continuity of $\nabla f_{i_k}(x_k)$

$$f_{i_k}(x_k+d_k) \leq f_{i_k}(x_k)+\nabla f_{i_k}(x_k)^T d_k+\frac{L_{i_k}}{2}\|d_k\|_2^2. \tag{14}$$

By equation (14) and the definition of f_{min}, we obtain

$$f_{min}(x_k+d_k) \leq f_{i_k}(x_k+d_k) \overset{(14)}{\leq} f_{i_k}(x_k)+\nabla f_{i_k}(x_k)^T d_k+\frac{L_{i_k}}{2}\|d_k\|_2^2. \tag{15}$$

Through the expressions (7) and (9), we have

$$\begin{aligned} &\|F_{\mathcal{C}_{i_k}}(x_k)+J_{\mathcal{C}_{i_k}}(x_k)d_k\|_2^2+\gamma_k\|d_k\|_2^2 = \\ &\quad= \|F_{\mathcal{C}_{i_k}}(x_k)\|_2^2+2F_{\mathcal{C}_{i_k}}(x_k)^T J_{\mathcal{C}_{i_k}}(x_k)d_k+\|J_{\mathcal{C}_{i_k}}(x_k)d_k\|_2^2+\gamma_k\|d_k\|_2^2 \\ &\quad= \|F_{\mathcal{C}_{i_k}}(x_k)\|_2^2+2F_{\mathcal{C}_{i_k}}(x_k)^T J_{\mathcal{C}_{i_k}}(x_k)d_k \\ &\quad+ d_k^T\left(J_{\mathcal{C}_{i_k}}(x_k)^T J_{\mathcal{C}_{i_k}}(x_k)+\gamma_k I\right)d_k \\ &\quad\overset{(7),(9)}{=} 2f_{i_k}(x_k)+2\nabla f_{i_k}(x_k)^T d_k-d_k^T\nabla f_{i_k}(x_k) \\ &\quad= 2f_{i_k}(x_k)+\nabla f_{i_k}(x_k)^T d_k. \end{aligned} \tag{16}$$

Using (15), (16) and the definition of m_{k,i_k} in (13), we get

$$\begin{aligned} 1-\frac{\rho_{k,i_k}}{2} &= \frac{f_{min}(x_k+d_k)+f_{i_k}(x_k)-\|F_{\mathcal{C}_{i_k}}(x_k)+J_{\mathcal{C}_{i_k}}(x_k)d_k\|_2^2-\gamma_k\|d_k\|_2^2}{2(m_{k,i_k}(0)-m_{k,i_k}(d_k))} \\ &\overset{(16)}{=} \frac{f_{min}(x_k+d_k)-f_{i_k}(x_k)-\nabla f_{i_k}(x_k)^T d_k}{2(m_{k,i_k}(0)-m_{k,i_k}(d_k))} \\ &\overset{(15)}{\leq} \frac{L_{i_k}\|d_k\|_2^2}{4(m_{k,i_k}(0)-m_{k,i_k}(d_k))}. \end{aligned} \tag{17}$$

From (9) and the definition of γ_k, we note that

$$\|d_k\|_2 \leq \frac{\|\nabla f_{i_k}(x_k)\|_2}{\sigma_k+\gamma_k} \leq \frac{\|\nabla f_{i_k}(x_k)\|_2}{\gamma_k} = \frac{1}{\|\nabla f_{i_k}(x_k)\|_2\lambda}, \tag{18}$$

where $\sigma_k=\sigma_{min}(J_{\mathcal{C}_{i_k}}(x_k)^T J_{\mathcal{C}_{i_k}}(x_k))$ and $\sigma_{min}(B)$ represents the smallest eigenvalue of B.

Replacing (18) in (17), we obtain

$$
\begin{aligned}
1-\frac{\rho_{k,i_k}}{2} &\le \frac{\dfrac{L_{i_k}}{\|\nabla f_{i_k}(x_k)\|_2^2\lambda^2}}{4(m_{k,i_k}(0)-m_{k,i_k}(d_k))} \overset{(12)}{\le} \frac{\dfrac{L_{i_k}}{\|\nabla f_{i_k}(x_k)\|_2^2\lambda^2}}{\dfrac{4\theta\|\nabla f_{i_k}(x_k)\|_2^2}{2(\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2+\gamma_k)}} \\
&= \frac{(\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2+\gamma_k)L_{i_k}}{2\theta\|\nabla f_{i_k}(x_k)\|_2^4\lambda^2} \le \left(\frac{\|J_{\mathcal{C}_{i_k}}(x_k)\|_2^2}{\|\nabla f_{i_k}(x_k)\|_2^4}+\frac{1}{\|\nabla f_{i_k}(x_k)\|_2^2}\right)\frac{L_{i_k}}{2\theta\lambda},
\end{aligned}
\tag{19}
$$

where the last inequality comes from the definition of γ_k in Algorithm 1 and assuming that $\lambda \ge 1$, which can always be enforced.

Using (19), we conclude that

$$\lim_{\lambda\to\infty} 1-\frac{\rho_{k,i_k}}{2} \le 0,$$

or equivalently

$$\lim_{\lambda\to\infty} \rho_{k,i_k} \ge 2,$$

which proves the result. □

Our studies move toward showing convergence results for Algorithm 1 to stationary points. At this point we should be aware of the fact that LOVO problems admit two types of stationary condition: *weak* and *strong* [4].

Definition 2.5. A point x^* is a weakly critical point of (3) when x^* is a stationary point of f_i for some $i \in I_{min}(x^*)$. A point x^* is a strongly critical point of (3) if x^* is a stationary point of f_i for all $i \in I_{min}(x^*)$.

The global convergence to weakly critical points is given by Theorem 2.6. This type of convergence is less expensive to verify in practice and therefore more common to deal with. Convergence to strongly critical points is theoretically interesting and can be accomplished by using the concept of δ-active indexes, defined in [5]. The algorithm, for such type of convergence has to be slightly modified. We provide the new algorithm and all the technical details in Appendix A.

Theorem 2.6. *Let $\{x_k\}_{k\in\mathbb{N}}$ be a sequence generated by Algorithm 1 by choosing $\varepsilon = 0$ and x^* a limit point of that sequence. Consider $\mathcal{K}' = \{k \mid i_k = i\} \subset \mathbb{N}$ an infinite subset of indexes for $i \in \{1,\ldots,q\}$ such that $\lim_{k\in\mathcal{K}'} x_k = x^*$ and assume that Assumption 2.2 holds. Then, we have*

$$\lim_{k\in\mathcal{K}'} \|\nabla f_i(x_k)\|_2 = 0$$

and $i \in I_{min}(x^*)$.

Proof. Clearly, there is an index i chosen an infinite number of times by Algorithm 1, since $\{1,\ldots,q\}$ is a finite set.

Let us suppose by contradiction that, for this index i, there exist $\beta > 0$ and an infinite subset $\mathcal{K}_1 \subset \mathcal{K}'$ such that $\|\nabla f_i(x_k)\|_2 \ge \beta$, for all $k \in \mathcal{K}_1$.

Using the continuity of the Jacobian $J_{\mathcal{C}_i(x)}$, we ensure that

$$\|J_{\mathcal{C}_i}(x_k)\|_2 \leq \sup_{k \in \mathcal{K}_1} \{\|J_{\mathcal{C}_i}(x_k)\|_2^2\} = J_i, \tag{20}$$

for all $k \in \mathcal{K}_1$.

By (19), for $\lambda \geq 1$ we obtain

$$\begin{aligned} & 1 - \frac{\rho_{k,i}}{2} \leq \left(\frac{\|J_{\mathcal{C}_i}(x_k)\|_2^2}{\|\nabla f_i(x_k)\|_2^4} + \frac{1}{\|\nabla f_i(x_k)\|_2^2} \right) \frac{L_i}{2\theta\lambda} \\ \Rightarrow & 1 - \frac{\rho_{k,i}}{2} \overset{(20)}{\leq} \left(\frac{J_i^2}{\beta^4} + \frac{1}{\beta^2} \right) \frac{L_i}{2\theta\lambda} \\ \Rightarrow & \rho_{k,i} \geq 2 - \left(\frac{J_i^2}{\beta^4} + \frac{1}{\beta^2} \right) \frac{L_i}{\theta\lambda}, \end{aligned} \tag{21}$$

for all $k \in \mathcal{K}_1$.

Through expression (21), we have that Step 5 of Algorithm 1 will certainly be executed when $\lambda \geq b = \max\left\{1, \left(\frac{J_i^2}{\beta^4} + \frac{1}{\beta^2} \right) \frac{L_i}{\theta} \right\}$ since

$$\rho_{k,i} \geq 2 - \left(\frac{J_i^2}{\beta^4} + \frac{1}{\beta^2} \right) \frac{L_i}{\theta\lambda} \geq 2 - \left(\frac{J_i^2}{\beta^4} + \frac{1}{\beta^2} \right) \frac{L_i}{\theta b} \geq 1 > \mu, \tag{22}$$

for all $k \in \mathcal{K}_1$. Therefore, based on Step 4 of Algorithm 1, the value of λ_k will be upper bounded by $\lambda_k \leq M = \overline{\lambda} b$, for all $k \in \mathcal{K}_1$.

Then, for all $k \in \mathcal{K}_1$, we get

$$\begin{aligned} & \frac{f_{min}(x_k) - f_{min}(x_{k+1})}{m_{k,i}(0) - m_{k,i}(d_k)} \geq \mu \\ & \Leftrightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \mu(m_{k,i}(0) - m_{k,i}(d_k)) \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \overset{(12)}{\geq} \mu \left(\theta \frac{\|\nabla f_i(x_k)\|_2^2}{2(\|J_{\mathcal{C}_i}(x_k)\|_2^2 + \gamma_k)} \right) \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \frac{\mu\theta\|\nabla f_i(x_k)\|_2^2}{2(\|J_{\mathcal{C}_i}(x_k)\|_2^2 + \lambda_k\|\nabla f_i(x_k)\|_2^2)} \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \frac{\mu\theta}{2\left(\frac{\|J_{\mathcal{C}_i}(x_k)\|_2^2}{\|\nabla f_i(x_k)\|_2^2} + \lambda_k \right)} \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \frac{\mu\theta}{2\left(\frac{\|J_{\mathcal{C}_i}(x_k)\|_2^2}{\beta^2} + \lambda_k \right)} \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \frac{\mu\theta\beta^2}{2(\|J_{\mathcal{C}_i}(x_k)\|_2^2 + \beta^2\lambda_k)} \\ & \Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \frac{\mu\theta\beta^2}{2(J_i^2 + \beta^2 M)} \\ & \Leftrightarrow f_{min}(x_{k+1}) - f_{min}(x_k) \leq -\frac{\mu\theta\beta^2}{2c}, \end{aligned} \tag{23}$$

where $c = J_i^2 + \beta^2 M$.

Expression (23) and the fact that $f_{min}(x_{k+1}) \leq f_{min}(x_k)$, for all $k \in \mathcal{K}'$, contradict the hypothesis that f_{min} is bounded from below. We conclude that there is no such $\mathcal{K}_1$ and, therefore,

$$\lim_{k \in \mathcal{K}'} \|\nabla f_i(x_k)\|_2 = 0.$$

To prove the second statement of the theorem, we use the fact that, in Algorithm 1, $i_k \in I_{min}(x_k)$, for all k, obtaining that

$$f_{i_k}(x_k) = f_i(x_k) \leq f_j(x_k), \ \forall \ k \in \mathcal{K}' \text{ and } \forall \ j \in \{1, \ldots, q\}. \tag{24}$$

By (24) and the continuity of the function f_i, for all $i \in \{1, \ldots, q\}$, we have

$$f_i(x^*) \leq f_j(x^*), \ \forall \ j \in \{1, \ldots, q\},$$

which means that $i \in I_{min}(x^*)$, concluding the proof. □

It is not hard to show that, if $i \in I_{min}(x_k)$, then all the previous theoretical results remain valid if we replace $\rho_{k,i}$ by

$$\hat{\rho}_{k,i} = \frac{f_i(x_k) - f_i(x_k + d_k)}{m_{k,i}(0) - m_{k,i}(d_k)},$$

where f_i was used instead of f_{min}. The main reason for using $\hat{\rho}_{k,i}$ is practical: when computing $f_i(x_k + d_k)$ we can use the same set $\mathcal{C}_i$ of index of functions F_j that was used when $f_i(x_k) = f_{min}(x_k)$ was computed. Recall that the computation of $f_{min}(x)$ requires the computation of all F_j, $j = 1, \ldots, r$, sorting their values and taking the smallest p functions. On the other hand, since $f_i(x_k + d_k) \geq f_{min}(x_k + d_k)$ and $f_i(x_k) = f_{min}(x_k)$, we have that

$$\hat{\rho}_{k,i} = \frac{f_i(x_k) - f_i(x_k + d_k)}{m_{k,i}(0) - m_{k,i}(d_k)} \leq \frac{f_{min}(x_k) - f_{min}(x_k + d_k)}{m_{k,i}(0) - m_{k,i}(d_k)} = \rho_{k,i},$$

which means that it might be necessary more executions of Step 3 before condition $\hat{\rho}_{k,i_k} \geq \mu$ is satisfied. Each execution of Step 3 involves the solution of a linear system. If the number r of functions is very large so that sorting is the bottleneck of the algorithm, the use of $\hat{\rho}_{k,i}$ can be an interesting alternative.

3 The voting system

The main drawback of Algorithm 1 is the need to know the number p of trusted points, which is used by S_p (2) (or, equivalently, by f_{min}). It is not usual to know the exact number of trusted points in any experiment.

To overcome this difficulty, an algorithm for testing different values of p was created, detailed by Algorithm 2. The main idea of the method is to call Algorithm 1 for several different values of p and store the obtained solution. The solutions are then preprocessed, where stationary points that are not global minimizers of their respective problem are eliminated. This elimination is based on the fact that, if $\bar{x}_p$ and $\bar{x}_q$, $p < q$, are solutions for their respective problems, then $S_p(\bar{x}_p)$ cannot be greater than $S_q(\bar{x}_q)$ if they are both global minimizers. Therefore, if $S_p(\bar{x}_p) > S_q(\bar{x}_q)$, then $\bar{x}_p$ is not a global minimizer and can be safely eliminated. The last steps (Steps 4 and 5) compute the similarity between each pair of solutions and obtain the most similar ones. Element C_p of vector C stores the number of times that some other solution was considered similar to $\bar{x}_p$, in the sense of a tolerance ϵ. The most similar solution with greatest p is considered the robust adjustment model for the problem. Algorithm 2 is a proposal of a voting system, where the solution that was not eliminated by the preprocessing and occurred with highest frequency (in the similarity sense) is selected.

Algorithm 2: Voting algorithm for fitting problems

```
  Input: $x_0 \in \mathbb{R}^n$, $\epsilon \in \mathbb{R}_+$ and $0 \leq p_{min} < p_{max}$
1 Define $C \in \mathbb{R}^s = \mathbf{0}$, where $s = p_{max} - p_{min} + 1$
2 Compute $\bar{x}_p \in \mathbb{R}^n$ by calling Algorithm 1 for the given $p$, for all
    $p \in \{p_{min}, p_{min} + 1, ..., p_{max}\}$
3 Preprocess solutions
4 Let $M_{pq}$ be the similarity between solutions $\bar{x}_p$ and $\bar{x}_q$
5 for $p = p_{min}, \ldots, p_{max}$ do
      $k \leftarrow 0$
      for $q = p_{min}, \ldots, p_{max}$ do
          if $M_{pq} < \epsilon$ then
              $k \leftarrow k + 1$
      $C_p \leftarrow k$
6 $x^\star \leftarrow \bar{x}_p$, where $p = \operatorname*{arg\,max}_{q = p_{min}, \ldots, p_{max}} \{C_q\}$
```

The execution of Algorithm 2 can be easily parallelizable. Each call of Algorithm 1 with a different value of p can be performed independently at Step 2. All the convergence results from Section 2 remain valid, so Algorithm 2 is well defined. All the specific implementation details of the algorithm are discussed in Section 4.

4 Numerical implementation and experiments

In this Section we discuss the implementation details of Algorithms 1 and 2. From now on, Algorithm 1 will be called `LM-LOVO` and Algorithm 2 will be called `RAFF`. Both algorithms were implemented in the Julia language, version 1.0.4 and are available in the official Julia repository. See [8] for information about the `RAFF.jl` package installation and usage.

Algorithm `LM-LOVO` is a sequential nonlinear programming algorithm, which means that only the traditional parallelization techniques can be applied. Since fitting problems have small dimension and a large dataset, the main gains would be the parallelization of the objective function, not the full algorithm. Matrix and vector operations are also eligible for parallelization.

Following traditional LOVO implementations [1], the choice of index $i_k \in I_{min}(x_k)$ is performed by simply evaluating functions $F_i(x_k)$, $i = 1, \ldots, r$, sorting them in ascending order and them dropping the $r - p$ largest values. Any sorting algorithm can be used, but we used our implementation of the selection sort algorithm. This choice is interesting, since the computational cost is linear when the vector is already in ascending order, what is not unusual if LM-LOVO is converging and $i_{k+1} = i_k$, for example.

The convergence theory needs the sufficient decrease parameter ρ_{k,i_k} to be calculated in order to define step acceptance and the update of the damping parameter. In practice, LM-LOVO uses the simple decrease test at Step 4

$$f_{min}(x_k + d_k) < f_{min},$$

which was shown to work well in practice.

The computation of direction d_k is performed by solving the linear system (9) by the Cholesky factorization of matrix $J_{\mathcal{C}_{i_k}}(x_k)^T J_{\mathcal{C}_{i_k}}(x_k) + \gamma_k I$. In the case where Steps 3 and 4 are repeated at the same iteration k, the QR factorization is more indicated, since it can be reused when the iterate x_k remains the same and only the dumping factor is changed. See [19] for more details about the use of QR factorizations in the Levenberg-Marquardt algorithm. If there is no interest in using the QR factorization, then the Cholesky factorization is recommended.

LM-LOVO was carefully implemented, since it is used as a subroutine of RAFF for solving adjustment problems. A solution $\bar{x} = x_k$ is declared as successful if

$$\|\nabla f_{i_k}(\bar{x})\|_2 \leq \varepsilon \qquad (25)$$

for some $i_k \in I_{min}(\bar{x})$, where f_{i_k} is given by (5). The algorithm stops if the gradient cannot be computed due to numerical errors or if the limit of 400 iterations has been reached. We also set $\overline{\lambda} = 2$ as default.

In order to show the behavior of LM-LOVO we solved the problem of adjusting some data to the one-dimensional logistic model, widely used in statistics

$$\phi(x,t) = x_1 + \frac{x_2}{1 + \exp(-x_3 t + x_4)},$$

where $x \in \mathbb{R}^4$ represents the parameters of the model and $t \in \mathbb{R}$ represents the variable of the model. In order to generate random data for the test, the procedures detailed in Subsection 4.1 were used. The produced data is displayed in Figure 1, where $r = 10$, $p = 9$ and the exact solution was $x^* = (6000, -5000, -0.2, -3.7)$. This example has only $r - p = 1$ outlier.

LM-LOVO was run with its default parameters, using $x = (0, 0, 0, 0)$ as a starting point and $p = 9$, indicating that there are 9 points which are trustable for adjusting the model. The solution found is also shown in Figure 1, given by $\bar{x} = (795.356, 5749.86, 0.161791, 3.02475)$, as a continuous line, while the "exact" solution is depicted as a dashed line. We observe that it is not expected the exact solution x^* to be found, since the points were perturbed. The outlier is correctly identified as the dark/red triangle.

The example in Figure 1 has an outlier that is visually easy to identify, so the correct number of $p = 9$ trusted points was used. However, that might not be the case, specially if there is an automated process that needs to perform the adjustments, or if the model is multi-dimensional. Algorithm RAFF was implemented to solve this drawback.

RAFF was also implemented in the Julia language and is the main method of the RAFF.jl package [8]. As already mentioned in Section 2, RAFF is easily parallelizable, so serial and parallel/distributed versions are available, through the Distributed.jl package. The algorithm (or the user) defines an interval

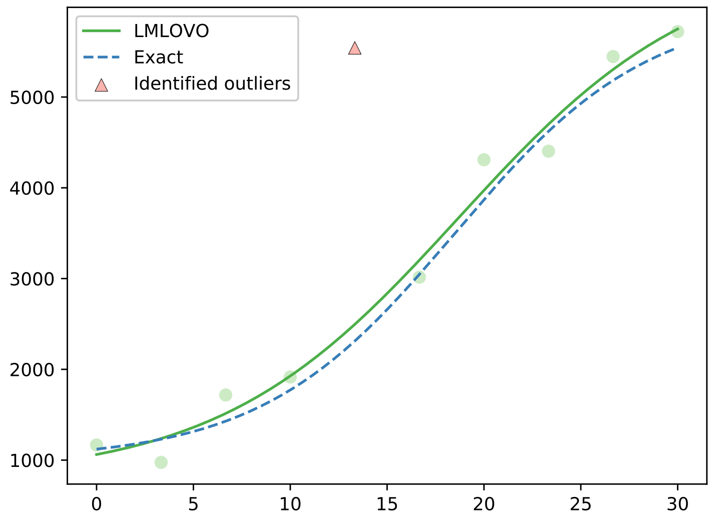


Figure 1: Test problem simulating an experiment following the logistic model. The continuous line represents the adjusted model, while the dashed line is the "exact" solution. `LM-LOVO` correctly identifies and ignores the outlier

of values of p to test and calls `LM-LOVO` to solve each subproblem for a given value of p. It is known that LOVO problems have many local minimizers, but we are strongly interested in global ones. Therefore, the traditional multistart technique is applied to generate random starting points. The larger the number of different starting points, the greater is the chance to find global minimizers. Also, the computational cost is increased. The parallel/distributed version of `RAFF` solves this drawback, distributing problems with different values of p among different cores, processors or even computers.

For the computation of the similarity between solutions $\bar{x}_p$ and $\bar{x}_q$ in Step 4, the Euclidean norm of the vector of differences was used

$$M_{pq} = \|\bar{x}_p - \bar{x}_q\|_2.$$

For each p in the interval, the best solution $\bar{x}_p$ obtained among all the runs of `LM-LOVO` for that p is stored. In order to avoid considering points in the cases where `LM-LOVO` has not converged for some value p, we set $M_{ip} = M_{pi} = \infty$ for all $i = p_{min}, \dots, p_{max}$ in case of failure.

In the preprocessing phase (Step 3 of `RAFF`) solutions $\bar{x}_q$ that clearly are not minimizers are also eliminated by setting $M_{iq} = M_{qi} = \infty$ for all $i = p_{min}, \dots, p_{max}$. To detect such points, we check if $S_q(\bar{x}_q) > S_p(\bar{x}_p)$ for some $q < p \leq p_{max}$. The idea is that the less points are considered in the adjustment, the smaller the residual should be at the global minimizer. The preprocessing phase also tries to eliminate solution $\bar{x}_{p_{max}}$. To do that, the valid solution $\bar{x}_p$ with smallest value of $S_p(\bar{x}_p)$, which was not eliminated by the previous strategy, is chosen, where $p < p_{max}$. Solution $\bar{x}_{p_{max}}$ is eliminated if $S_p(\bar{x}_p) < S_{p_{max}}(\bar{x}_{p_{max}})$

and the number of observed points (t_i, y_i) such that $|y_i - \phi(\bar{x}_p, t_i)| < |y_i - \phi(\bar{x}_{p_{max}}, t_i)|$, for $i = 1, \dots, r$, is greater or equal than $r/2$.

The last implementation detail of RAFF that needs to be addressed is the choice of ϵ. Although this value can be provided by the user, we found very hard to select a number that resulted in a correct adjustment. Very small or very large values of ϵ, result in the selection of $\bar{x}_{p_{max}}$ as the solution, since each solution will be similar to itself or similar to every solution, and we always select the largest p in such cases. To solve this issue, the following calculation has been observed to work well in practice

$$\epsilon = \min(M) + \operatorname{avg}(M)/(1 + p_{max}^{1/2}), \tag{26}$$

where M is the similarity matrix and function avg computes the average similarity by considering only the lower triangular part of M and ignoring ∞ values (which represent eliminated solutions). If there is no convergence for any value of $p \in [p_{min}, p_{max}]$, then $\bar{x}_{p_{max}}$ is returned, regardless if it has successfully converged or not.

4.1 Experiments for outlier detection and robust fitting

In the first set of tests, we verified the ability and efficiency of RAFF to detect outliers for well known statistical and mathematical models:

- Linear model: $\phi(x, t) = x_1 t + x_2$
- Cubic model: $\phi(x, t) = x_1 t^3 + x_2 t^2 + x_3 t + x_4$
- Exponential model: $\phi(x, t) = x_1 + x_2 \exp(-x_3 t)$
- Logistic model: $\phi(x, t) = x_1 + \frac{x_2}{1+\exp(-x_3 t + x_4)}$

The large number of parameters to be adjusted increases the difficulty of the problem, since the number of local minima also increases. For these tests, we followed some ideas described in [20]. For each model, we created 1000 random generated problems having: 10 points and 1 outlier, 10 points and 2 outliers, 100 points and 1 outlier, and 100 points and 10 outliers. For each combination, we also tested the effect of the multistart strategy using: 1, 10, 100 and 1000 random starting points.

The procedure for generating each random instance is described as follows. It is also part of the RAFF.jl package [8]. Let x^* be the exact solution for this fitting problem. First, r uniformly spaced values for t_i are selected in the interval $[1, 30]$. Then, a set $O \subset \{1, \dots, r\}$ with $r-p$ elements values is randomly selected to be the set of outliers. For all $i = 1, \dots, r$ a perturbed value is computed, simulating the results from an experiment. Therefore, we set $y_i = \phi(x^*, t_i) + \xi_i$, where $\xi_i \sim \mathcal{N}(0, 200)$, if $i \notin O$ and, otherwise, $y_i = \phi(x^*, t_i) + 7s\xi_i'\xi_i$, where $\xi_i \sim \mathcal{N}(0, 200)$, ξ_i' a uniform random number between 1 and 2 and $s \in \{-1, 1\}$ is randomly selected at the beginning of this process (so all outliers are in the "same side" of the curve). The exact solutions used to generate the instances are given in Table 1. The example illustrated in Figure 1 was also generated by this procedure.

The parallel version of RAFF was run with its default parameters on a Intel Xeon E3-1220 v3 3.10GHz with 4 cores and 16GB of RAM and Linux LUbuntu

Model	x^*
Linear	$(-200, 1000)$
Cubic	$(0.5, -20, 300, 1000)$
Exponential	$(5000, 4000, 0.2)$
Logistic	$(6000, -5000, -0.2, -3.7)$

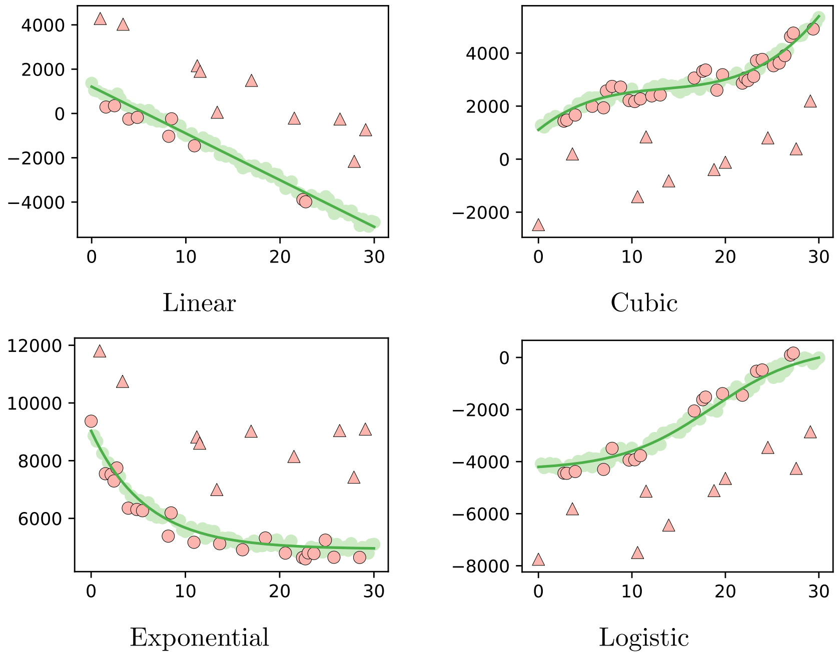
Table 1: Exact solutions used for each model in order to generate random instances

18.04 operating system. The obtained results are displayed in Tables 2 and 3. In those tables, r is the number of points representing the experiments, p is the number of trusted points, `FR` is the ratio of problems in which all the outliers have been found (but other points may be declared as outliers), `ER` is the ratio of problems where exactly the $r - p$ outliers have been found, `TP` is the average number of correctly identified outliers, `FP` is the average number of incorrectly identified outliers, `Avg.` is the average number of points that have been declared as outliers by the algorithm and `Time` is the total CPU time in seconds to run all the 1000 tests, measured with the `@elapsed` Julia macro. By default, $p_{min} = 0.5r$ and $p_{max} = r$ are set in the algorithm. The success criteria (25) of `LM-LOVO` was set to $\varepsilon = 10^{-4}$, while $\overline{\lambda}$ was set to 2. For each combination (Model, r, p) there are 4 rows in Tables 2 and 3, representing different numbers of multistart trials: 1, 10, 100 and 1000.

Some conclusions can be drawn from Tables 2 and 3. We can see that `RAFF` attains its best performance for outlier detection when the number of correct points is not small, even though the percentage of outliers is high. For the exponential and logistic models, we also can see clearly the effect of the multistart strategy in increasing the ratio of identified outliers. In problems with 100 experiments, we observe that in almost all the cases the number of outliers have been overestimated in average: although the ratio of outlier identification is high (`FR`), the ratio of runs where only the exact outliers have been detected (`TR`) is very low, being below 20% of the runs. For small test sets, this ratio increases up to 50%, but difficult models, such as the exponential and logistic, have very low ratios. However, as we can observe in Figure 2, the shape and the parameters of the model are clearly free from the influence of outliers. This observation suggests that maybe the perturbation added to all the values is causing the algorithm to detect correct points as outliers. The effect of the number of multi-start runs linearly increases the runtime of the algorithm, but is able to improve the adjustment, specially for the logistic model. The exponential model has an awkward behavior, where the `ER` ratio decreases when the number of multi-start runs increases, although the ratio of problems where all the outliers have been detected increases (`FR`). This might indicate that the tolerance (26) could be improved. We can also observe that the runtime of the exponential model is ten times higher than the other models.

When the size of the problem is multiplied by 10 (from 10 points to 100), we observe that the CPU time is multiplied by 5. This occurs because the time used by communication in the parallel runs is less important for larger datasets. Again, the exponential model is an exception.

In a second round of experiments, the same procedure was used to generate



Linear

Cubic

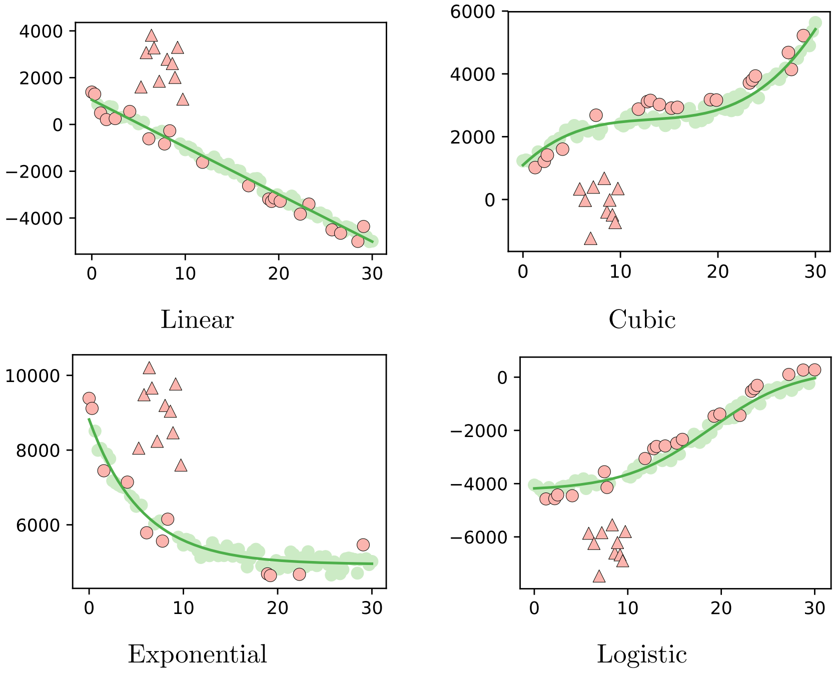
Exponential

Logistic

Figure 2: Selected instances of test problems with $r = 100$ and $p = 90$ and the solutions obtained by `RAFF`. All the outliers have been correctly identified in those cases (dark/red triangles). Non-outliers are described by circles, where the dark/red ones represent points incorrectly classified as outliers by the algorithm

random test problems simulating results from 100 experiments ($r = 100$), where a cluster of 10% of the points are outliers ($p = 90$). The default interval used for the values of t is $[1, 30]$, and the clustered outliers always belong to $[5, 10]$. Selected instances for each type of model are shown in Figure 3 as well as the solution found by `RAFF`. Again, 1000 random problems were generated for each type of model and the multi-start procedure was fixed to 100 starting points for each problem. The obtained results are shown in Table 4. A clustered set of outliers can strongly affect the model but is also easier to detect, when the number of outliers is not very large. As we can observe in Table 4, the ratio of instances where all the outliers have been successfully detected has increased in all models. The logistic model is the most difficult to fit since, on average, `RAFF` detects 17 points as outliers and 9 of them are correctly classified (`TP`). All the other models are able to correctly identify 10 outliers, on average, and have a higher `FR` ratio.

This set of experiments also shows another benefit of the present approach. If the user roughly knows the number of points that belong to a given model, such information can be used in the elimination of random (not necessary Gaussian) noise. The clustered example will be also shown to be an advantage over traditional robust least-squares algorithms in Subsections 4.2 and 4.3.



Linear

Cubic

Exponential

Logistic

Figure 3: Selected instances of test problems containing a clustered set of outliers and the solutions obtained by RAFF. All the outliers have been correctly identified in those cases (dark/red triangles). Non-outliers are described by circles, where the dark/red ones represent points incorrectly classified as outliers by the algorithm

4.2 Comparison against robust algorithms

We compared the fitting obtained by RAFF against classical and robust fitting algorithms provided by the SciPy library version 1.3.1 in Python[1]. The robust fitting algorithm in SciPy consists of using different loss functions in the least squares formulation. The following loss functions were used: linear (usual least squares formulation), soft_l1 (smooth approximation of the ℓ_1 loss function), huber and cauchy. The PyCall.jl Julia library was used to load and call SciPy.

Two more algorithms based on the RANSAC (Random Sample Consensus) [11], implemented in C++ from the Theia Vision Library[2] version 0.8, were considered. The first one, called here RANSAC, is the traditional version of RANSAC and the second one is LMED, based on the work [22], which does not need the error threshold, the opposite of case of RANSAC (where the threshold is problem dependent).

All the algorithms from SciPy were run with their default parameters. The best model among 100 runs was selected as the solution for each algorithm and the starting point used was randomly generated following the normal distribution with $\mu = 0$ and $\sigma = 1$. Algorithms RANSAC and LMED were run only 10

[1] https://docs.scipy.org/doc/scipy/reference/optimize.html

[2] http://www.theia-sfm.org/ransac.html

times, due to the higher CPU time used and the good quality of the solution achieved. RANSAC and LMED were run with a maximum of 1000 iterations, sampling 10% of the data and with the MLE score parameter activated. In order to adjust models to the sampled data, the Ceres least squares solver[3] version 1.13.0 was used, since Theia has a natural interface to it. All the scripts used in the tests are available at https://github.com/fsobral/RAFF.jl. Once again, the parallel version of RAFF was used. The test problems were generated by the same procedures discussed in Subsection 4.1. However, only one problem (instead of 1000) for each configuration (Model, r, p) was used.

Unlike RAFF, traditional fitting algorithms do not return the possible outliers of a dataset. Robust algorithms such as least squares using ℓ_1 or Huber loss functions are able to ignore the effect of outliers, but not to easily detect them. Therefore, for the tests we selected one instance of each test of type (model, r, p), where the models and values for r and p are the same used in Tables 2–4. The results are displayed in Table 6. For each problem p and each algorithm a, we measured the adjustment error $A_{a,p}$ between the model obtained by the algorithm $\phi_p(x^\star_{a,p}, t)$ and the points that are non-outliers, which is given by

$$A_{a,p} = \sqrt{\sum_{\substack{i \in \mathcal{P} \\ i \text{ non-outlier}}} (\phi_p(x^\star_{a,p}, t_i) - y_i)^2},$$

where ϕ_p was the model used to adjust problem p. Each row of Table 6 represents one problem and contains the relative adjustment error for each algorithm, which is defined by

$$\bar{A}_{a,p} = \frac{A_{a,p}}{\min_i\{A_{i,p}\}} \tag{27}$$

and the time taken to find the model (in parenthesis). The last row contains the number of times that each algorithm has found a solution with adjustment error smaller than 1% of best, smaller than 10% of the best and smaller than 20% of the best adjustment measure found for that algorithm, respectively, in all the test set. We can observe that RAFF, LMED and soft_l1 were the best algorithms. RAFF was the solver that found the best models in most of the problems (11/24), followed by LMED (9/24). Its parallel version was consistently the fastest solver among all. It is important to observe that RAFF was the only who was easily adapted to run in parallel. However, the parallelism is related only to the solution of subproblems for different p, not to the multistart runs, which are run sequentially. Therefore, RAFF solves considerably more problems per core than the other algorithms in a very competitive CPU time. When parallelism is turned of, the CPU time is very similar to the traditional least squares algorithm (linear). Also, RAFF was the only one that easily outputs the list of possible outliers without the need of any threshold parameter. Clustered instance (cubic, 100, 90) and instance (logistic, 10, 9) and the models obtained by each algorithm are shown in Figure 4.

4.3 Experiments for circle detection

The problem of detecting patterns in images is very discussed in the vision area in Computer Science. LOVO algorithms have also been applied to solv-

[3] http://ceres-solver.org/

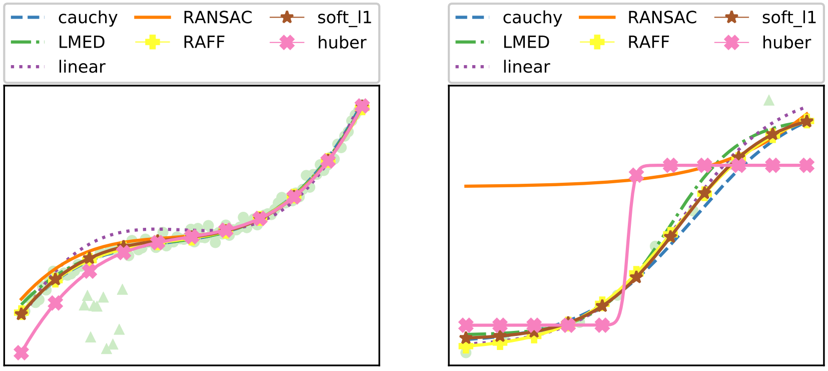


Figure 4: Two problems and the models found for each algorithm. On the left, a cubic model with 100 points and a set of 10 clustered outliers. On the right, a logistic model with 10 points and only one outlier

ing such problems, as a nonlinear programming alternative to traditional techniques [1]. The drawback, again, is the necessity of providing a reasonable number of trusted points. RAFF allows the user to provide an interval of possible trusted points, so the algorithm can efficiently save computational effort when trying to find patterns in images. Since LOVO problems need a model to be provided, circle detection is a perfect application to the algorithm.

We followed tests similar to [26], using a circular model

$$\phi(x, t) = (t_1 - x_1)^2 + (t_2 - x_2)^2 - x_3^2$$

instead of the ellipse model considered in the work. Two test sets were generated. In the first set $r = 100$ points were uniformly distributed in the border of the circle with center $(-10, 30)$ and radius 2. If the point is not an outlier, a random perturbation $\xi \sim \mathcal{N}(0, 0.1)$ is added to each one of its t_1 and t_2 coordinates. For outliers, the random noise is given by $\xi \sim \mathcal{N}(0, 2)$, as suggested in [26]. In the second set, $r = 300$ was considered. The same circumference was used and p points (non-outliers) were uniformly distributed in the circumference and slightly perturbed with a noise $\xi \sim \mathcal{N}(0, 0.1)$ as before. The remaining $300 - p$ points (the outliers) were randomly distributed in a square whose side was 4 times the radius of the circle, using the uniform distribution. Nine problems were generated in each test set, with outlier ratio ranging from 10% up to 90% (i. e. ratio of non-outliers decreasing 90% to 10%).

The same algorithms were compared, the only difference from Subsection 4.2 is that the error threshold of RANSAC was reduced to 10 and 100 random starting points near $(1, 1, 1)$ were used for all algorithms, except RANSAC and LMED. For those two algorithms, we kept the number of trials to 10. Also, we tested two versions of RAFF. In pure RAFF, we decreased the lower bound p_{min} from its default value $0.5r$ to the value of p, when p falls below the default. In RAFFint, we used the option of providing upper and lower bounds for the number of trusted points. If p is the current number of non-outliers in the instance, the interval given to RAFFint is $[p - 0.3r, p + 0.3r] \cap [0, r]$. The measure (27) was used and the results are shown in Figure 5.

We can observe that RAFF, LMED and cauchy achieved the best results. RAFF found worse models than most of the robust algorithms in the problems of the first test set, although the results are still very close. Its relative performance increases as the outlier ratio increases. This can be explained as the strong attraction that RAFF has to finding a solution similar to traditional least squares algorithms. In Figure 6 we can see that RAFF has difficulty in finding outliers that belong to the interior of the circle. To solve this drawback, RAFF also accepts a lower bound in the number of outliers, rather than only an upper bound. This ability is useful for large datasets with a lot of noise, as is the case of the second test set, and allows the detection of inner outliers. This is represented by RAFFint. We can see in Figure 5 that the performance of both versions of RAFF is better than traditional robust algorithms in the case of a large number of outliers.

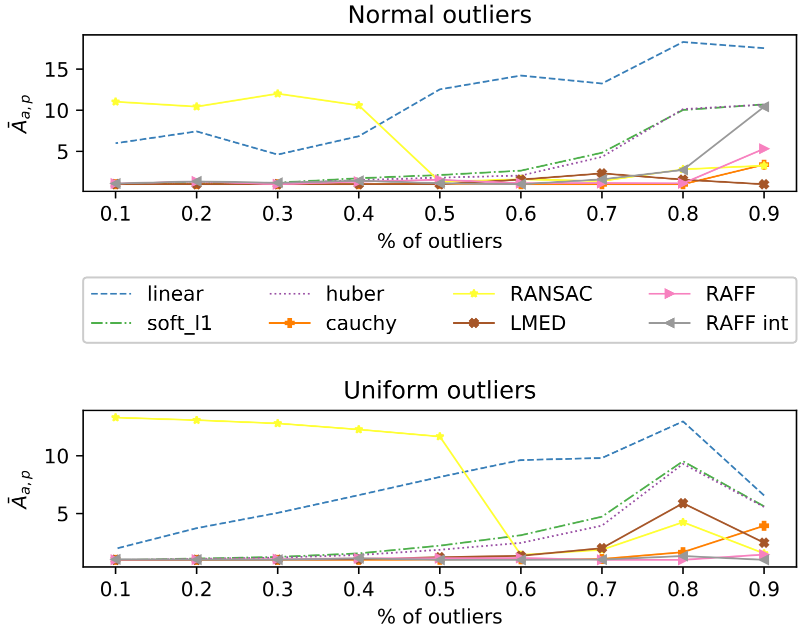
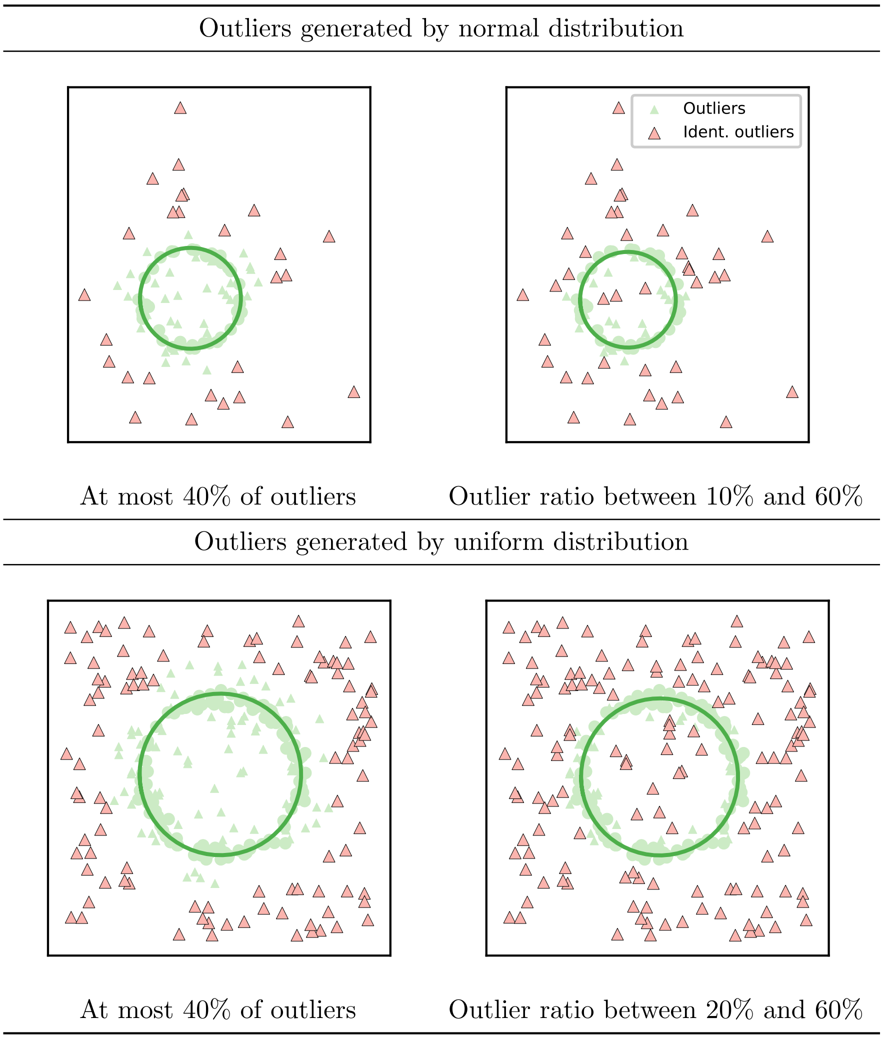


Figure 5: Relative adjustment error in the circle detection problem for increasing outlier ratio and two different types of perturbation: by normal and uniform distributions

4.4 Comparison against another LOVO approach for model adjustment

We also compared RAFF against the LOVO Gauss-Newton line-search algorithm described in [1]. The authors were unable to obtain the codes used in the paper, thus a Julia implementation was coded following the description of the algorithm. The algorithm is also available in the RAFF.jl package as function gnlslovo.

As previously mentioned, the algorithm in [1] needs a reasonable choice of parameter p, the number of trusted points. A direct comparison between the



Outliers generated by normal distribution

At most 40% of outliers

Outlier ratio between 10% and 60%

Outliers generated by uniform distribution

At most 40% of outliers

Outlier ratio between 20% and 60%

Figure 6: Difference of outlier detection when upper bounds on the outlier ratio are provided. The inner outliers are harder to detect, since the error they cause in the model is smaller

LOVO Gauss-Newton algorithm and Algorithm 1 is not very useful, since both algorithms are well known to solve nonlinear least-squares problems. Gauss-Newton algorithms are usually faster and more accurate than the Levenberg-Marquardt approach, but they might suffer when the matrix used to compute the descent direction is ill conditioned. Therefore, in this subsection, our aim is to show that bad choices of p in the LOVO Gauss-Newton algorithm [1] lead to poorer adjustments and are usually avoided when the voting system used by `RAFF` is applied with an inexact interval estimate around p.

The same set of clustered problems from Subsection 4.2 was used, but only for $r = 100$ and $p = 90$. Also, we generated a circle detection problem with random noise ($r = 300$ and $p = 100$), in the same fashion as Subsection 4.3. To compare the two approaches, the adjustment error (27) was used. For each model, different values of p were given to the LOVO Gauss-Newton algorithm and an interval around each p was given to RAFF. The interval was defined as $[\max\{0, p/r - 0.3\}r, \min\{1, p/r + 0.3\}r]$. The Gauss-Newton algorithm was allowed to use 100 random initial points while RAFF was allowed to use 30 random initial points for each p in the interval. The values of p used and the adjustment error obtained are displayed in Table 5.

We can clearly see the benefits of using the voting system. Even if poor values of p are provided, RAFF is usually able to find better adjustments than the Gauss-Newton algorithm with fixed p. The voting system provides a way to compare the solution for different values of p, which is not a trivial task. We also observe that LOVO Gauss-Newton is able to find reasonable solutions using values of p not too close to the true ones. For the logistic model, the results of Gauss-Newton are worse, since the matrix that appears in the problems is very ill conditioned. The LOVO Gauss-Newton algorithm could also be used inside the voting system of RAFF, replacing the LOVO Levenberg-Marquardt algorithm, but the results would be very similar.

5 Conclusions

In this paper, we have described a LOVO version of the Levenberg-Marquardt algorithm for solving nonlinear equations, which is specialized to the adjustment of models where the data contains outliers. The theoretical properties of the algorithm were studied and convergence to strongly and weakly stationary points has been proved. To overcome the necessity of providing the number of outliers in the algorithm, a voting system has been proposed. A complete framework to robust adjustment of data was implemented in the Julia language and compared to public available and well tested robust fitting algorithms. The proposed algorithm was shown to be competitive, being able to find better adjusted models in the presence of outliers in most of the problems. In the circle detection problem, the proposed algorithm was also shown to be competitive and had a good performance even when the outlier ration exceeds 50%. By comparing against a LOVO Gauss-Newton algorithm for model adjustment, the voting system was shown to be a good strategy when the number of outliers cannot be estimated beforehand. The implemented algorithm and all the scripts used for testing and generation of the tests are freely available and constantly updated at `https://github.com/fsobral/RAFF.jl`.

Data availability statement

The data that support the findings of this study can be generated by the scripts provided in `https://github.com/fsobral/RAFF.jl`, but can also be requested from the corresponding author upon request.

References

[1] R. Andreani, G. Cesar, R. Cesar-Jr., J. M. Martínez, and P. J. S. Silva. Efficient curve detection using a Gauss-Newton method with applications in

agriculture. In *Proc. 1st International Workshop on Computer Vision Applications for Developing Regions in Conjunction with ICCV 2007-CVDR-ICCV07*, 2007.

[2] R. Andreani, C. Dunder, and J. M. Martínez. Order-Value Optimization: Formulation and solution by means of a primal cauchy method. *Mathematical Methods of Operations Research (ZOR)*, 58(3):387–399, 2003.

[3] R. Andreani, C. Dunder, and J. M. Martínez. Nonlinear-programming reformulation of the order-value optimization problem. *Mathematical Methods of Operations Research*, 61(3):365–384, 2005.

[4] R. Andreani, J. M. Martínez, L. Martínez, and F. S. Yano. Continuous optimization methods for structure alignments. *Mathematical Programming*, 112(1):93–124, 2008.

[5] R. Andreani, J. M. Martínez, L. Martínez, and F. S. Yano. Low Order-Value Optimization and applications. *Journal of Global Optimization*, 43(1):1–22, 2009.

[6] E. H. Bergou, Y. Diouane, and V. Kungurtsev. Convergence and Complexity Analysis of a Levenberg–Marquardt Algorithm for Inverse Problems. *Journal of Optimization Theory and Applications*, 185(3):927–944, 2020.

[7] E. G. Birgin, L. F. Bueno, N. Krejić, and J. M. Martínez. Low Order-Value approach for solving VaR-constrained optimization problems. *Journal of Global Optimization*, 51(4):715–742, 2011.

[8] E. V. Castelani, R. Lopes, W. Shirabayashi, and F. N. C. Sobral. RAFF.jl: Robust Algebraic Fitting Function in Julia. *Journal of Open Source Software*, 4(39):1385, 2019.

[9] R. D. Cook. Detection of influential observations in linear regression. *Technometrics*, 19:15–18, 1977.

[10] R. O. Duda and P. E. Hart. Use of the Hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1971.

[11] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[12] F. R. Hampel, E. Ronchetti, P. Rousseeuw, and W. Stahel. *Robust Statistics: The Approach Based on Influence Functions.* John Wiley & Sons, New York, NY, USA, 1986.

[13] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[14] P. V. Hough. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654.

[15] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[16] Z. Jiang, Q. Hu, and X. Zheng. Optimality condition and complexity of Order-Value Optimization problems and Low Order-Value Optimization problems. *Journal of Global Optimization*, 69(2):511–523, 2017.

[17] J. M. Martínez. Generalized order-value optimization. *TOP*, 20(1):75–98, 2012.

[18] L. Martínez, R. Andreani, and J. M. Martínez. Convergent algorithms for protein structural alignment. *BMC Bioinformatics*, 8(1):306, 2007.

[19] J. J. Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer Berlin Heidelberg, 1978.

[20] H. J. Motulsky and E. R. Brown. Detecting outliers when fitting data with nonlinear regression – a new method based on robust nonlinear regression and the false discovery rate. *BMC Bioinformatics*, 7(123), 2006.

[21] J. Nocedal and S. J. Wright. *Numerical Optimization.* Springer, New York, 2nd edition, 2006.

[22] P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79(388):871–880, 1984.

[23] R. E. Shiffler. Maximum Z scores and outiliers. *The American Statistics*, 42(1):79–80, 1988.

[24] S. S. Sreevidya. A survey on outlier detection methods. *International Journal of Computer Science and Information Technologies*, 5(6):8153–8156, 2014.

[25] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: randomized Hough transform (RHT). *Pattern recognition letters*, 11(5):331–338, 1990.

[26] J. Yu, H. Zheng, S. R. Kulkarni, and H. Poor. Two-stage outlier elimination for robust curve and surface fitting. *EURASIP Journal on Advances in Signal Processing*, 2010(1), 2010.

Type	r	p	FR	ER	TP	FP	Avg.	Time (s)
Linear	10	9	0.858	0.552	0.858	0.349	1.21	2.167
			0.859	0.554	0.859	0.347	1.21	3.511
			0.859	0.554	0.859	0.347	1.21	12.893
			0.859	0.554	0.859	0.347	1.21	89.285
		8	0.467	0.418	1.112	0.144	1.26	2.344
			0.467	0.417	1.112	0.145	1.26	3.596
			0.467	0.417	1.112	0.145	1.26	13.164
			0.467	0.417	1.112	0.145	1.26	88.684
	100	99	0.983	0.078	0.983	10.656	11.64	10.297
			0.982	0.074	0.982	10.655	11.64	42.091
			0.982	0.074	0.982	10.677	11.66	316.604
			0.982	0.075	0.982	10.682	11.66	3082.385
		90	0.916	0.069	9.858	6.768	16.63	9.799
			0.916	0.070	9.858	6.798	16.66	40.581
			0.915	0.070	9.854	6.782	16.64	317.722
			0.917	0.070	9.860	6.799	16.66	3099.536
Cubic	10	9	0.767	0.572	0.767	0.290	1.06	3.062
			0.810	0.563	0.810	0.371	1.18	4.111
			0.886	0.549	0.886	0.461	1.35	16.370
			0.886	0.545	0.886	0.465	1.35	126.554
		8	0.150	0.122	0.581	0.243	0.82	2.353
			0.333	0.282	0.894	0.202	1.10	4.221
			0.525	0.462	1.220	0.143	1.36	16.482
			0.533	0.469	1.232	0.142	1.37	126.088
	100	99	0.990	0.046	0.990	10.997	11.99	11.485
			0.991	0.041	0.991	11.351	12.34	51.548
			0.992	0.037	0.992	11.788	12.78	420.033
			0.993	0.036	0.993	11.706	12.70	4123.040
		90	0.945	0.064	9.838	6.941	16.78	11.325
			0.930	0.063	9.816	7.299	17.11	50.685
			0.941	0.063	9.835	7.584	17.42	414.084
			0.940	0.060	9.833	7.714	17.55	4042.454

Table 2: Results of RAFF for the detection of outliers for linear and cubic models. For each kind of problem, a multistart strategy was tested with 1, 10, 100 and 1000 random starting points

Type	r	p	FR	ER	TP	FP	Avg.	Time (s)
Exponential	10	9	0.549	0.141	0.549	0.751	1.30	5.627
			0.698	0.463	0.698	0.354	1.05	17.289
			0.777	0.535	0.777	0.338	1.11	136.491
			0.822	0.581	0.822	0.306	1.13	1215.738
		8	0.213	0.080	0.771	0.459	1.23	4.417
			0.292	0.264	0.921	0.152	1.07	18.053
			0.406	0.367	1.138	0.148	1.29	138.862
			0.516	0.480	1.246	0.092	1.34	1245.882
	100	99	0.982	0.089	0.982	5.444	6.43	47.235
			0.992	0.046	0.992	10.673	11.66	392.884
			0.992	0.047	0.992	10.794	11.79	3521.630
			0.993	0.044	0.993	11.298	12.29	35418.130
		90	0.532	0.133	8.234	1.921	10.15	47.915
			0.972	0.060	9.946	7.181	17.13	384.544
			0.980	0.054	9.959	7.611	17.57	3389.777
			0.980	0.063	9.939	7.772	17.71	34121.028
Logistic	10	9	0.009	0.001	0.009	0.116	0.13	2.705
			0.245	0.156	0.245	0.419	0.66	3.714
			0.420	0.309	0.420	0.292	0.71	18.144
			0.524	0.364	0.524	0.279	0.80	150.310
		8	0.003	0.001	0.091	0.400	0.49	1.914
			0.032	0.028	0.396	0.369	0.77	3.932
			0.065	0.059	0.389	0.285	0.67	21.091
			0.167	0.143	0.559	0.203	0.76	175.915
	100	99	0.535	0.006	0.535	7.105	7.64	9.426
			0.536	0.012	0.536	11.754	12.29	34.022
			0.894	0.063	0.894	2.529	3.42	309.246
			0.929	0.095	0.929	5.276	6.21	2678.522
		90	0.002	0.000	4.345	3.295	7.64	9.459
			0.008	0.001	4.599	5.502	10.10	38.605
			0.432	0.001	6.551	4.629	11.18	319.713
			0.430	0.099	8.084	2.001	10.09	2774.727

Table 3: Results for RAFF for the detection of outliers for exponential and logistic models. For each kind of problem, a multistart strategy was tested with 1, 10, 100 and 1000 random starting points

Type	FR	ER	TP	FP	Avg.	Time (s)
Linear	0.949	0.104	9.936	6.363	16.30	323.623
Cubic	0.991	0.047	9.991	8.368	18.36	423.634
Exponential	0.987	0.097	9.983	6.775	16.76	3445.482
Logistic	0.745	0.007	8.778	8.382	17.16	326.675

Table 4: Numerical results for problems with $p = 100$ data points and 10% of clustered outliers

		20	60	70	95	99
linear	`RAFF`	9.3e+02	9.5e+02	9.5e+02	9.2e+02	9.5e+02
	`GN`	3.2e+04	9.7e+02	9.7e+02	1.1e+03	1.6e+03
cubic	`RAFF`	1.0e+03	9.2e+02	9.7e+02	9.2e+02	9.2e+02
	`GN`	1.1e+03	9.6e+02	9.7e+02	1.2e+03	1.8e+03
expon	`RAFF`	3.9e+03	9.7e+02	9.7e+02	9.9e+02	9.4e+02
	`GN`	9.1e+03	8.9e+03	4.4e+114	7.9e+03	8.0e+03
logistic	`RAFF`	1.1e+03	9.7e+02	9.4e+02	1.4e+04	9.5e+02
	`GN`	3.4e+04	3.4e+04	1.6e+04	3.4e+04	1.4e+04
		15	30	120	150	180
circle	`RAFF`	3.7e+00	4.1e+00	7.6e+00	9.1e+00	1.3e+02
	`GN`	1.7e+02	4.5e+01	4.1e+00	1.6e+01	4.1e+01

Table 5: Adjustment error for the LOVO Gauss-Newton algorithm [1] (`GN`) with fixed p and `RAFF` using the voting system with an interval around p.

(Model, r, p)	`linear`	`soft_l1`	`huber`	`cauchy`	`RANSAC`	`LMED`	`RAFF`
linear, 10, 9	1.35 (0.60)	1.06 (2.75)	1.06 (3.11)	1.37 (1.64)	4.28 (0.52)	1.00 (0.53)	1.35 (1.48)
linear, 10, 8	3.62 (0.60)	1.18 (2.55)	1.18 (2.89)	1.14 (2.07)	1.17 (0.54)	1.01 (0.55)	1.00 (0.12)
linear, 100, 99	1.03 (0.60)	1.00 (3.22)	1.00 (8.43)	1.00 (2.30)	1.22 (3.95)	1.00 (3.98)	1.00 (0.82)
linear, 100, 90	1.75 (0.60)	1.00 (3.41)	1.00 (8.41)	1.04 (1.65)	1.23 (4.71)	1.01 (47.23)	1.04 (0.67)
cubic, 10, 9	1.27 (0.81)	1.00 (9.44)	1.11 (11.42)	4.71 (4.67)	22.26 (0.98)	2.47 (0.99)	4.91 (1.04)
cubic, 10, 8	4.06 (0.73)	1.18 (14.80)	1.18 (16.28)	1.19 (4.75)	114.88 (1.00)	1.21 (1.00)	1.00 (0.18)
cubic, 100, 99	1.05 (0.72)	1.00 (21.08)	2.24 (22.20)	1.09 (6.00)	1.12 (8.00)	1.04 (8.04)	1.08 (0.81)
cubic, 100, 90	1.76 (0.73)	1.00 (21.59)	1.78 (22.86)	1.19 (5.79)	1.15 (7.94)	1.00 (79.15)	1.00 (0.75)
expon, 10, 9	1.16 (5.04)	12.88 (12.26)	12.88 (12.59)	4.68 (10.70)	14.22 (0.76)	1.00 (0.76)	1.16 (1.35)
expon, 10, 8	1.34 (3.28)	7.74 (12.14)	10.07 (12.45)	1.00 (10.89)	1.72 (0.72)	1.61 (0.73)	1.34 (0.25)
expon, 100, 99	1.00 (3.86)	8.57 (13.61)	8.58 (12.86)	1.22 (10.99)	2.04 (5.32)	1.05 (5.32)	1.03 (5.19)
expon, 100, 90	1.76 (3.65)	8.99 (13.53)	8.99 (13.18)	3.70 (11.04)	1.93 (5.35)	1.00 (53.63)	1.02 (5.32)
logistic, 10, 9	1.68 (3.24)	1.31 (18.98)	7.60 (20.55)	1.91 (17.28)	24.45 (0.89)	2.27 (0.90)	1.00 (0.99)
logistic, 10, 8	4.23 (10.27)	1.00 (18.14)	9.40 (19.18)	26.13 (3.98)	39.56 (0.85)	1.05 (0.87)	22.19 (0.14)
logistic, 100, 99	1.01 (4.44)	2.34 (17.68)	8.88 (18.84)	2.64 (8.24)	1.02 (7.75)	1.00 (7.76)	1.01 (0.51)
logistic, 100, 90	1.78 (2.57)	1.10 (18.16)	7.13 (19.10)	7.53 (8.06)	1.06 (7.59)	1.00 (76.37)	1.01 (0.53)
Clustered							
linear, 10, 8	4.10 (0.52)	1.18 (2.33)	1.18 (2.54)	1.00 (2.02)	4.69 (0.54)	1.07 (0.55)	1.00 (0.10)
linear, 100, 90	1.92 (0.62)	1.00 (2.76)	1.00 (5.70)	1.12 (2.10)	1.11 (4.66)	1.02 (46.45)	1.03 (0.68)
cubic, 10, 8	4.29 (0.72)	1.00 (11.90)	1.00 (14.59)	11.59 (4.87)	31.72 (1.04)	10.79 (1.02)	12.61 (0.14)
cubic, 100, 90	2.18 (0.73)	1.02 (20.39)	2.47 (22.25)	1.07 (6.31)	1.52 (7.70)	1.09 (73.61)	1.00 (0.83)
expon, 10, 8	1.32 (12.79)	4.11 (10.54)	1.81 (10.34)	1.20 (9.18)	1.00 (0.75)	2.69 (0.78)	4.22 (0.29)
expon, 100, 90	1.99 (3.56)	8.58 (13.31)	8.57 (12.79)	4.99 (10.90)	1.74 (5.66)	1.14 (57.14)	1.00 (5.82)
logistic, 10, 8	3.79 (10.17)	1.00 (15.28)	7.74 (18.09)	22.46 (10.53)	16.32 (0.66)	1.40 (0.65)	12.31 (0.13)
logistic, 100, 90	2.00 (4.87)	4.62 (18.10)	6.77 (19.83)	7.35 (7.10)	1.40 (7.78)	1.03 (78.43)	1.00 (0.52)
	2, 5, 6	9, 11, 15	4, 5, 9	3, 6, 11	1, 3, 7	9, 16, 17	11, 16, 17

Table 6: Comparison against different robust fitting algorithms using one instance of each test problem and 100 random starting points as a multistart strategy. `RANSAC` and `LMED` run for 10 random initial points

A Convergence to strongly critical points

In order to achieve convergence to strongly critical points, it is necessary to modify Algorithm 1. Given $\delta > 0$, we start by defining the δ-relaxation of the set I_{min} at a point x, given by $I_{\delta-min}(x)$. This set is known in [5] as the set of δ-active indexes.

Definition A.1. Given $x \in \mathbb{R}^n$ we define the *δ-minimal function set of f_{min} in x* by

$$I_{\delta-min}(x) = \{i \in \{1, \ldots, q\} \mid f_i(x) \leq f_{min}(x) + \delta\}.$$

Using Definition A.1, we define a new model function to be minimized:

$$m_k(d) = \min_{i \in G_k} m_{k,i}(d), \tag{28}$$

where $m_{k,i}$ was defined by (8) and G_k is defined as

$$G_k = \{i \in I_{\delta-min}(x_k) \mid \|\nabla f_i(x_k)\|_2 \neq 0\}.$$

Note that, by the definition of $m_{k,i}$, it is not hard to minimize $m_k(d)$. The minimum will occur in a global minimizer of some $m_{k,i}$. Therefore, to calculate direction d_k we first define d_k^i, for $i \in G_k$, which is the solution of $(J_{\mathcal{C}_i}(x_k)^T J_{\mathcal{C}_i}(x_k)) + \gamma_k I)d = -\nabla f_i(x_k)$. Then, d_k is given by $d_k^{\hat{\imath}}$, where

$$\hat{\imath} = \arg\min_{i \in G_k}\{m_{k,i}(d_k^i)\}.$$

It is interesting to observe that, using this new definition of m_k, we have that $m_k(0) = \min_{i \in G_k} \frac{1}{2}\|F_{\mathcal{C}_i}(x_k)\|_2^2 = f_{min}(x_k)$. Also, for all $i \in G_k$, $m_{k,i}(0) \geq m_k(0)$ and $m_{k,\hat{\imath}}(d_k^{\hat{\imath}}) \leq m_{k,i}(d_k^i)$. All the steps are given by Algorithm 3.

It is not hard to observe that the Algorithm 3 is also well defined in the sense of Lemma 2.4. The key argument is that, if the **if** part in Step 4 is repeated an infinite number of times, then there is an index $\hat{\imath}_k \in G_k$ such that the global minimizer $d_k = d_k^{\hat{\imath}_k}$ will be chosen an infinite number of times, since $I_{\delta-min}(x_k)$ is fixed and finite. We, then, apply Lemma 2.4 with $i_k = \hat{\imath}_k$, observing that $\rho_{k,\hat{\imath}_k} = \rho_k$.

Now, we want to show that Algorithm 3 is able to generate sequences whose limit points are strongly critical, according to Definition 2.5. This is given by Theorem A.2.

Theorem A.2. *Let $\{x_k\}_{k \in \mathbb{N}}$ be a sequence generated by Algorithm 3 by choosing $\varepsilon = 0$. Consider $\mathcal{K}' \subset \mathbb{N}$ such that $\lim_{k \in \mathcal{K}'} x_k = x^*$ and suppose that Assumption 2.2 holds. Then, x^* is a strongly critical point of* (3).

Proof. Suppose by contradiction that x^* is not a strongly critical point of (3). Therefore, there exist an index $j \in I_{min}(x^*)$, $\beta \in \mathbb{R}_+$ and an infinite subset $\mathcal{K}_\beta \subset \mathcal{K}'$ such that $\|\nabla f_j(x_k)\|_2 \geq \beta$, for all $k \in \mathcal{K}_\beta$. Using the continuity of f_i, $i \in \{1, \ldots, q\}$, we have that $\lim_{k \in \mathcal{K}_\beta} f_i(x_k) = f_i(x^*)$, that is, there is $K_i > 0$ such that

$$|f_i(x_k) - f_i(x^*)| \leq \frac{\delta}{2}, \tag{29}$$

Algorithm 3: `LM-LOVO-SC` – Levenberg-Marquardt for the LOVO problem (Strong Critical version).

```
    Input: $x_0 \in \mathbb{R}^n$, $0 < \lambda_{min} \leq \lambda_0$, $\varepsilon \geq 0$, $\delta > 0$, $\overline{\lambda} > 1, \mu \in (0,1)$ and
           $p \in \mathbb{N}$
    Output: $x_k$
    Set $k \leftarrow 0$;
1   $\tau_k = \max\limits_{I_{min}(x_k)} \|\nabla f_i(x_k)\|_2^2$;
    $G_k = \{i \in I_{\delta-min}(x_k) \mid \|\nabla f_i(x_k)\|_2 \neq 0\}$;
    $\lambda \leftarrow \lambda_k$;
2   if $\tau_k \leq \varepsilon$ then
        Stop the algorithm, $x_k$ is a strongly critical solution for the LOVO
          problem;
3   $\gamma_k \leftarrow \lambda\tau_k$;
    Compute $d_k = d_k^{\hat{\imath}}$, where $\hat{\imath} = \arg\min_{i \in G_k}\{m_{k,i}(d_k^i)\}$;
    Calculate $\rho_k$ as
        $$\rho_k = \frac{f_{min}(x_k) - f_{min}(x_k + d_k)}{m_k(0) - m_k(d_k)}$$
4   if $\rho_k < \mu$ then
        $\lambda \leftarrow \overline{\lambda}\lambda$;
        Go back to the Step 3;
    else
        Go to the Step 5;
5   $\lambda_{k+1} \in [\max\{\lambda_{min}, \lambda/\overline{\lambda}\}, \lambda]$;
    $x_{k+1} \leftarrow x_k + d_k$;
    $k \leftarrow k + 1$ and go back to the Step 1 ;
```

for all $k \geq K_i, k \in \mathcal{K}_\beta$, where δ is a constant used by Algorithm 3. For each $k \geq K_\delta = \max\limits_{i=1,\ldots,q} K_i, k \in \mathcal{K}_\beta$, consider $i \in I_{min}(x^*)$ and some $\ell_k \in I_{min}(x_k)$. Equation (29), used twice, implies that

$$\begin{aligned} f_i(x_k) - f_{min}(x_k) = f_i(x_k) - f_{\ell_k}(x_k) &= f_i(x_k) - f_i(x^*) + f_i(x^*) - f_{\ell_k}(x_k) \\ &\leq \frac{\delta}{2} + f_i(x^*) - f_{\ell_k}(x_k) = \frac{\delta}{2} + f_{min}(x^*) - f_{\ell_k}(x_k) \\ &\leq \frac{\delta}{2} + f_{\ell_k}(x^*) - f_{\ell_k}(x_k) \leq \delta. \end{aligned}$$

We can conclude that, for all $k \geq K_\delta$, we have $I_{min}(x^*) \subset I_{\delta-\min}(x_k)$.

In particular, $j \in G_k$ for all $k \geq K_\delta, k \in \mathcal{K}_\beta$.

With similar arguments of Theorem 2.6, there exist $M \in \mathbb{R}_+$ and $K_M \in \mathcal{K}_\beta$ such that $\lambda_k \leq M$, for all $k \geq K_M, k \in \mathcal{K}_\beta$. Since the functions f_i, $i \in \{1, \ldots, q\}$, have continuous gradients there exist $L \in \mathbb{R}_+$ and $K_L \in \mathcal{K}_\beta$ such that $\tau_k = \max\limits_{i \in G_k}\{\|\nabla f_i(x_k)\|_2^2\} \leq L$, for all $k \geq K_L, k \in \mathcal{K}_\beta$.

Defining $c = \sup_{k \in \mathcal{K}'}\{\|J_{\mathcal{C}_j}(x_k)\|_2^2\} + LM$, by the continuity of f_{min} and f_j

there exists $K_m \in \mathcal{K}_\beta$ such that

$$|f_{min}(x^*) - f_{min}(x_k)| \leq \frac{\theta\beta^2}{8c} \quad \text{and} \quad |f_j(x_k) - f_{min}(x^*)| = |f_j(x_k) - f_j(x^*)| \leq \frac{\theta\beta^2}{8c}$$

for all $k \geq K_m, k \in \mathcal{K}_\beta$. Hence, we have that

$$\begin{aligned} m_{k,j}(0) - m_k(0) = f_j(x_k) - f_{min}(x_k) &= f_j(x_k) - f_{min}(x^*) + f_{min}(x^*) - f_{min}(x_k) \\ &\leq \frac{\theta\beta^2}{4c} \end{aligned} \tag{30}$$

for all $k \geq K_m, k \in \mathcal{K}_\beta$. Through expressions (28) and (30), we obtain, for each $k \geq K_m, k \in \mathcal{K}_\beta$, that

$$m_k(d_k) \leq m_{k,j}(d_k^j) \overset{(30)}{\Rightarrow} m_k(0) - m_k(d_k) \geq m_{k,j}(0) - m_{k,j}(d_k^j) - \frac{\theta\beta^2}{4c}. \tag{31}$$

Using (31) and similar arguments of (23), we obtain, for all $k \geq \max\{K_\delta, K_L, K_M, K_m\}$, $k \in \mathcal{K}_\beta$, that

$$\begin{aligned} &\frac{f_{min}(x_k) - f_{min}(x_{k+1})}{m_k(0) - m_k(d_k)} \geq \mu \\ &\Leftrightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \mu(m_k(0) - m_k(d_k)) \\ &\Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \overset{(31)}{\geq} \mu\left(m_{k,j}(0) - m_{k,j}(d_k^j) - \frac{\theta\beta^2}{4c}\right) \\ &\Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \overset{(12)}{\geq} \mu\left(\frac{\theta\|\nabla f_j(x_k)\|_2^2}{2(\|J_{\mathcal{C}_j}(x_k)\|_2^2 + \gamma_k)} - \frac{\theta\beta^2}{4c}\right) \\ &\Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \left(\frac{\mu\theta\beta^2}{2(\|J_{\mathcal{C}_j}(x_k)\|_2^2 + \lambda_k L)} - \frac{\mu\theta\beta^2}{4c}\right) \\ &\Rightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \left(\frac{\mu\theta\beta^2}{2(\sup_{k\in\mathcal{K}'}\{\|J_{\mathcal{C}_j}(x_k)\|_2^2\} + ML)} - \frac{\mu\theta\beta^2}{4c}\right) \\ &\Leftrightarrow f_{min}(x_k) - f_{min}(x_{k+1}) \geq \left(\frac{\mu\theta\beta^2}{2c} - \frac{\mu\theta\beta^2}{4c}\right) \\ &\Leftrightarrow f_{min}(x_{k+1}) - f_{min}(x_k) \leq -\left(\frac{\mu\theta\beta^2}{4c}\right), \end{aligned} \tag{32}$$

where the second implication follows from observing that $\lambda_k L \geq \lambda_k \tau_k = \gamma_k$ and $\|\nabla f_j(x_k)\|_2 \geq \beta$. Expression (32) and the property $f_{min}(x_{k+1}) \leq f_{min}(x_k)$, for all $k \in \mathcal{K}'$, contradict the hypothesis that f_{min} is bounded from below. Therefore, we conclude that there is no such $\mathcal{K}_\beta$ and

$$\lim_{k\in\mathcal{K}'} \|\nabla f_i(x_k)\|_2 = 0, \ \forall i \in I_{min}(x^*).$$

The continuity of the gradients ∇f_i, $i \in \{1, \ldots, q\}$, ensures that $\nabla f_i(x^*) = 0$ for all $i \in I_{min}(x^*)$ and, hence, x^* is a strongly critical point. □