RAFF.jl: Robust Algebraic Fitting Function in Julia

11 March 2019

Summary

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function whose mathematical description is not available. This function can be, for example, a black-box, a proprietary computer program or an experiment. Suppose that a data set $S = \{(x_1, f(x_1)), \ldots, (x_m, f(x_m))\}$ of m observations of f is available and we want to approximate f by a known model ϕ. Model ϕ can be defined as $\phi(x; \theta)$, where x are the n independent variables of f and θ represents some parameters of ϕ. `RAFF.jl` (Robust Algebraic Fitting Function) is a Julia package developed to find the optimal parameters θ for ϕ in order to adjust it to the observed values S of the unknown function f. Following (Liu and Wang 2008), in general, the adjust can be related to

1. Classical least square (algebraic fit): which consider the sum of deviations of type $|\phi(x_i; \theta) - f(x_i)|^2$;
2. Ortogonal least square (geometric fit): which consider the sum of deviations of type $\|\phi(x; \theta) - (x_i, f(x_i))\|^2$ (orthogonal projection on the curve to be adjusted).

`RAFF.jl` was developed to solve a generalization of the first case.

The adjustment of mathematical functions to data is a problem that appears in many areas of science. When data comes from real experiments, non-expected errors may cause the appearance of outliers. Detection of outliers is always regarded as the statistical part of data adjustment. `RAFF.jl` provides automatic detection of outliers using a voting system. It is an optimization-based package, based on algorithms for Lower Order-Value Optimization (LOVO) which were introduced in (Andreani, Dunder, and Martínez 2005) and revisited in (Andreani et al. 2009) to fit the user-provided models ϕ to experimental data. Recently, a complete review about LOVO problems considering theoretical aspects of algorithms to solve it and potential applications can be found in (Martínez 2012). In order to find a robust adjustment, a voting system is used, which is also responsible for the detection of possible outliers.

Background Material

To elucidate the essence of how `RAFF.jl` works, let us detail some aspects related to the LOVO problem and its resolution. Let us consider m functions $F_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, ..., m$. Given $\theta \in \mathbb{R}^n$, we can sort the set $\{F_i(\theta), i = 1, ..., m\}$ in ascending order:

$$F_{i_1(\theta)}(\theta) \leq F_{i_2(\theta)}(\theta) \leq ... \leq F_{i_m(\theta)}(\theta).$$

Considering a value $1 \leq p \leq m$, we can define the LOVO function as

$$S_p(\theta) = \sum_{k=1}^{p} F_{i_k(\theta)}(\theta)$$

and the LOVO problem as

$$\min_{\theta \in \mathbb{R}^\kappa} S_p(\theta).$$

Assuming that $F_i, i = 1, ..., m$ are continuous functions we have that S_p is a continuous function but assuming that F_i's are differentiable functions we cannot conclude that S_p is differentiable. It can see by reformulating the LOVO problem as follows. Denoting $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_r\}$ as the set of all combinations of $\{1, ..., m\}$ taken p at time, we can define for each $i \in \{1, ..., r\}$ the following functions

$$f_i(\theta) = \sum_{k \in \mathcal{C}_i} F_k(\theta)$$

and

$$f_{min}(\theta) = \min\{f_1(\theta), ..., f_r(\theta)\}.$$

It is simple to note that $f_{\min}(\theta) = S_p(\theta)$ and consequently is easy to see the possible non differentiability of LOVO function. In (Andreani et al. 2009) the authors introduced methods using linear search and handle with the possible singularities in a clever way, using an approximation for the gradient of ∇f_{min} of kind of

$$\nabla f_{min}(\theta) = \nabla f_i(\theta),$$

where $i \in \mathcal{I}(\theta) = \{k \in \{1, ..., r\}; f_k(\theta) = f_{min}(\theta)\}$. Naturally, this approach to approximate the gradient can be extend for second order derivatives.

An important point for practical purposes is when we consider the LOVO problem with $p = m$ and $F_i(\theta) = (\phi(x_i, \theta) - f(x_i))^2$. In this case, LOVO problem coincides with classical least square and consequently, it can see like a generalization of least square problem. When $p \leq m$, the solution θ provides a model $\phi(x, \theta)$ free from influence of $m - p$ worst points, that is, $m - p$ possible outliers were identified.

One of the most usual ways to solve the problem of least square is by using the Levenberg-Marquardt method (Moré 1978) which is a first order method,

where derivatives of the model ϕ is used to compute the gradient of the objective function of the associated least square problem. The reason for wide use of Levenberg-Marquardt method is, in general, associated with quadratic convergence properties even using only first-order derivatives. In this direction, it is relevant to ask about Levenberg-Marquadt-based methods to solve LOVO problems in the context of adjustment functions.

`RAFF.jl` implements a Levenberg-Marquardt algorithm in the context of LOVO problems. Moreover, LOVO problems need the number of possible outliers (or equivalently trust points) to be given by the user. `RAFF.jl` solves this limitation by implementing a voting system. A voting system is a brute force algorithm, where several LOVO problems are solved with different numbers of possible outliers. The solution which most occurs among them is declared as the solution by `RAFF.jl`.

Functionality

`RAFF.jl` main methods expect as input a data set of the observed data and a model function, whose parameters one intends to adjust. The model function is a regular Julia function with 2 arguments: θ represents the parameters of the model and x represents the arguments of function f. The following function is an example of a model representing a circle

$$\phi(x;\theta) = (x_1 - \theta_1)^2 + (x_2 - \theta_2)^2 - \theta_3^2.$$

Note that θ_1 and θ_2 are parameters related to the center of the circle, while θ_3 is related to its radius. Therefore, this model contains 3 parameters. Since we think that the unknown function f is a circle, its arguments are given by a two-dimensional vector x. The observed data can be represented by the following table:

x_1	x_2	$f(x_1, x_2)$
6.1112	1.0000	0.0000
6.1213	1.0038	0.0000
5.7386	2.1833	0.0000
5.3524	3.5871	0.0000
4.7620	4.2487	0.0000
3.5713	5.3119	0.0000
2.7589	5.7070	0.0000
2.0788	5.7852	0.0000
1.6562	6.0197	0.0000
-0.3132	5.7656	0.0000
-2.1821	4.9540	0.0000
-2.4564	4.4935	0.0000

x_1	x_2	$f(x_1, x_2)$
-3.1596	3.8885	0.0000
-3.7751	2.5237	0.0000
-4.0324	1.4327	0.0000
-3.9313	-0.0512	0.0000
-2.5500	-2.6777	0.0000
0.1608	-4.0584	0.0000
1.8317	-3.9190	0.0000
4.1909	-2.7741	0.0000
5.6775	-0.6784	0.0000
5.8163	0.2874	0.0000
5.9457	0.3777	0.0000
-1.1213	3.1213	0.0000
-3.5961	-3.5961	0.0000

In this example, we are trying to find a circle to fit the observed data. Therefore, all the values of f should be zero. The observed data is shown in Figure 1. The dots represent the observed data, the red ones being the outliers. They were generated as a perturbation of the points lying in the blue dashed circle. Using the Least Squares technique with the model above, the red circle is found. When RAFF.jl is applied to the same problem, it correctly identifies the two outliers. The resulting circle is depicted as the green circle, very close the true circle.

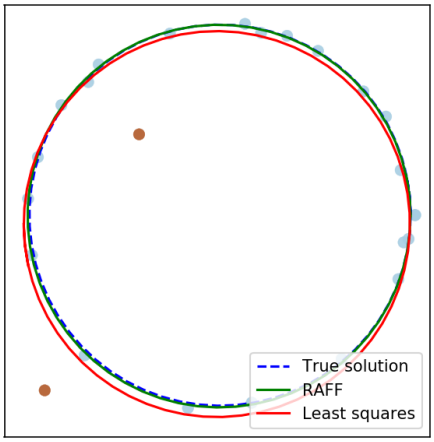

Figure 1: Points representing a circle. The red dots are two outliers that should be ignored. The blue dashed circle is the true one, while the red was obtained by traditional Least Squares techniques and the green one was obtained by RAFF.jl.

Additional features

The user may also provide more information to `RAFF.jl`, such as the expected number of *trusted* observations. Additional methods and options are also available to more advanced users, such as generation of random test data and multistart strategies. Derivatives of the model ϕ can also be provided, what results in a faster executing time. When they are not provided by the user, `RAFF.jl` uses Julia's `ForwardDiff.jl` (Revels, Lubin, and Papamarkou 2016) package.

`RAFF.jl` can be run in serial, parallel and distributed environments. Parallel and distributed methods use the native `Distributed.jl` package. The distributed version is a centralized implementation that does not use shared arrays, therefore, can be run both locally or in a cluster of computers.

This package is intended to be used by all experimental researchers who know a little about mathematical modeling and fitting functions.

Instalation and usage

`RAFF.jl` is an open-source software that can be downloaded from Github. All the description for first time usage or its API is available at its documentation.

References

Andreani, R., C. Dunder, and J. M. Martínez. 2005. "Nonlinear-Programming Reformulation of the Order-Value Optimization Problem." *Mathematical Methods of Operations Research* 61 (3). Springer: 365–84. doi:10.1007/s001860400410.

Andreani, R., J. M. Martínez, L. Martínez, and F. S. Yano. 2009. "Low Order-Value Optimization and Applications." *Journal of Global Optimization* 43 (1): 1–22. doi:10.1007/s10898-008-9280-3.

Liu, Yang, and Wenping Wang. 2008. "A Revisit to Least Squares Orthogonal Distance Fitting of Parametric Curves and Surfaces." In *Advances in Geometric Modeling and Processing*, edited by Falai Chen and Bert Jüttler, 384–97. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-79246-8_29.

Martínez, J. M. 2012. "Generalized Order-Value Optimization." *Top* 20 (1). Springer: 75–98. doi:10.1007/s1175001001691.

Moré, Jorge. 1978. "The Levenberg-Marquardt Algorithm: Implementation and Theory." *Numerical Analysis*. Springer, 105–16. doi:10.1007/BFb0067700.

Revels, J., M. Lubin, and T. Papamarkou. 2016. "Forward-Mode Automatic Differentiation in Julia." *ArXiv:1607.07892 [Cs.MS]*. https://arxiv.org/abs/1607.

07892.