Geoinformatic project

Efficient outliers rejection in positioning of mobile devices

Dorotea Rigamonti
969365

Academic Year 2021/2022
Tutor: Prof. Ludovico Biagi

Contents

1. Introduction

In Least Squares (LS), the linearized functional model between M observables and N unknown parameters is given. LS provides estimates of parameters, observables, residuals and a posteriori variance. To identify outliers and to estimate accuracies and reliabilities, tests on the model and on the individual residuals can be performed at different levels of significance and power. However, LS is not robust: one outlier could be spread into all the residuals and its identification is difficult. A possible solution to this problem is given by a Leave One Block Out (LOBO) approach. Let's suppose that the observation vector can be decomposed into m sub-vectors (blocks) that are reciprocally uncorrelated: in the case of completely uncorrelated observations, m = M. A suspected block is excluded from the adjustment, whose results are used to check it. Clearly, the check is more robust, because one outlier in the excluded block does not affect the adjustment results. The process can be repeated on all the blocks, but can be very slow, because m adjustments must be computed. To efficiently apply Leave One Block Out, an algorithm has been studied by Ludovico Biagi and Stefano Caldera in 2012 [1]. The usual LS adjustment is performed on all the observations to obtain the 'batch' results. The contribution of each block is subtracted from the batch results by algebraic decompositions, with a minimal computational effort: this holds for parameters, a posteriori residuals and variance.
The aim of this project is to implement in python the Efficient Leave One Block Out (ELOBO) algorithm and to perform two tests to on the reliability and the speed of ELOBO procedure with respect to classic LS solution and repeated LOBO procedure.

2. Specification Analysis

2.1 Input and output

The algorithm takes as input:

- the matrices of the least squares problem expressed as array numpy
- An array containing the dimension of each blocks
- Number of parameters
- Number of observation
- A significance level

Given the inputs the algorithm return:

- the solution of the leave one out procedure

2.2 Data structure

The library implemented has to perform the numerical leave one out algorithm using the blocks of the given matrices. The main problem was to be able to access to the single block and use it or remove it when necessary. The data structure of the dictionary was chosen to deal with it: the key-value structure allows to assign each block its position and use it to quickly access to it.

2.3 Used libraries

The libraries needed to run this code are:

- Numpy : to handle every operation on the inputs numpy arrays
- Scipy.stat: to compute the limit value of a Fisher (f.ppf()) and a Chi-Square (chi2.ppf())
- Timeit: we use only the function default_timer() in speed_test.py to return the current time

3. Structure of the code

The main file is elobo_library.py, the library that contains all the functions to perform ELOBO and LOBO procedures.
Other two additional python files where created :

- reliability_test.py: a test on simulated data to compare the reliability of ELOBO algorithm, repeated LOBO algorithm and classic Least Squares solution;
- speed_test.py: a numerical test on simulated data to explore the operational time of ELOBO and LOBO with different combination of number of parameters and number of blocks.

3.1 elobo_library.py

Algorithm

The algorithm of ELOBO and LOBO is very similar.

1. Divide the input into blocks by using the function *split_blocks()*
2. Compute the solution of a least square problem calling the function *least_square_block()*
3. For each block:
    a. implement the leave one out procedure: numerical solution in case of ELOBO and using classic least square solution if LOBO each time not considering a block with the function *copy().*
    b. Compute the theoretical an the empirical Fisher value (*scipy.stat.f.ppf()*) and compute the ratio between them
4. Find the ratio maximum value and check if the correspondent block is an outlier comparing it with the limit threshold
5. Compute the final value by removing the outlier if found
6. Compute the covariance matrices

For a more detailed description about the algorithms refer to ELOBO_flowchart.pdf document in the repository of the code.

Functions

Create a dictionary of the blocks of an array: *split_block()*

Table 3.1: *split_block()* function

Inputs	matrices of the least squares problem expressed as array numpy
	Array of the dimension of each block
	Number of parameters
outputs	dictionaries containing the input matrices and array split in blocks
	dictionary storing the raw from which each block starts
What it does	This function splits the input matrices and arrays into blocks and put them in dictionaries. At each block is associated a position that corresponds to the order with which it appears. This position is used as key in the dictionary.

Perform a least squares procedure with blocks: *least_quares_block()*

Table 3.2: *least_quares_block()* function

Used libraries	Numpy
Inputs	Design matrix
	Vector of observation
	Outputs of *split_block()* function
	Number of parameters
	Number of observations
Outputs	Estimates of the least square procedure
	Dictionary of the estimated residual removing the k-th block
What it does	It performs the least square procedure using data split in blocks

Copy function: *copy()*

Table 3.3: *copy()* function

Inputs	Any dictionary
	A key
Outputs	A copy of the dictionary without the value corresponding to the given key
What it does	This function create a copy of a given dictionary and remove from the copy the element corresponding the given key. It was created to handle the need to implement a for-loop each time removing a different block, without modify the original dictionary as instead the *pop()* function does.

Efficient Leave One Block Out function: *elobo()*

Table 3.4: *elobo()* function

Used libraries	Numpy
	Scipy.stat.f.ppf
Inputs	Matrices of the least squares problem expressed as array numpy
	Array of the dimension of each block
	Number of parameters
	Number of observations
	Significance level
Used functions	*Split_blocks()*
	Least_squares_blocks()
Outputs	Final estimates removing the outliers
	The key of the outlier block
What it does	Given the input data of the problem it split them into blocks and performs the numerical efficient leave one out procedure computing the final solution by removing the outlier if found.

Leave One Block Out function: *classic_lobo()*

Table 3.5: classic_*lobo()* function

Used libraries	Numpy
	Scipy.stat.f.ppf
Inputs	Matrices of the least squares problem expressed as array numpy
	Array of the dimension of each block
	Number of parameters
	Number of observations
	Significance level
Used functions	*Split_blocks()*
	Least_squares_blocks()
Outputs	Final estimates removing the outliers
	The key of the outlier block
What it does	Given the input data of the problem it split them into blocks and compute the solution by using the classic Leave One Out procedure repeating the least squares each time removing a different block.

Classic outlier detection with Least Squares: *outlier_ls()*

Table 3.4: *outlier_ls()* function

Used libraries	Scipy.stat.chi2.ppf
Inputs	A priori variance
	Estimated variance
	Dictionary of estimated residuals
	dictionary storing the raw from which each block starts
	Number of parameters
	Number of observations
	Significance level
Outputs	Outlier position
What it does	function that finds the outlier from the global least square solution it identify the presence of an outlier with a chi-square test and if present the as outlier is taken the block with the higher residual

3.2 reliability_test.py

The aim of this code is to compare the reliability of ELOBO, repeated LOBO and classical Least Squares (LS) test. The simulated data represent a regular grid of 25 points taken on an 1000x1000px image with coordinates (i1,j1). An image2 is obtained by the transformation of image1.
100 simulations with independently generated noise have been iterated for different levels of outliers (1.5, 1.75, 2.00, 2.50, 3.00, 3.50 and 4.0 pixels) added each time to the following pixel.
All the statistical tests have been performed with a significance level $\alpha = 0.01$ and the results have been clustered in three classes:

- {LOBO/ELOBO/LS}_OK: correct identification of the outlier
- {LOBO/ELOBO/LS}_NO: no outlier is identified by the test
- {LOBO/ELOBO/LS}_WO: one outlier is identified, but in the wrong observation

The results reported in Table 3.1 shows how ELOBO and LOBO reliability is the same as expected. Both the two leave one out algorithms always identify the outlier and 96% of the time correctly . Also classical LS always identify the outliers, 95% in a correct way.

Table 3.1 Reliability comparison between LOBO, ELOBO and classical test (LS)

outlier	lobo_no	lobo_ok	lobo_wo	elobo_no	elobo_ok	elobo_wo	ls_no	ls_ok	ls_wo
1.5	0	86.2	13.8	0	86.2	13.8	0	85.86	14.24
1.75	0	93.92	6.08	0	93.92	6.08	0	91.76	8.24
2	0	97.08	2.92	0	97.08	2.92	0	95.56	4.44
2.5	0	99.88	0.12	0	99.88	0.12	0	99.24	0.76
3	0	100	0	0	100	0	0	99.72	0.28
4	0	100	0	0	100	0	0	100	0

3.3 speed_test.py

This script performs a numerical comparison between ELOBO and repeated LOBO. It tests the efficiency of ELOBO end LOBO with respect to the operational time with different combination of number of parameters (n) and number of blocks (m). M=2000 number of observation are generated with a simple model adding a random noise with standard deviation of 0.1.
default_timer() from the library timeit measures the current time and has been used to compute the execution of *elobo()* and *classic_lobo()* functions.
The results are expressed as ration between the execution time of LOBO and the one of ELOBO and are shown in Table 3.2. The ratio is always greater than one revealing that ELOBO it's faster than LOBO in every case. We can also state ELOBO performs better with a higher number of blocks.
Due to the slowness of the program the last case with 250 parameters could not be performed.

Table 3.2 Ratio between LOBO execution time and ELOBO execution time with different combination

		n° of blocks (n° of observation per block)									
		2000 (1)	1000 (2)	500 (4)	250 (8)	100 (20)	50 (40)	25 (80)	10 (200)	5 (400)	2 (1000)
n° of parameters	1	201.87	203.88	188.89	177.1	168.53	162.17	156.43	147.53	129.5	88.07
	10	107.05	119.15	119.89	118.01	116.28	114.83	113.39	110.98	105.77	91.18
	50	104.17	153.96	153.14	151.44	149.87	148.54	147.1	144.93	140.46	127.5
	100	133.47	138.06	237.93	136.91	135.81	134.81	133.8	132.39	129.48	120.81
	250	157.24	161.39	160.44	158.29	156.67	155.29	154.14	152.66	149.69	141.97
	500	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

4. Conclusion and comments

The algorithm works with any least squares problem. Results show the advantages of using ELOBO procedure with respect to LOBO and the classical LS solution as shown in the article[1].
With big amount of data, which were simulated by big number of iterations, the algorithm took some time to work and it didn't manage to compute the last case of speed test probably due to computing capacity of personal computer.

Referencies

[1] L. Biagi, S. Caldera *Efficient Leave One Block Out approach to identify outliers,* J. Appl. Geodesy, Vol. 7 (2013), pp. 11–19