IMPLEMENTATION OF A TRIGGER ALGORITHM FOR SUPERCDMS

Chiara Magliocca

Supervisor : Noah Alexander Kurinsky

What is the aim

The aim of this algorithm is to select the data that we can consider valid for a further analysis, starting from a sample of data in ADC values taken directly by the detector without any hardware trigger.

The algorithm discriminate good pulses from noise thanks to a **threshold**, set after an accurate study of noise distribution, and examine the eventuality of **contamination between pulses** that occur in the same event or in two different ones.

The algorithm gives, as a result, two two-dimensional lists:

- includes the current values of the events that passed all the filters and are considered "good" so far
- includes the indexes of these events

I then studied the efficiency of the algorithm, that, as a result, can be greatly improved.

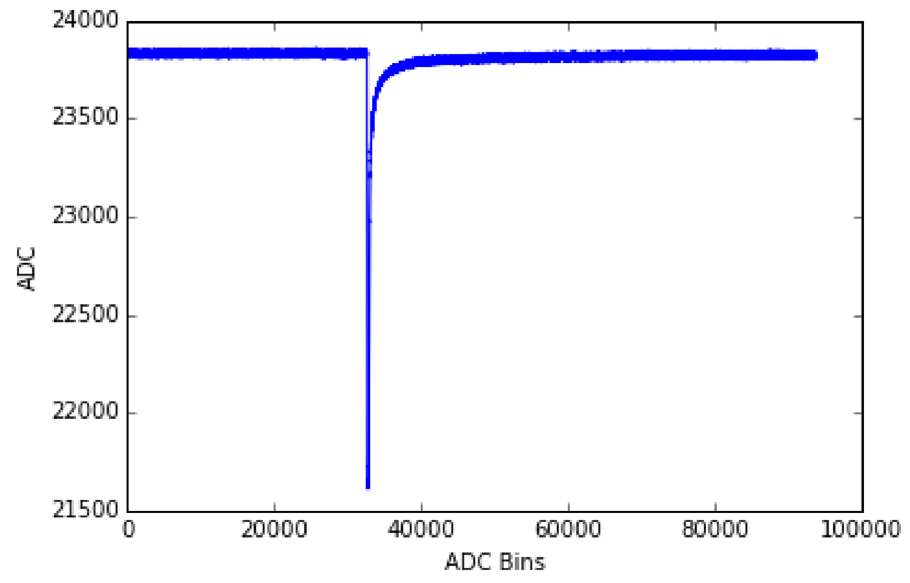
SO…

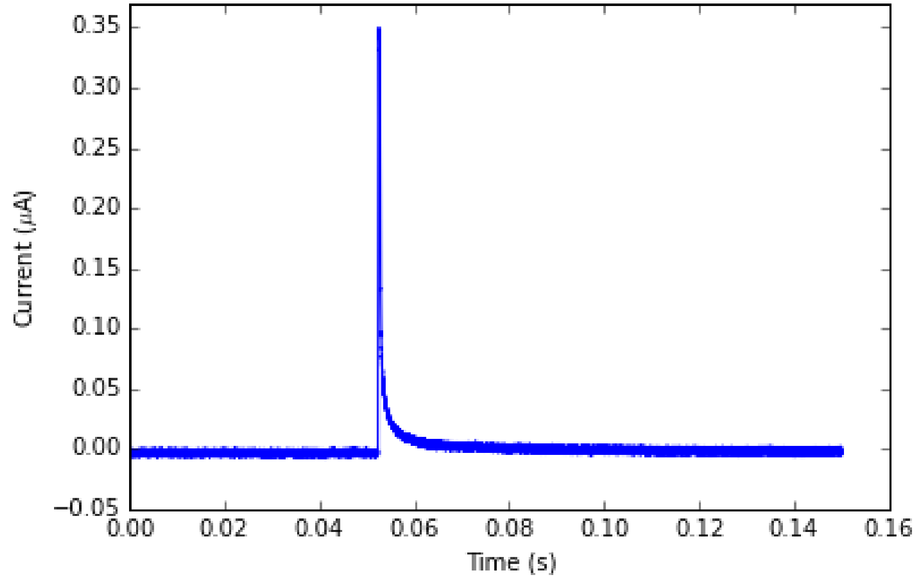
An event can be considered "**GOOD**" if:

- It contains at least one pulse
- All the pulses start and end within the acquisition frame
- No pulse is contaminated by another one that occur in the same event
- No pulse is contaminated by another one that occur in the previous event

THE CODE

After reading the ADC values data from a file, the algorithm stores them in a two-dimensional list called res(). Data are then converted from ADC values to current, centered to zero and reversed (so that the pulses will be positive and the subsequent studies simplified) and stored in a list called pulse().





Study of the Noise Distribution

An event can be considered valid for a further analysis if at least a pulse is present. But how can we know if that pulse is actually a signal or just noise?

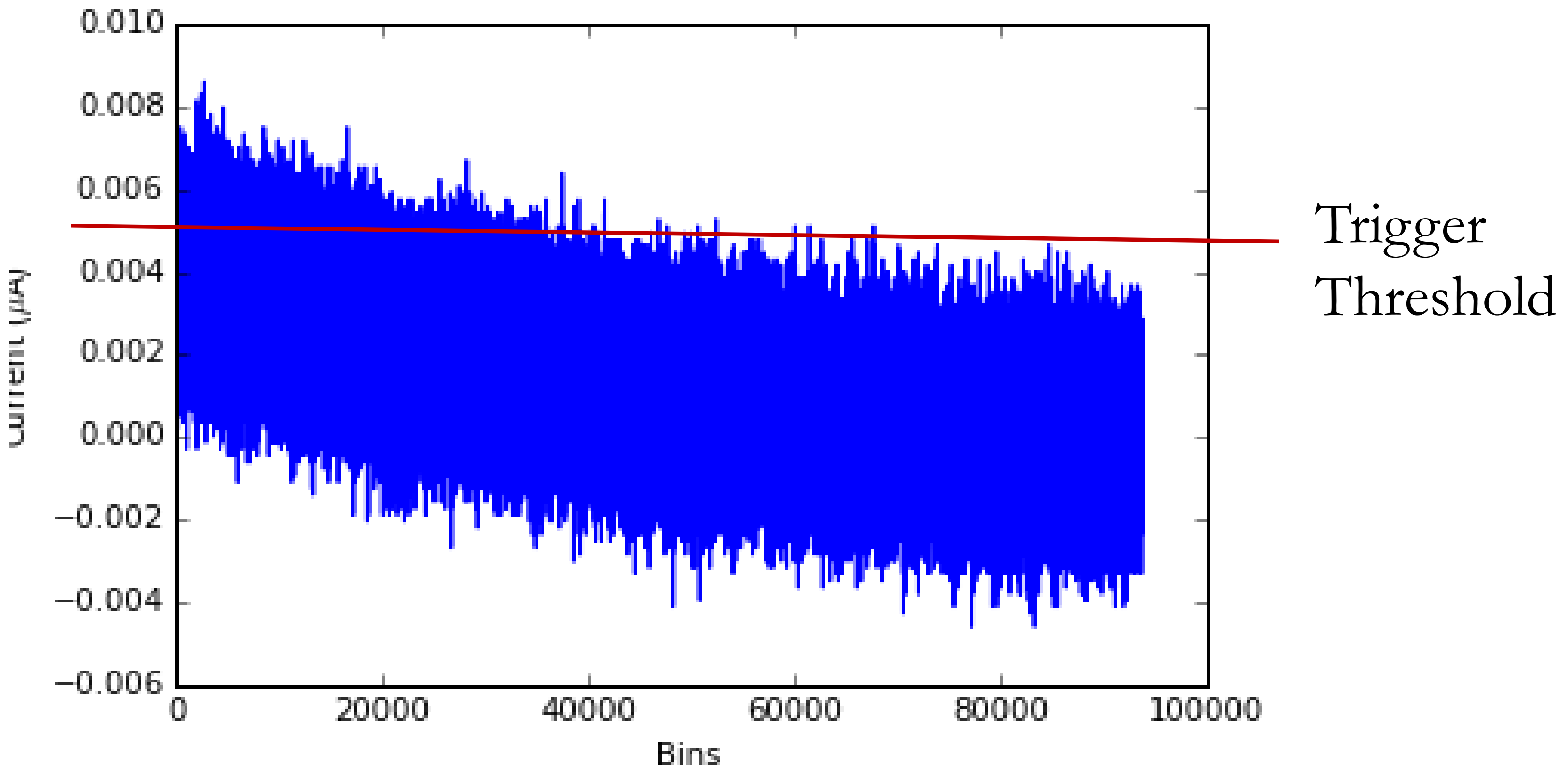
→Easy : we need to set a threshold and discriminate pulses from noise.

Study the noise distribution → the threshold will be set at 3σ

We start selecting all the events that contain no pulses, but just noise. To do this, a temporary threshold is set to 0.02 μA (this value has been chosen after an analysis of a sample of events with both noise and signal). The indexes of the events that contain just noise are then stored in noise(), while the events that contain both noise and signal are stored in nonoise().

At this time the standard deviation (σ) of each just-noise event is calculated to determine, at the end, the mean value of all of these

$$\boxed{thr = 3 * mean \approx 0.005}$$


0.010
0.008
0.006
0.004
0.002
0.000
−0.002
−0.004
−0.006
Current (μA)
0
20000
40000
60000
80000
100000
Bins
Trigger Threshold

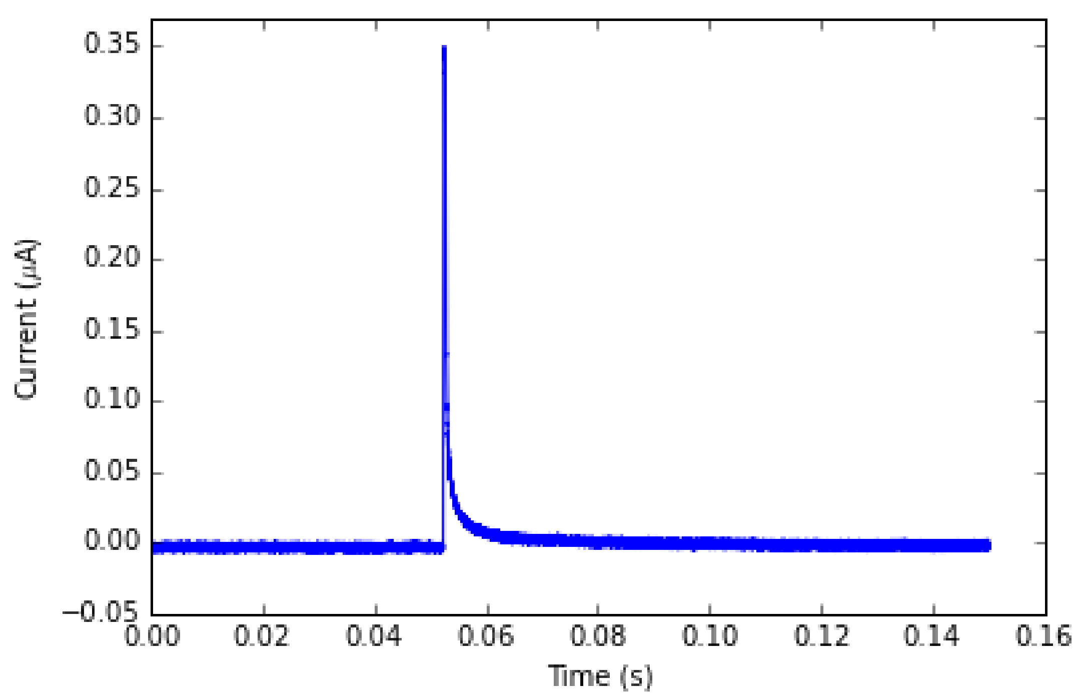
Low-Pass Filtering

What we need to do now is low-pass filter the pulse to get rid of high-frequency information which are not actually useful for this algorithm. To do this, we calculate the moving average.

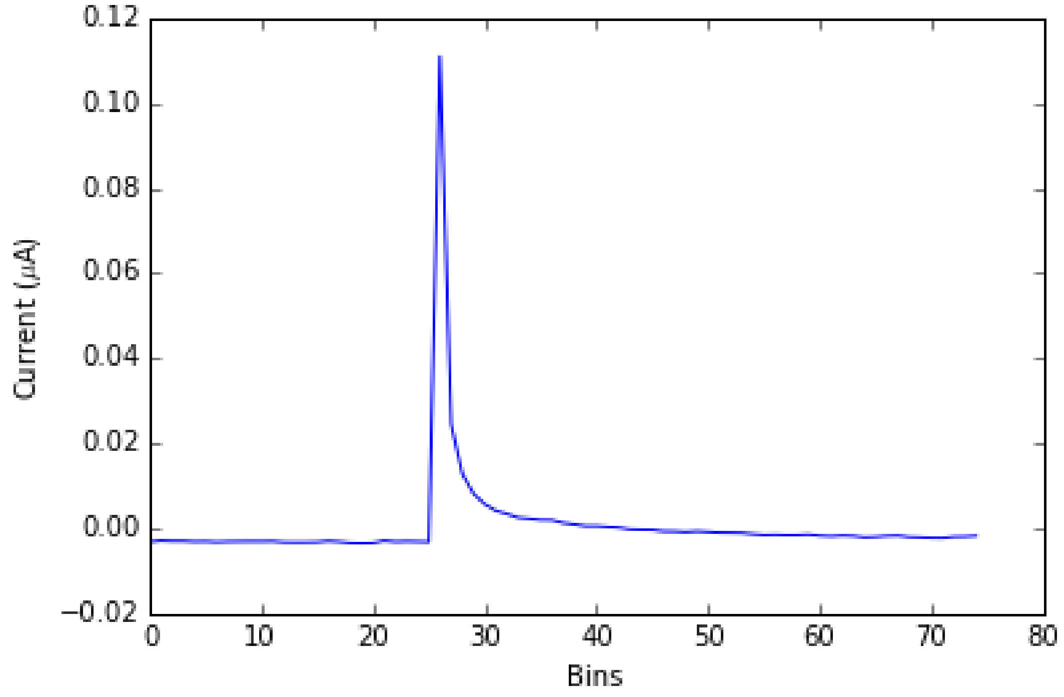
Each event is divided into intervals of fixed length (=sample period) T_sample (chosen to be 0.002 s) and the mean value of the signal in each interval is calculated (to clean it up from additional noise and smooth it).

T_sample parameter can be change to optimize the algorithm, but, to understand its optimal value, an analysis of the algorithm's efficiency is needed.
→ The mean values of the intervals are stored in a two-dimensional list mean[i][j], where i is the index of the event and j the one of the interval.

From now on, we will always use the filtered pulses.



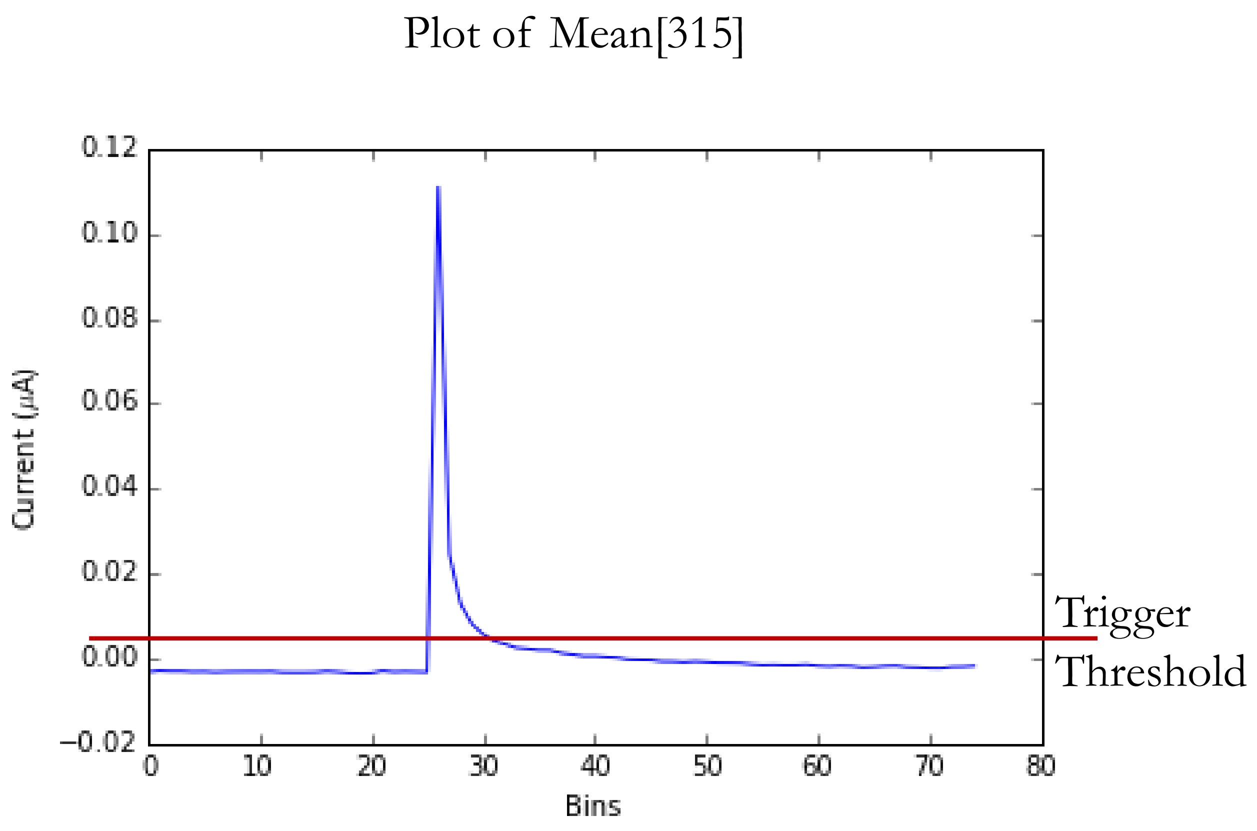
Current (μA)
Time (s)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16



Current (μA)
Bins
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
0
10
20
30
40
50
60
70
80

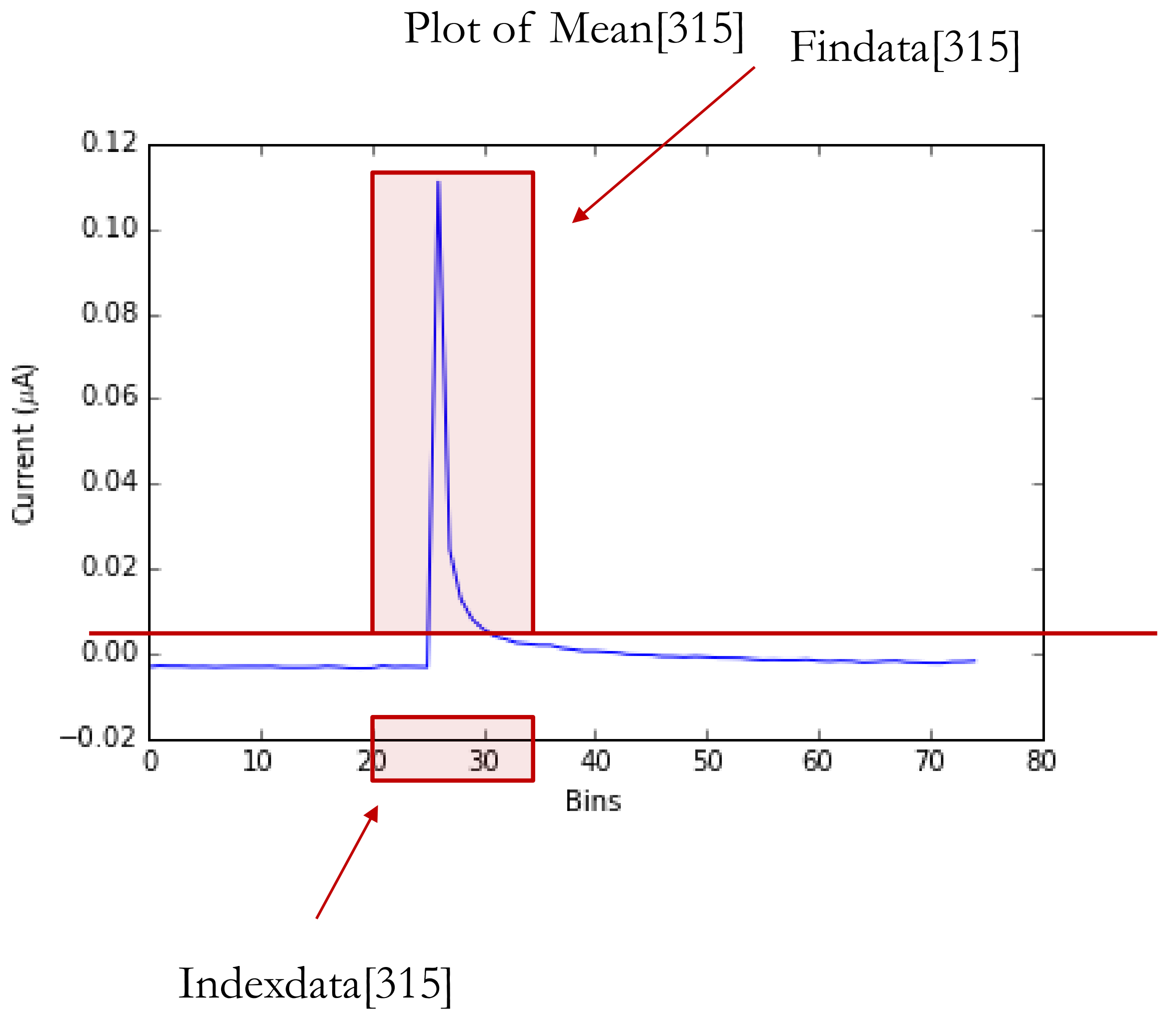
Is now time to check the number of pulses present in each event.
To do so, the algorithm scans all the events and, in each event, checks whether a value exceeds the trigger threshold (the one that was previously fixed). If it does, it saves the index of the interval that include that value in a two-dimensional list called indexdata() and the value itself in findata().
Then it looks for groups of consecutive indexes in indexdata() and saves in first() and last() respectively, the index of the values of the first and last signal above threshold (basically the first and last value to cross the threshold).

→The length of first() (or equally the length of last()) is actually the number of pulses present in the event under examination.



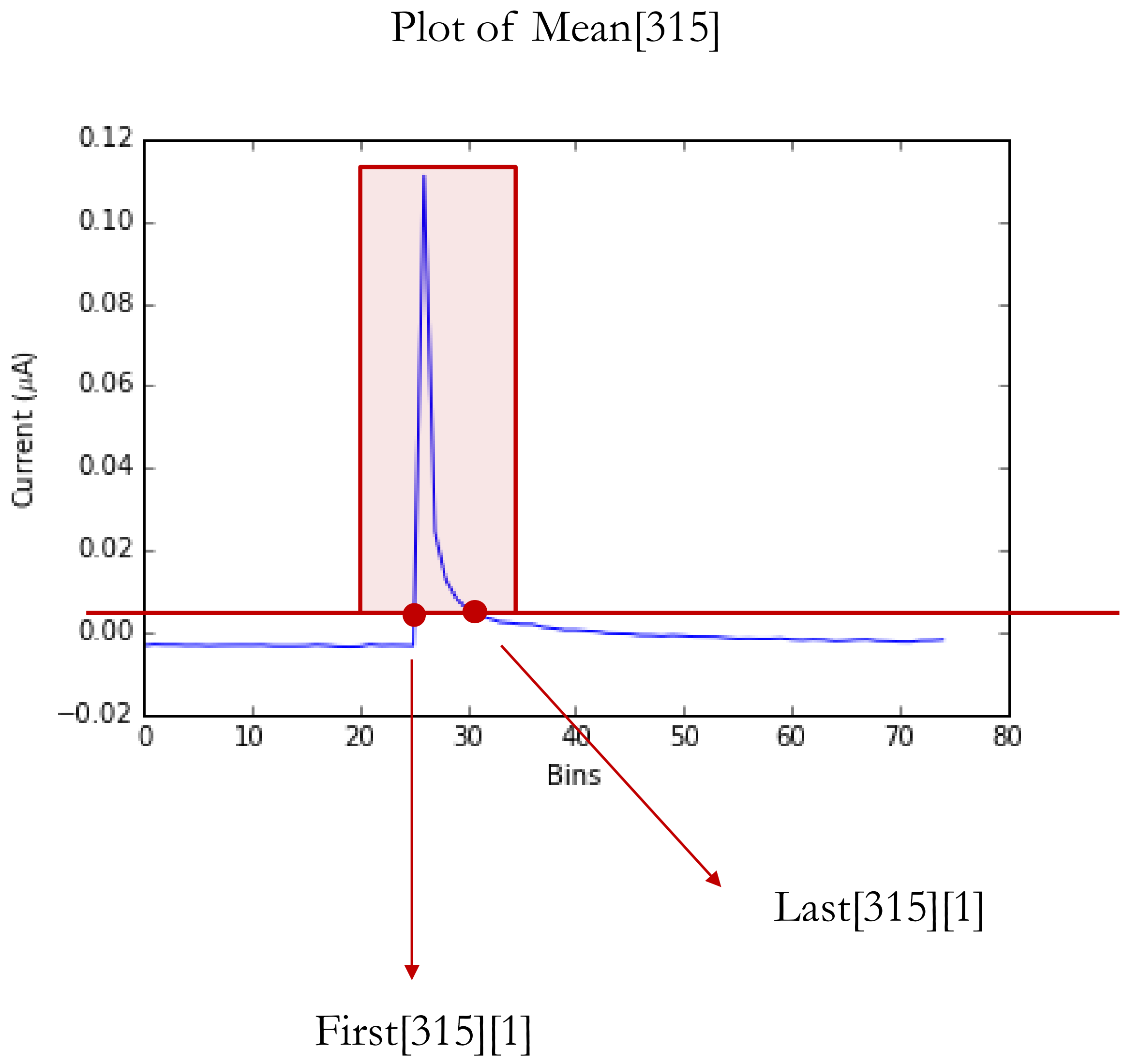
Is now time to check the number of pulses present in each event.
To do so, the algorithm scans all the events and, in each event, checks whether a value exceeds the trigger threshold (the one that was previously fixed). If it does, it saves the index of the interval that include that value in a two-dimensional list called indexdata() and the value itself in findata().
Then it looks for groups of consecutive indexes in indexdata() and saves in first() and last() respectively, the index of the values of the first and last signal above threshold (basically the first and last value to cross the threshold).

→The length of first() (or equally the length of last()) is actually the number of pulses present in the event under examination.



Is now time to check the number of pulses present in each event.
To do so, the algorithm scans all the events and, in each event, checks whether a value exceeds the trigger threshold (the one that was previously fixed). If it does, it saves the index of the interval that include that value in a two-dimensional list called indexdata() and the value itself in findata().
Then it looks for groups of consecutive indexes in indexdata() and saves in first() and last() respectively, the index of the values of the first and last signal above threshold (basically the first and last value to cross the threshold).

→The length of first() (or equally the length of last()) is actually the number of pulses present in the event under examination.



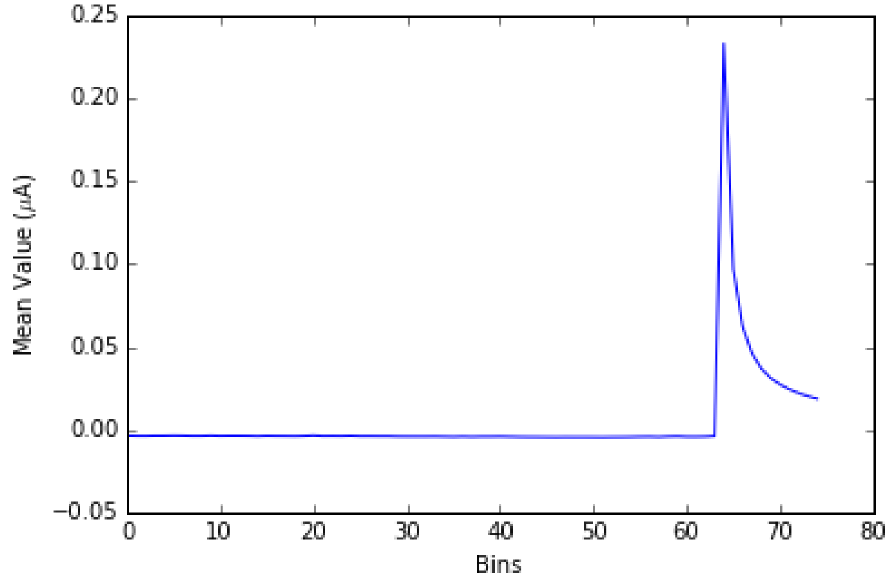
Pulse Contamination Study

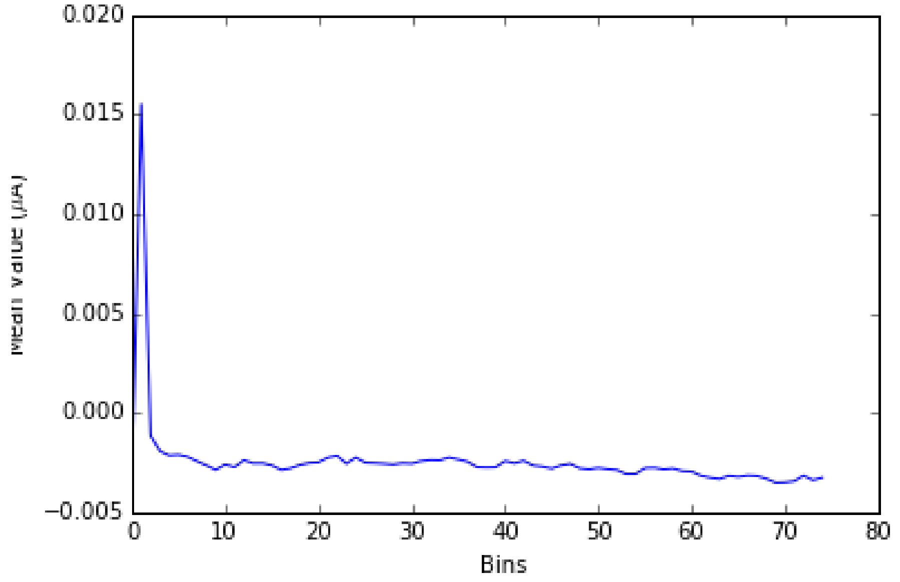
As we said before, we consider "good" an event in which all the pulses are not contaminated by another one that occur in that same event or in the one before. For this reason, to discriminate "good" event from "bad" event it is necessary to study the eventual contamination between pulses.

Before starting, the algorithm throws all the event with at least an uncomplete pulse.

Such event can occur when:

- The acquisition stops before the last pulse is finished → We check if the last value of the last pulse is still above the trigger threshold. In this case the pulse is not finished and the entire event is thrown.
- The pulse starts before the acquisition, so that we lose the beginning of the signal → We check if the first value of the first pulse is already above the trigger threshold. In this case the pulse started before the acquisition and the entire event is thrown.





Is a pulse contaminated by another one that occured in the **previous** event ?

To understand if a pulse is contaminated by another one that occur in the previous event, the algorithm calculate the **slope** of the k points before 'first' (=first value above trigger threshold of that specific pulse). At this time k = 6.

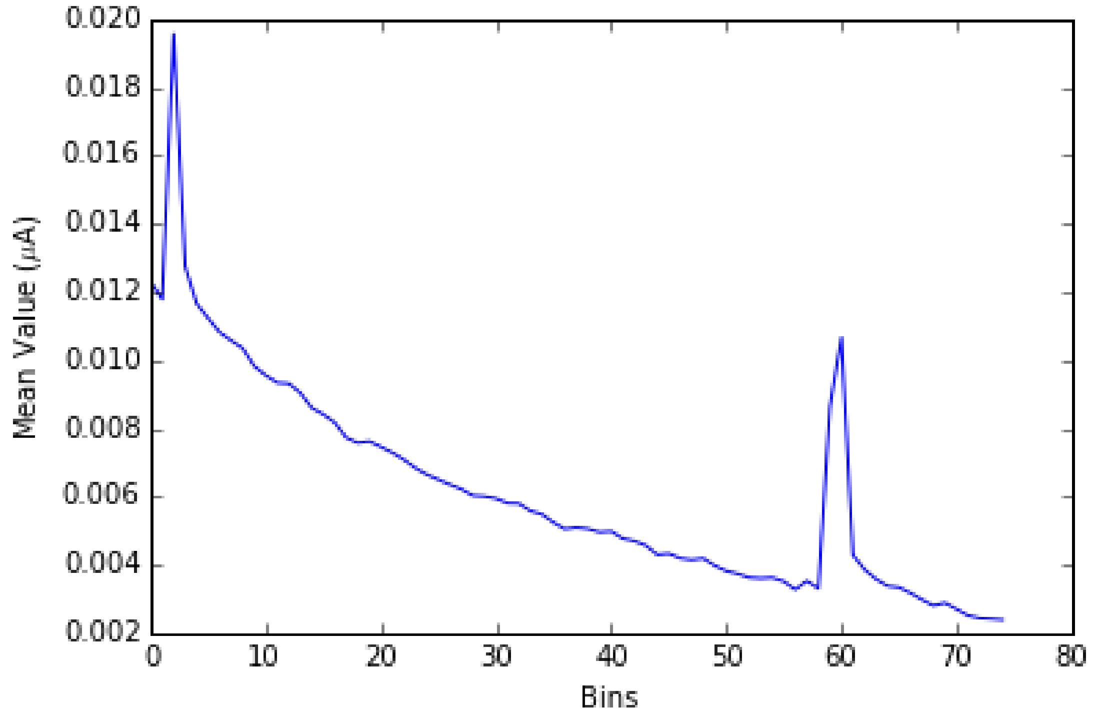
- slope < 0 : it means that the pulse under consideration occur on the tail of a previous pulse → the algorithm throws the entire event and start analyzing the next one
- slope ≥ 0 → the algorithm saves the index of the event in a list called indexgoodevent().

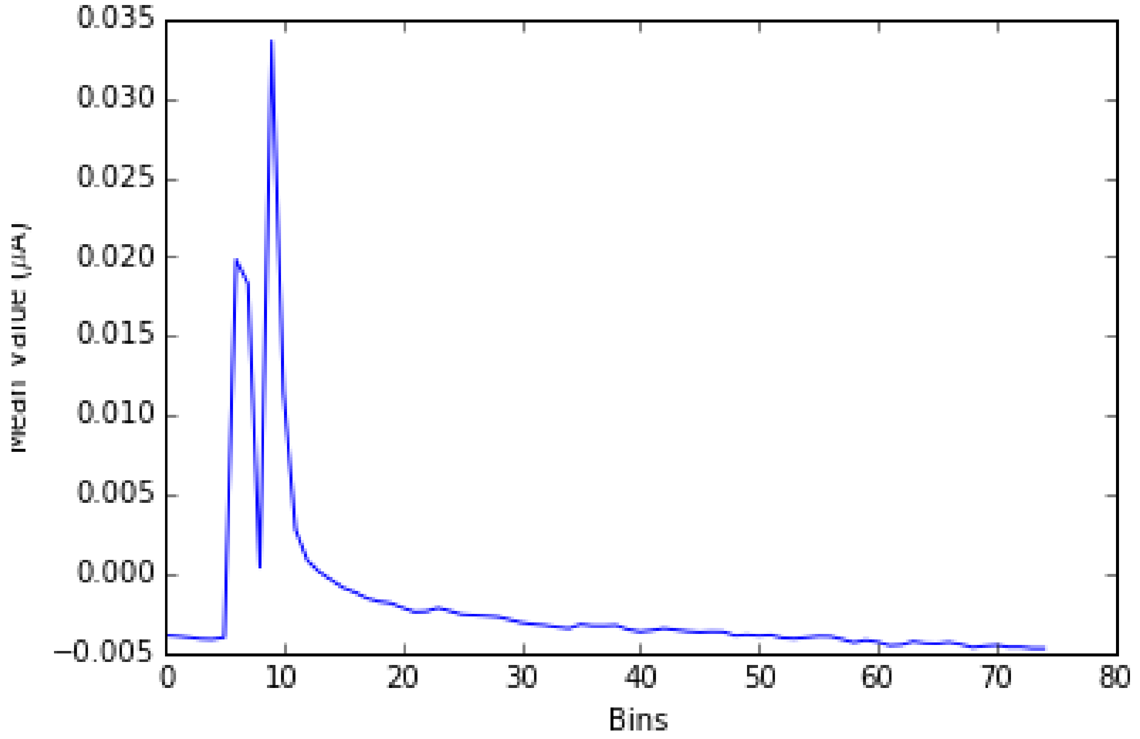
Is a pulse contaminated by another one that occur in the **same** event ?

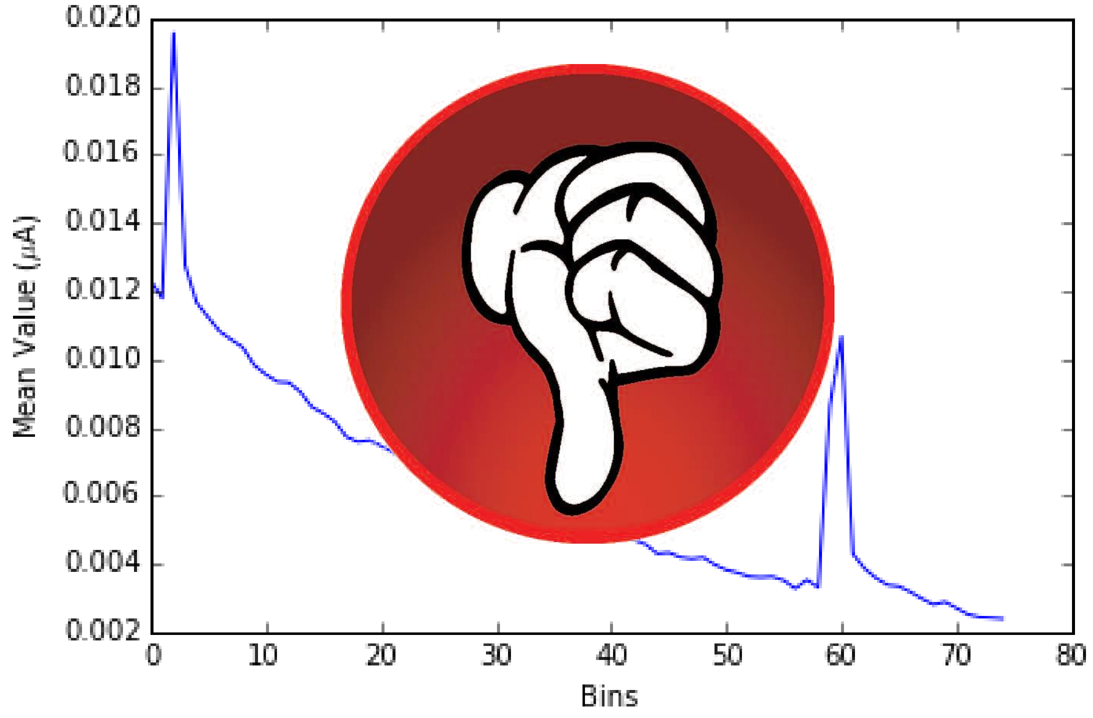
The algorithm scans only the events included in indexgoodevent(), to analyze just the pulses considered good so far.

A pulse is contaminated if another one occurs before it is finished (another pulse is **present on its tail**) → the algorithm check only the tail of the pulse, starting from the index of the interval that contain the maximum value of the current.

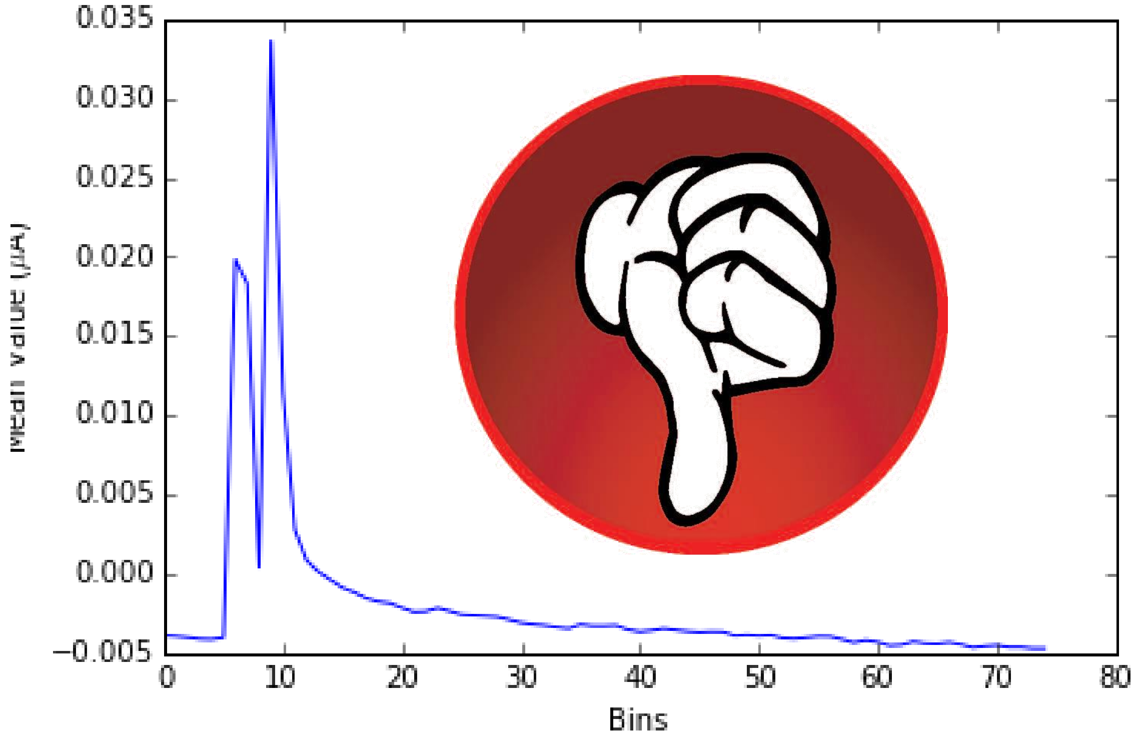
After the maximum value, the signal decreases, so each value should be greater than the subsequent one. If this condition is respected until the value of last() of that specific pulse for all the pulses in the event, the algorithm saves its index in indexresult() and the corresponding current values in result(), otherwise the entire event is thrown.


0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
Mean Value (μA)
0
10
20
30
40
50
60
70
80
Bins

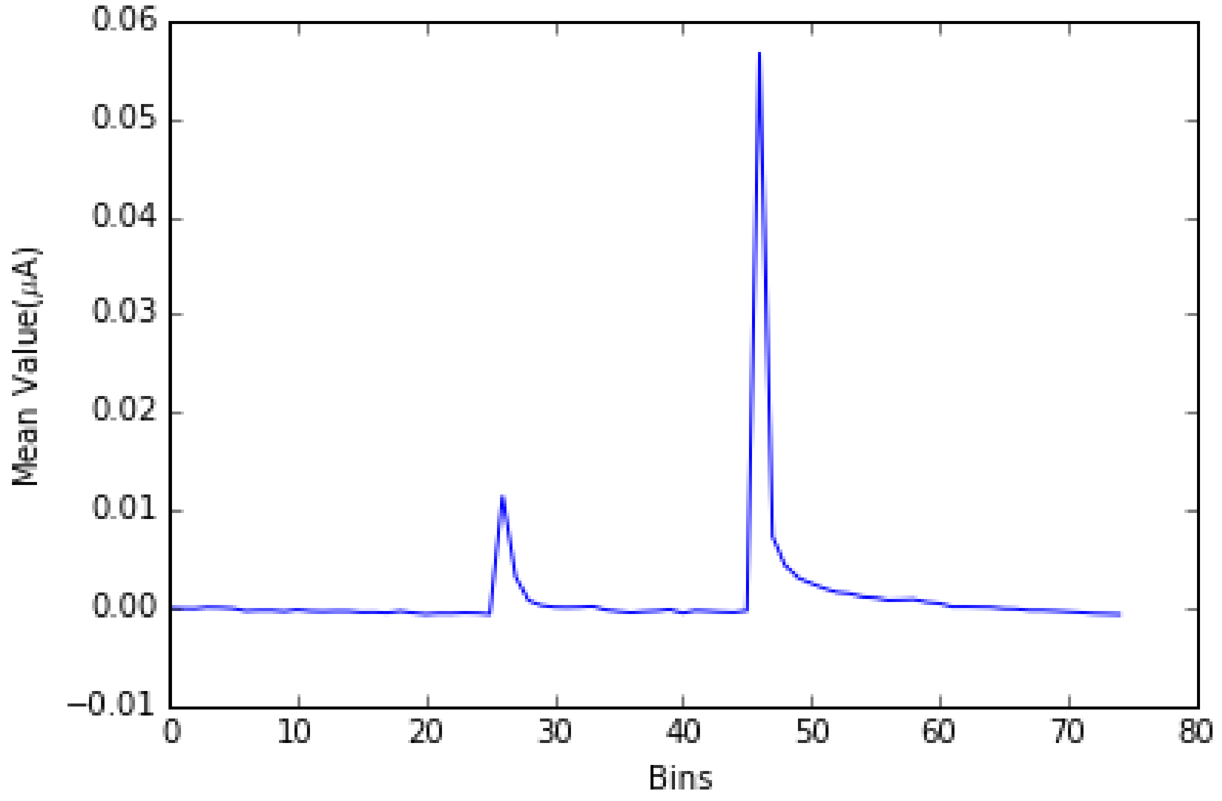

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−0.005
Mean Value (μA)
0
10
20
30
40
50
60
70
80
Bins



0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
Mean Value (μA)
0
10
20
30
40
50
60
70
80
Bins



0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−0.005
Mean Value (μA)
0
10
20
30
40
50
60
70
80
Bins



0.06
0.05
0.04
0.03
0.02
0.01
0.00
−0.01
Mean Value(μA)
0
10
20
30
40
50
60
70
80
Bins

At the end

→ indexresult() contain the indexes of the filtered events (the events in which at least a pulse is present, all the pulses start and end within the acquisition frame and with no contamination)

→ result() stores the current values of each of these event.

We can now study the efficiency of this algorithm to also select the optimal k e T_sample parameters.

STUDY OF THE EFFICIENCY

I decided to study the efficiency of this algorithm with:

Real Data

I looked at the plots of a sample of events as they are taken by the detector (res ()), and I selected by hand the pulses that, to me, were good and bad. I stored the indexes of the events in two different lists and I used the algorithm on my selected events.

Fake Data

I implemented an algorithm that produce fake data that follow the real data distribution. Noise and contamination between pulses within the same acquisition frame are present.

Real Data

Output

Input

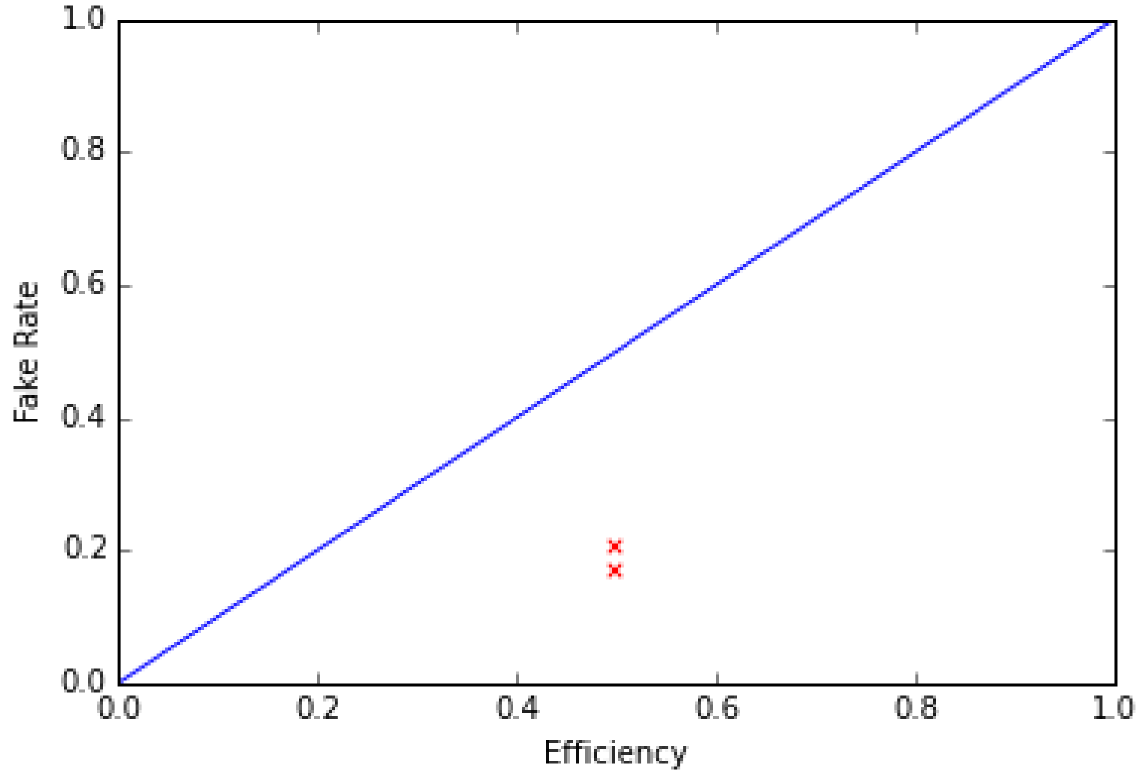
Sample Good Events = 100	Sample Bad Events = 100	
	Good	**Bad**
Good	50 %	50%
Bad	17%	83%

Output

Input

Sample Good Events = 100	Sample Bad Events = 280	
	Good	**Bad**
Good	50%	50%
Bad	20.7%	79.3%

Plot of the Efficiency – Real Data Study



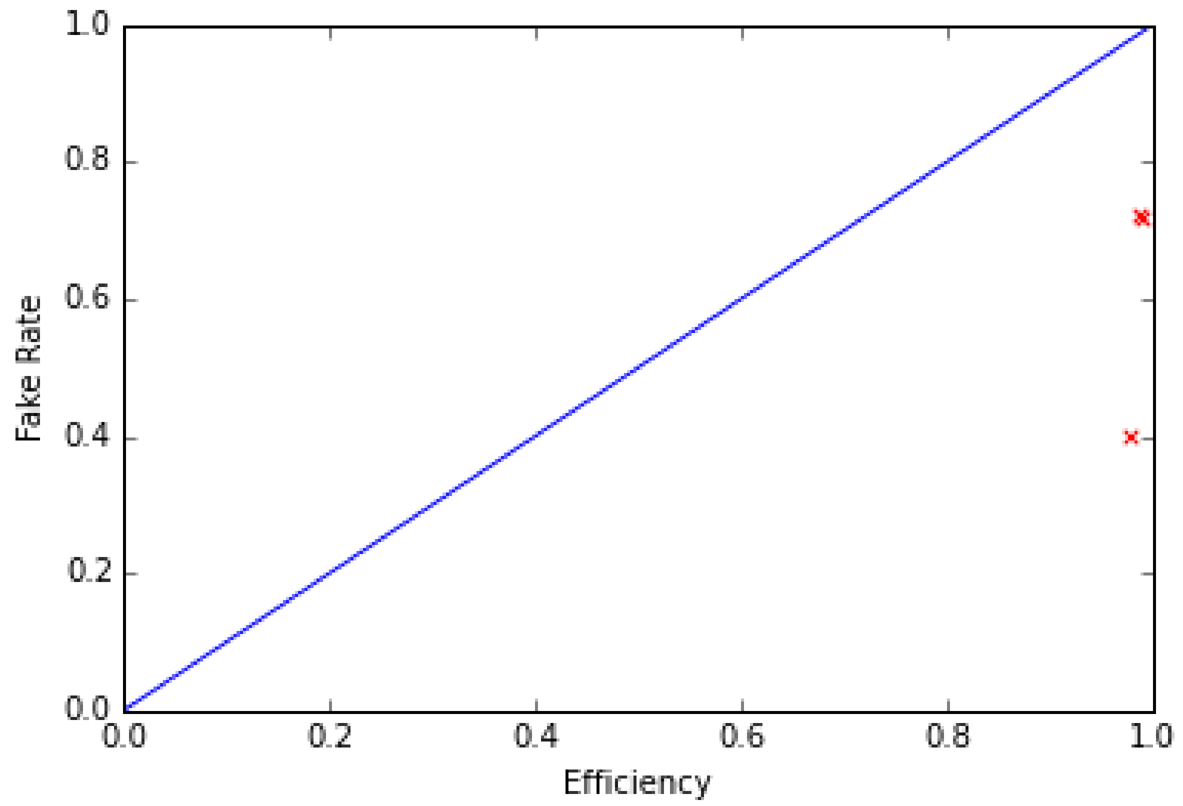
Fake Data

Sample Events =	100	
	Good	**Bad**
Good	98%	2%
Bad	40%	60%

Sample Events =	500	
	Good	**Bad**
Good	98.8%	1.2%
Bad	72%	28%

Sample Events =	1000	
	Good	**Bad**
Good	99.2%	0.8%
Bad	71.7%	28.3%

Plot of the Efficiency – Fake Data Study



CONCLUSIONS

The algorithm can be improved !

But .. In which way ?

Changing the Parameters

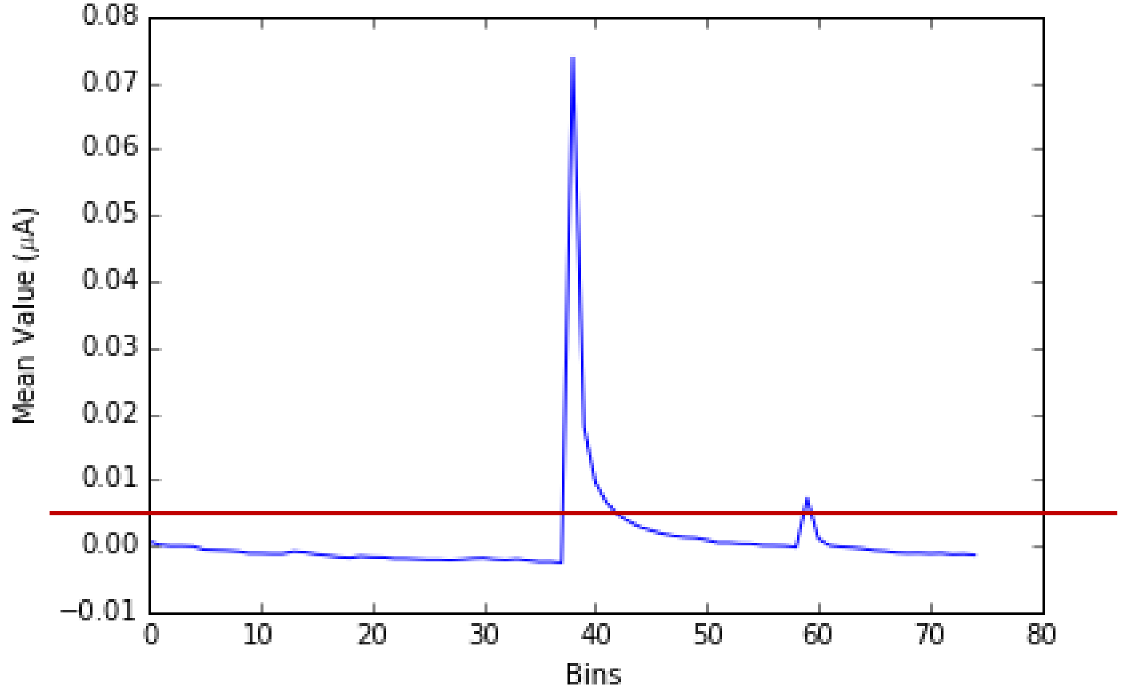
The two parameters

1. T_sample (sample period in which we calculate the moving average)
2. k (number of intervals in which we calculate the slope)

can be changed and optimized to improve the efficiency of the algorithm.

Last ( )

The list last() is filled with the last value of each pulse that cross the trigger threshold, but we cannot really know if at that time the pulse is finished or not. If it is not, another pulse can occur at the very end of its tail (after the value stored in last()) and the algorithm will not be able to flag that pulse as contaminated.



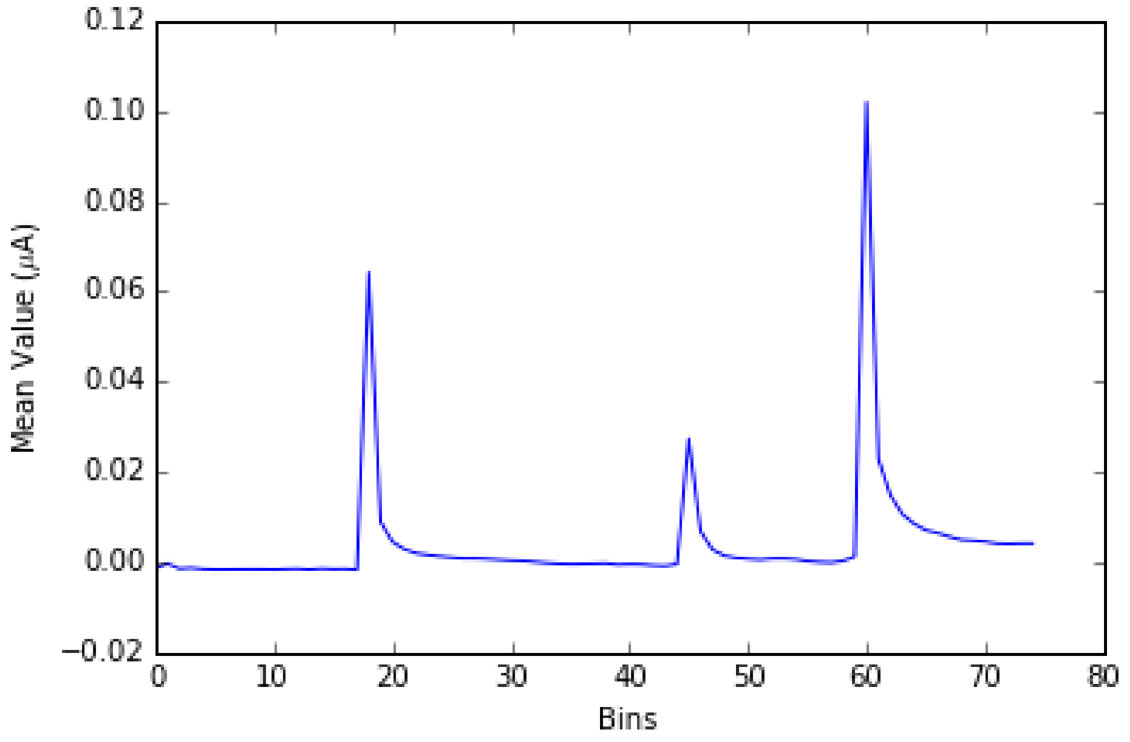
Re-definition of last() starting from first() , taking into consideration the time-lenght of a generic pulse.

• Uncompleted Pulses

As the algorithm is today, it is not able to detect all the uncompleted pulses → Bad data flagged as good data

This study should be implemented to be sure all the pulses that we have start and finish within the acquisition frame.

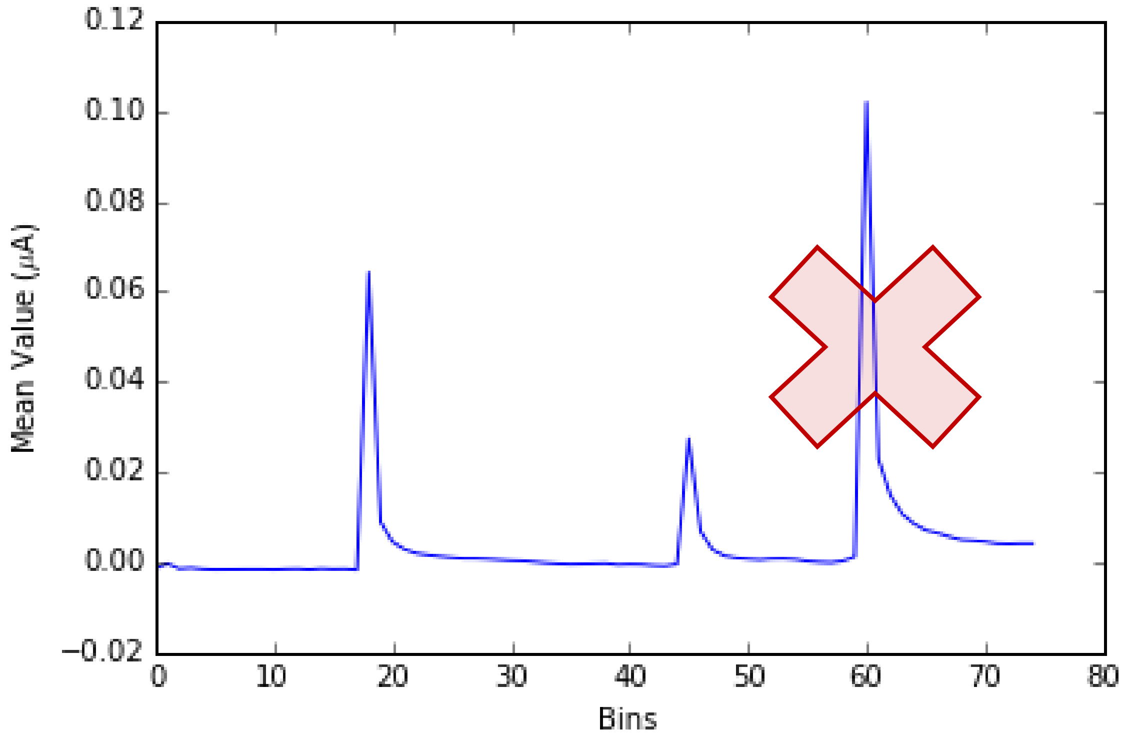
• Split Good Traces



Now, if a pulse is contaminated or if it is not finished, we throw the entire event. But this is NOT what we want at the end! If, for example, three pulses are present in a event and two of them are contaminated, we can still save the third one, and study that one.

→ We have a lot of pileup, what we want is to either reject a single pileup or split good traces into independent events if separated by some minimum distance.

• Split Good Traces



Now, if a pulse is contaminated or if it is not finished, we throw the entire event. But this is NOT what we want at the end! If, for example, three pulses are present in a event and two of them are contaminated, we can still save the third one, and study that one.

→ We have a lot of pileup, want we want is to either reject a single pileup or split good traces into independent events if separated by some minimum distance.

THANK YOU !