Introduction to "Noise" Analysis

-- by M. Schimassek (martin.schimassek@ijclab.in2p3.fr), 31.05.2024 --

This is a brief introduction to the "noise" analysis shown in commissioning context throughout the last years. It will make use of the sd-data.

Data: link

Code: repository

References: GAP-2023-022, GAP-2023-015

Introduction

In this tutorial, we will explore the "noise" of the detectors as measured in the event data. There has been previous work pointing out 'noise bursts', i.e. limited time periods with elevated noise in the recorded, that we will try to reproduce here.

Code, Requirements, and Installation

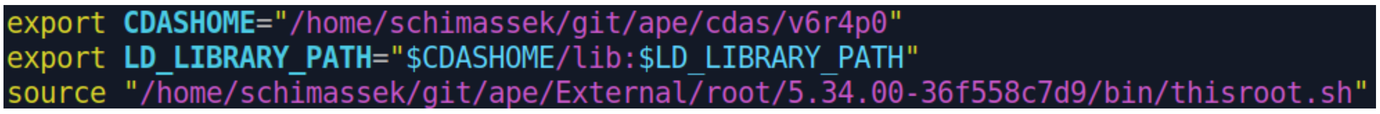
For this tutorial, we will use the "noise-analysis" repository that provides simple C++ programs that read the trace data and extract information about the first 300 bins of the traces. Please note that this is not a cleaned version for the purpose of this tutorial and as such contains old prototyping code that we will ignore.

To get the code (with an auger-gitlab account and ssh key), run

```
git clone --recurse-submodules git@gitlab.iap.kit.edu:auger-observatory/
sandboxes/schimassek-m/noise-analysis.git
```

As the programs read the sd/adsd files produced by the CDAS programs, we need a CDAS and corresponding root installation to compile the code.

You can have a look at the env.sh script to see how to set up your environment. The relevant lines are



```
export CDASHOME="/home/schimassek/git/ape/cdas/v6r4p0"
export LD_LIBRARY_PATH="$CDASHOME/lib:$LD_LIBRARY_PATH"
source "/home/schimassek/git/ape/External/root/5.34.00-36f558c7d9/bin/thisroot.sh"
```

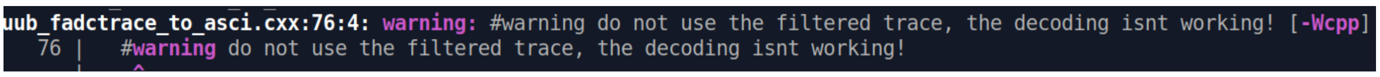
i.e. you need to set CDASHOME to be able to find includes and libraries from CDAS, set the LD_LIBRARY_PATH for linking and source the associated root-installation.

For instructions on how to install CDAS, we refer to ape and CDAS. A hidden requirement is the installation of boost for the command line options. On Ubuntu systems, you can install it on the system with

```
sudo apt-get install libboost-all-dev
```

or use the version you desire.

To compile the code with the environment set, simple type 'make'. On sufficiently new compilers, you will see the usual warnings associated to root-5 (TMatrixT). You can ignore these. There is a custom warning in one of the applications



```
uub_fadctrace_to_asci.cxx:76:4: warning: #warning do not use the filtered trace, the decoding isnt working! [-Wcpp]
   76 |   #warning do not use the filtered trace, the decoding isnt working!
      |    ^
```

which indicates a missing implementation that we do not use in this example.

Obtaining the Data

The data we will use in this tutorial is the trace data from event-data. Thus, we need the event data to extract the information we are interested in. You can use directly the sd-files as we don't need the merging information, however, it is also possible to use the merged files ad*.root or adsd*.root

You can get the data from irods through

```
iget /pauger/Malargue/Raid/data/Sd/2024/04/sd_2024_04_30_23h55.root
some_existing_folder/
```

or from the google-drive folder of this example. To extract the 'noise' information, we can then run

```
./uub_fadctrace_variance_analysis -i sd_2024_04_30_23h55.root -o
noise_2024_04_30_23h55
```

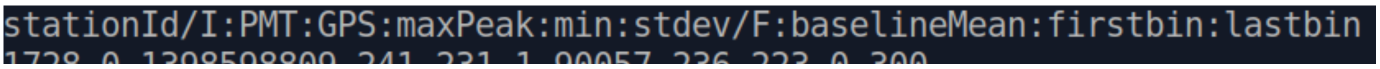
This should produce an output file called "noise_2024_04_30_23h55.out". You can find it here if you are unable to compile or run the code.

For a more comprehensive data set, we can use the first 10 days of May 2024, that data set you can find pre-prepared here.

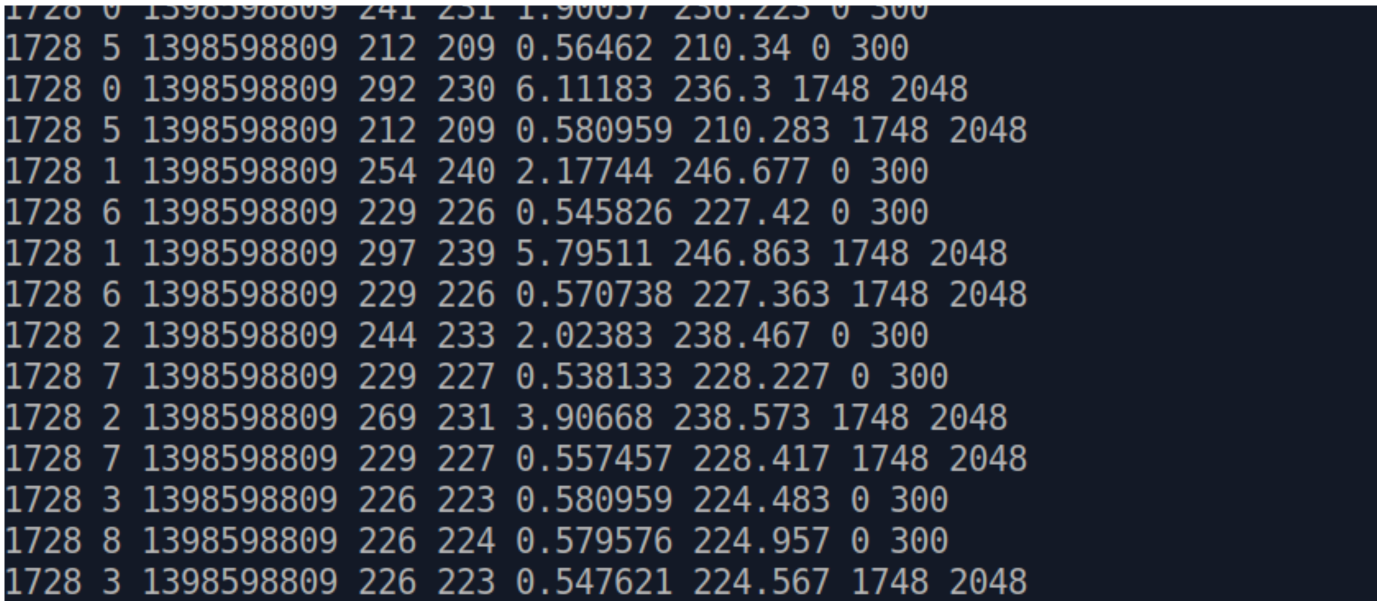
We also prepared a data set comprising all (non-special station) set for data from 2023 to April 2024 here.

Understanding the Extracted Data

The data we have extracted from the sd-files are stored in ASCII format and can be easily understood



```
stationId/I:PMT:GPS:maxPeak:min:stdev/F:baselineMean:firstbin:lastbin
1728 0 1308508800 241 231 1.00057 236 223 0 300
```



```
1728 0 1398598809 241 231 1.90037 236.225 0 300
1728 5 1398598809 212 209 0.56462 210.34 0 300
1728 0 1398598809 292 230 6.11183 236.3 1748 2048
1728 5 1398598809 212 209 0.580959 210.283 1748 2048
1728 1 1398598809 254 240 2.17744 246.677 0 300
1728 6 1398598809 229 226 0.545826 227.42 0 300
1728 1 1398598809 297 239 5.79511 246.863 1748 2048
1728 6 1398598809 229 226 0.570738 227.363 1748 2048
1728 2 1398598809 244 233 2.02383 238.467 0 300
1728 7 1398598809 229 227 0.538133 228.227 0 300
1728 2 1398598809 269 231 3.90668 238.573 1748 2048
1728 7 1398598809 229 227 0.557457 228.417 1748 2048
1728 3 1398598809 226 223 0.580959 224.483 0 300
1728 8 1398598809 226 224 0.579576 224.957 0 300
1728 3 1398598809 226 223 0.547621 224.567 1748 2048
```

You can see from the first line the "TTree" style column descriptor what the columns are. In detail, we have the station Id, which for the first entries the same, just like the GPS-second. This happens because we write one line into this file per PMT-trace and for each event we get 10 ADC-traces. In addition, we extract the noise in the first and last 300 bins. To distinguish the position in the trace, the last two columns can be used which indicate the bin interval used as [firstbin, lastbin).

The PMT-number is to be understood as follows: [0, 1, 2] are the LPMT, [3] is the sPMT, [4] is the SSD-PMT. Numbers larger than 4 are the LG-channels, e.g. [5, 6, 7] are the LG-data from the LPMTs.

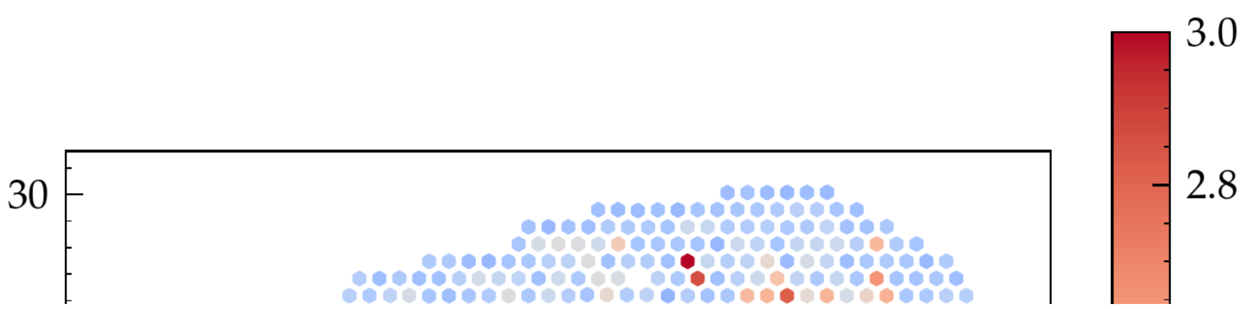
The information extracted per trace is summarized in 4 numbers: "maxPeak" which is the absolute maximal ADC value seen in the interval, "min" which is the minimal ADC value, "stdev" which is the standard deviation found for this interval, and "baselineMean" which is simply the average which can be used as indicator of the baseline.

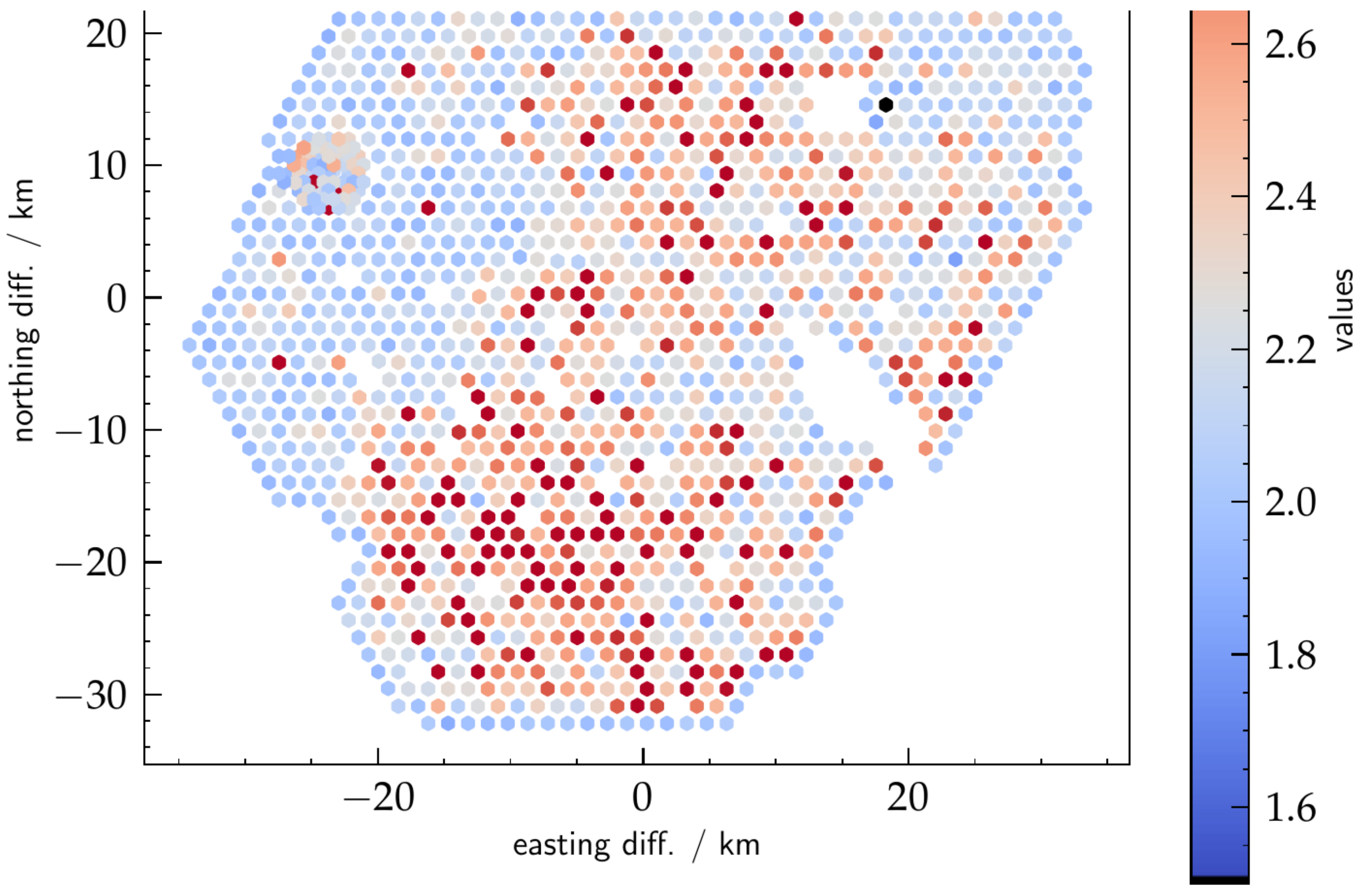
For analysis of the data, we refer to this jupyter-notebook.

For plotting for the station values extracted at the end of the notebook, you can download the python script from the google-drive or the given repository together with the station position file "AllStations.cfg". Then simply run

```
./plot_station_values.py -i ../moni-data/ssd_rms_max.dat -o ssd_noise --
column 1 --zmin 1.5 --zmax 3
```

You should find a plot similar to this:



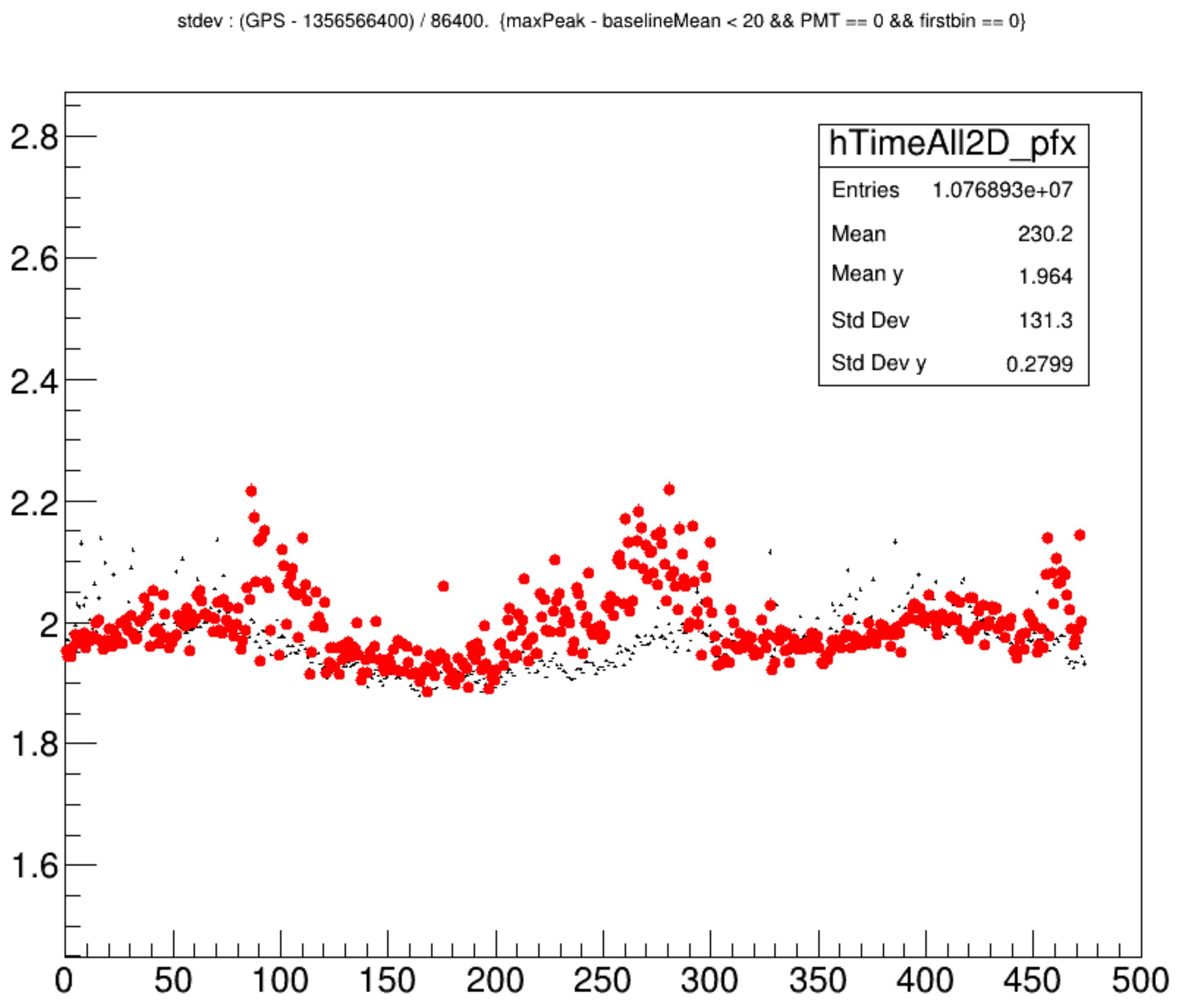


For more extensive analysis of the noise, you can also have a look at the README of the git-repository.

For the analysis of the year-long time series we use root, as pandas can be quite RAM-hungry for such large data sets. You can use a interactive root-session (we recommend root-6) for some quick analysis of the data, note that reading the GB of data can take several minutes in root (so in pandas it might be hours).

```
root [0] TTree t;
root [1] t.ReadFile("uub_noise_2023-2024-04_filtered.out")
(long long) 55427121
root [2] t.Draw("stdev : (GPS - 1356566400) / 86400. >> hTimeAll2D(500, 0, 500, 300, 1.0, 10.0)", "maxPeak - baselineMean < 20
&& PMT == 0 && firstbin == 0", "colz")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(long long) 10769040
root [3] new TCanvas();
root [4] t.Draw("stdev : (GPS - 1356566400) / 86400. >> hTime12UTC2D(500, 0, 500, 300, 1.0, 10.0)", "maxPeak - baselineMean < 2
0 && PMT == 0 && firstbin == 0 && abs(((GPS - 1356566400) % 86400) / 3600. - 12) < 2 ", "colz")
(long long) 1639878
root [5] new TCanvas();
root [6] pall = hTimeAll2D->ProfileX()
(TProfile *) @0x7ffdc8e1b678
root [7] p12 = hTime12UTC2D->ProfileX()
(TProfile *) @0x7ffdc8e1b678
root [8] p12->SetLineColor(kRed); p12->SetMarkerColor(kRed); p12->SetMarkerStyle(20)
root [9] pall->DrawClone()
(TObject *) 0x6145b39a9110
root [10] p12->DrawClone("PEsame")
(TObject *) 0x6145b3b59760
```

will lead you to an interesting plot of the noise as function of time in days after 01 Jan 2023 (the GPS second used in the command line):



To fully understand this plot, we have to remember that a shift in the red dots by 0.1 can be very significant as it averages a lot of times and all stations. If there is a tail (which you will see in the 2D histograms produced before this plot) it can have stronger implications than the average of 2.1 ADC suggests.