OCTOPUS

Giovanni Benato
July 24, 2024

What is Octopus?

- General framework for data processing
  - Started as small software to process Berkeley TES data
  - Expanded to process continuous data with any raw data format
  - Expanded to process triggered waveforms
- Use cases
  - Berkeley data
  - CROSS
  - BINGO
  - GAGG (Gio's neutron detector)
  - CUPID
  - DAREDEVIL

Main features

- Language: C++17
- Input data formats: Berkeley ROOT data, CUPID binary data, CAEN triggered data
  → Actual software decoupled from input format, any other format is acceptable
- Output format: ROOT
  → HDF5 and other formats can be considered
- GUI: not really, just plotting waveforms to tune the parameters of each algorithm
- Other dependencies: FFTW3
- Installation procedure: cmake

Where to find the code?

- At the moment, hosted in a private git repository of UC Berkeley
  - If you want access, send me your github.com username and you'll be satisfied
- In the coming N months, we are planning to:
  - License it
  - Make it open source
  - Write a technical paper

→ In this way we don't waste time making agreements between collaborations
→ Possible transition to collaboration tool to be discussed between the main authors

How do we talk to Octopus?

- User interface provided by TOML config files
- key=value approach
- Hierarchical (can define nested parameters)
- Available types: string, int, float, bool, datetime, array, table
  → Floats must have a digit after the dot (i.e. 1. is not valid, must specify 1.0)
- Non-valid keys are ignored → Octopus will throw a warning
- Non-valid values cause an error → Working on a better feedback from Octopus
- Spacing and indentation are ignored (useful only for readability)
- Comments start with #

TOML files in Octopus

```
[directories] # This is a section, or category
rawdir     = “/data/LSC/RUNS/DATA/RUN9”
triggerdir = “/home/benato/CROSS/Data/Triggered”

[runConfig]
filenamePrefix = "20230324T203458”
runNumber      = "000740"

[settings]
rawType   = "Cupid"       # This is a string
runType   = "reconstruct"
draw      = false         # This is a boolean
verbosity = 5             # This is an integer
```

TOML files in Octopus

```
[channels.heat] # This is a sub-category
list = [10,12]  # This is an array

[module.module.heat]
windowlength = [1.0,1.5] # Array of floats (one value per channel)
pretrigger   = [0.5]     # The value is copied over to all channels!
Draw         = false
Select = [ ["module","module","isnoise",true],                  # Array of
           ["module","numberoftriggers","numberoftriggers",1] ] # arrays!
```

Octopus workflow

4 sets of main programs:

1. **PlotDataStream**
   - Plots data stream of N channels in user-defined time window
   - Useful to look at waveforms and decide trigger parameters
2. **Trigger**
   - Runs threshold trigger on original or differentiated waveform
   - Writes ROOT file with sample index of triggered events
3. **Octopus**
   - Reads Raw data and trigger data in parallel
   - Reconstructs event-based quantities and writes them to ROOT file
4. **Multi-channel analysis** (not implemented yet)
   - Reads Octopus output
   - Reconstructs coincidences and any other multi-channel variable, writes ROOT file

Run on single channel!

Might merge into Octopus as a module

Channel handling

- User can define classes of channels (aka aliases)
- No hard-coded channel alias, no hard-coded alias names
- For each alias, must specify if the data are continuous or not
- Full analysis-chain defined on an alias base, but no trace of alias in output files
  → If you mess up the processing of one channel, defined a new alias exclusively for it, comment out the other aliases, and rerun Octopus just for that specific channel!

```
[channels.heat]
list = [10,12]
waveformtype = "continuous"

[channels.prisencolinensinainciusol]
list = [11]
waveformtype = "continuous"
```

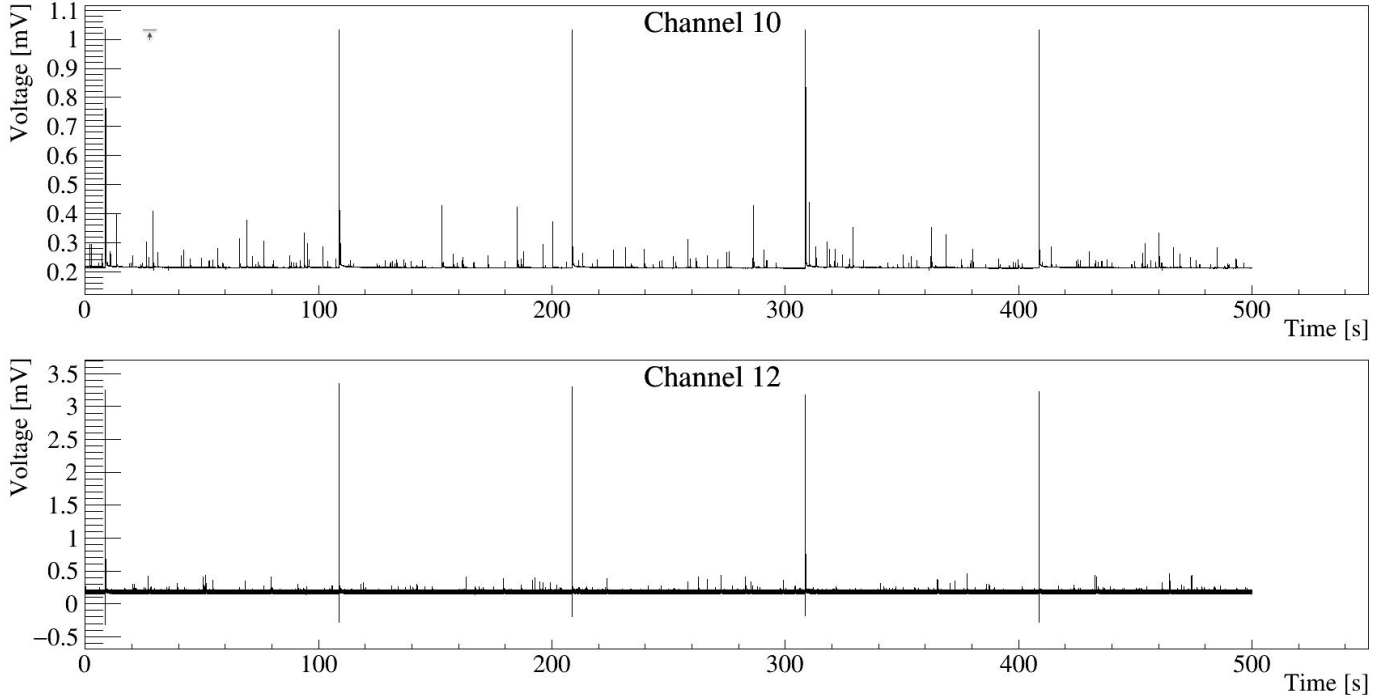
n

PlotDataStream

```
[directories]
rawdir       = "/home/gbenato/Cross/Data/OctopusData/Raw/"
triggerdir   = "/home/gbenato/Cross/Data/OctopusData/Triggered/Run000740/"
[runConfig]
filenamePrefix = "20230226T203003"
runNumber      = "000740"
[settings]
rawType     = "Cupid"
verbosity   = 5 # 1-->Error 2-->Warning ... 5-->Debug
startTime   = 0.0 # in seconds
stopTime    = 500.0 # in seconds
stride      = 1 # rebin (to make the plot faster for long periods)
[channels.heat]
list = [10,12]
waveformtype = "continuous"
```

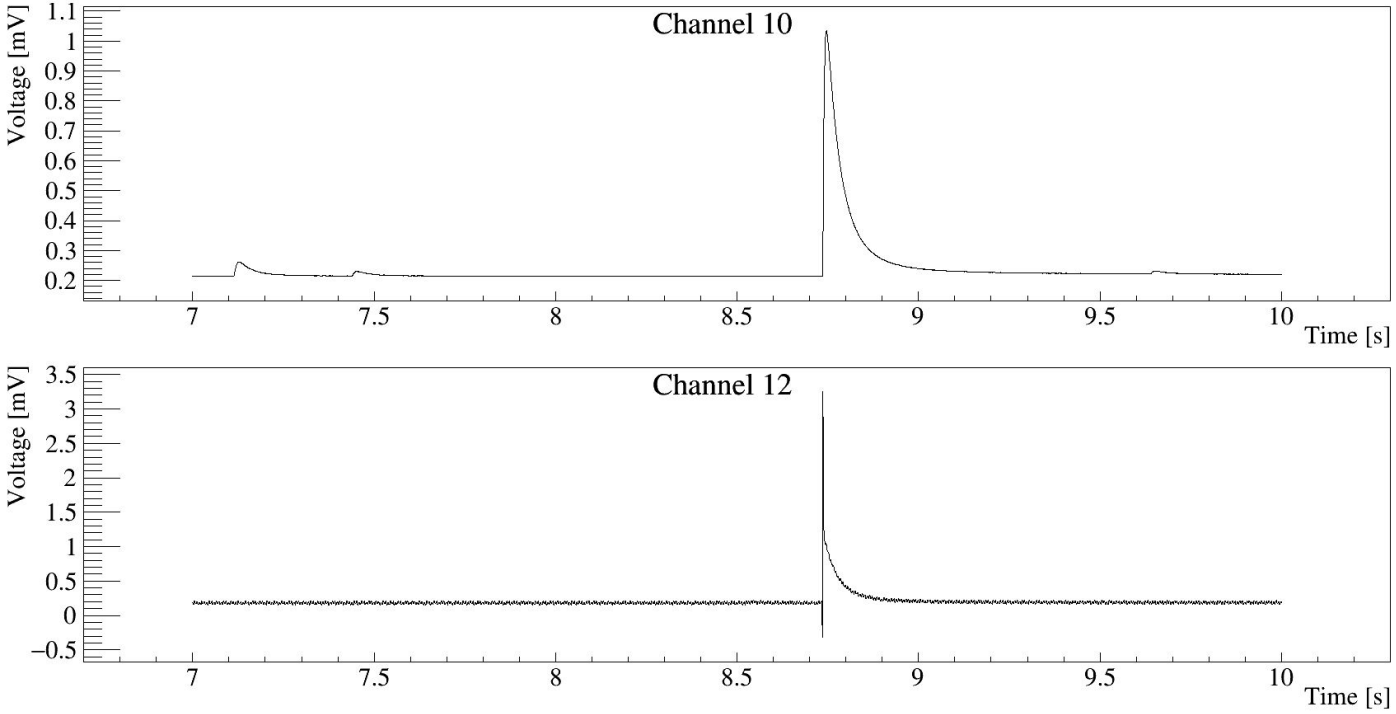
PlotDataStream

```
$ PlotDataStream /path/to/config/plotdatastream.toml
```



PlotDataStream

Same, but zoomed



Trigger

- Trigger logic:
  - Compute baseline and baseline RMS on running buffer
    $b_i = \sum_{k=i}^{k+N} x_i$
    $rms_i = \sum_{k=i}^{k+N} \sqrt{(x_i - b)^2/N}$
  - Check if the M samples right after the buffer are t·rms time higher than baseline:
    if $x_j > b_i + t \cdot rms_i \ \forall\ j \in \{i+N+1, i+N+3\} \Longrightarrow$ triggered event at sample j
  - Move buffer by 1 samples and repeat
- The same can be run on the original data stream, or on its derivative
- Fancier triggers on the to-do list
- Trigger speed: 3.5 days of data (2 kHz sampling) triggered in 50 sec
- CPU time does NOT depend on buffer size → If you touch that code I cut your fingers!

Trigger

```
[directories]
rawdir       = "/home/gbenato/Cross/Data/OctopusData/Raw/"
triggerdir   = "/home/gbenato/Cross/Data/OctopusData/Triggered/Run000740/"
processeddir = "/home/gbenato/Cross/Data/OctopusData/Reconstructed/Run000740/"
[runConfig]
filenamePrefix = "20230324T203458"
runNumber      = "000740"
[settings]
rawType = "Cupid"
runType = "trigger"
draw = false # Set to true to fine-tune the trigger parameters!
noiseTriggerPeriod = 60 # Save a noise event every 60 seconds
overwrite = true
verbosity = 5
...
```

Trigger

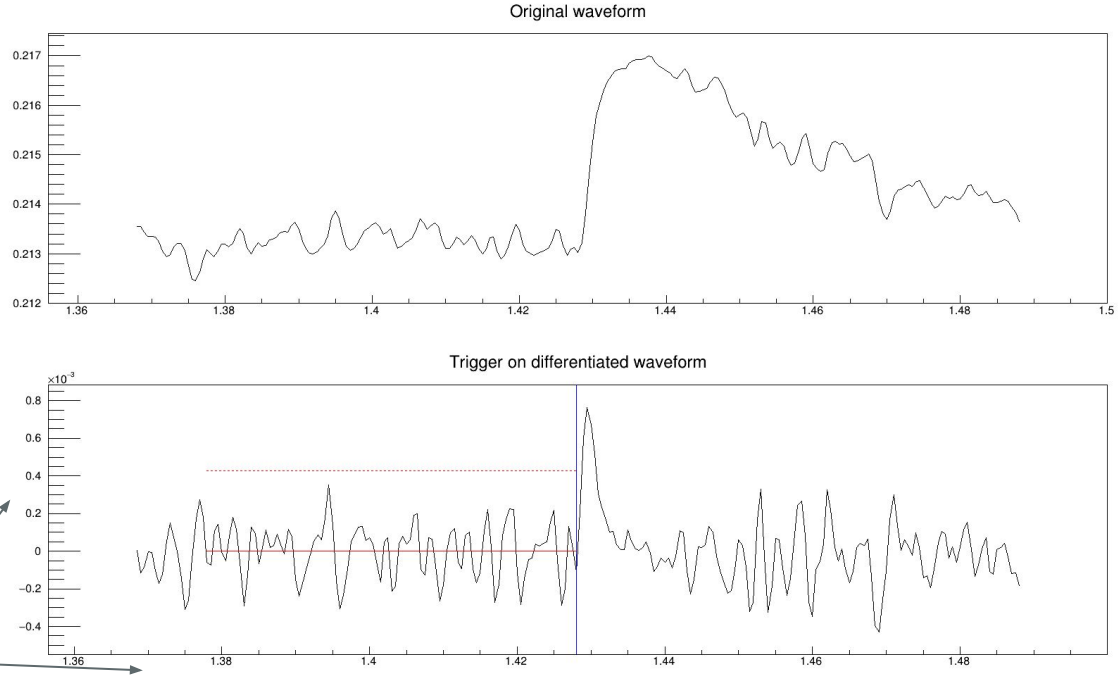
```
...

[channels.heat]
list = [10,12]
waveformtype = "continuous"

[trigger.heat]
rmsList = [2.0,3.0]  # Set threshold to baseline + rmsList[i]*RMS
nAboveList = [3]     # Requires 3 consecutive bins above threshold
bufferLength = [0.2] # Buffer length, in seconds
```

Trigger: some example

```
$ Trigger trigger.toml
```



Missing units!

Trigger: output format

```
root [0] TFile *f = new TFile("Trigger_Run_000740_Channel_10.root")
(TFile *) 0x558b060a0c60
root [1] f->ls()
TFile**         Trigger_Run_000740_Channel_10.root
 TFile*         Trigger_Run_000740_Channel_10.root
  KEY: TParameter<Long64_t>     StartTime_s;1
  KEY: TParameter<Long64_t>     StartTime_mus;1
  KEY: TParameter<double>       Livetime_s;1
  KEY: TParameter<double>       SamplingFreq_Hz;1
  KEY: TParameter<int>  NumberOfPulsers;1
  KEY: TParameter<bool> Derivative;1
  KEY: TParameter<Long64_t>     BufferSize;1
  KEY: TParameter<Long64_t>     nAbovePar;1
  KEY: TParameter<double>       threshold;1
  KEY: TVectorT<double> PhysicalChList;1
  KEY: TVectorT<double> PulserChList;1
  KEY: TVectorT<double> MuonChList;1
  KEY: TTree    trigger_tree;1  trigger_tree
  KEY: TTree    bad_interval_tree;1     bad_interval_tree
root [2] trigger_tree->Show(0)
======> EVENT:0
 TriggerSample   = 2055
 TriggerRMSRatio = 6.41389
 IsPulser        = 0
 NPulser         = -1
 IsMuon          = 0
 IsNoise         = 0
 IsSignal        = 1
root [3]
```

In the end, you will very rarely need to open this file!

Octopus (the executable, not the full software)

- Goal: reconstruct all event-based quantities
- Strategy:
  - Process each channel separately
    → Code will be easy to parallelize in the future
    → Can reprocess one channel without touching the others, if needed
  - Event reconstruction organized in “modules”
    → Each module does only one thing
    → Each module can access the output of previous modules
  - Smart logic to handle module order
    → Reads the module order from config file
    → Chains modules automatically, depending on their type to minimize access to raw data
    → Can place selection cuts on output of previous modules

Octopus: config file

```
[directories]
rawdir       = "/home/gbenato/Cross/Data/OctopusData/Raw/"
triggerdir   = "/home/gbenato/Cross/Data/OctopusData/Triggered/Run000740"
processeddir = "/home/gbenato/Cross/Data/OctopusData/Reconstructed/Run000740/"
[runConfig]
filenamePrefix = "20230324T203458"
runNumber      = "000740"
[settings]
rawType   = "Cupid"
runType   = "reconstruct"
draw      = false # General draw option
verbosity = 5
overwrite = true

...
```

Octopus: config file

```
...
[channels.heat]
list = [10]
waveformtype = "continuous"
[module.module.heat] # Retrieves waveform from raw data and writes some flags
windowlength = [1.0] # Default waveform length (overwritable by next modules)
pretrigger   = [0.5] # Baseline length (overwritable by next modules)
Draw         = false # Specific draw option of this module
# The following module will run for all channel aliases (“heat” not specified)
[module.timestamp] # Just writes the timestamp to the output file
Draw = false
[module.numberoftriggers.heat] # Counts triggers in window
...
```

Octopus: config file

- General options available to all modules:

```
Draw → bool
InputWaveform          → string
OutputWaveform         → string
Pretrigger             → array of floats
WindowLength           → array of floats
InheritFiltersFrom     → string
RunOnlyOnGoodEvents    → bool
Select, Cut            → Will explain later
CutOutside, CutInside  → Will explain later
CutAbove, CutBelow     → Will explain later
```

- Specific options available to each module
  → Module documentation available with executable **ModuleHelp**

Octopus: event filters (aka selection cuts)

```
[module.baselineslope.heat]
Select = [["module","module","isnoise",true], # Select only noise events
          ["module","numberoftriggers","numberoftriggers",1]]# Select events
                                                              # with just 1
                                                              # trigger
CutInside = [["module","baseline",[0.2,0.5],[0.3,0.6]],# 2 lists for min/max
              ["module","numberoftriggers",[0.0],[2.0]]]
CutInside = [["module","baseline",[0.2,0.5]]] # Here we need just one list

[module.risetime.heat]
InheritFrom = “module.baselineslope” # Import filters from previous module
```

Octopus: event filters

- What if we want to make a strict event selection and run the module only on events that pass the selection? E.g. what if we want to do the average pulse on 2615 keV events?

  ```
  RunOnlyOnGoodEvents = true
  ```

  This option will just run the module on events that pass the filters, and default the output variables to some predefined values otherwise

- What if we want to apply filters to events, but run the module on all events anyway?

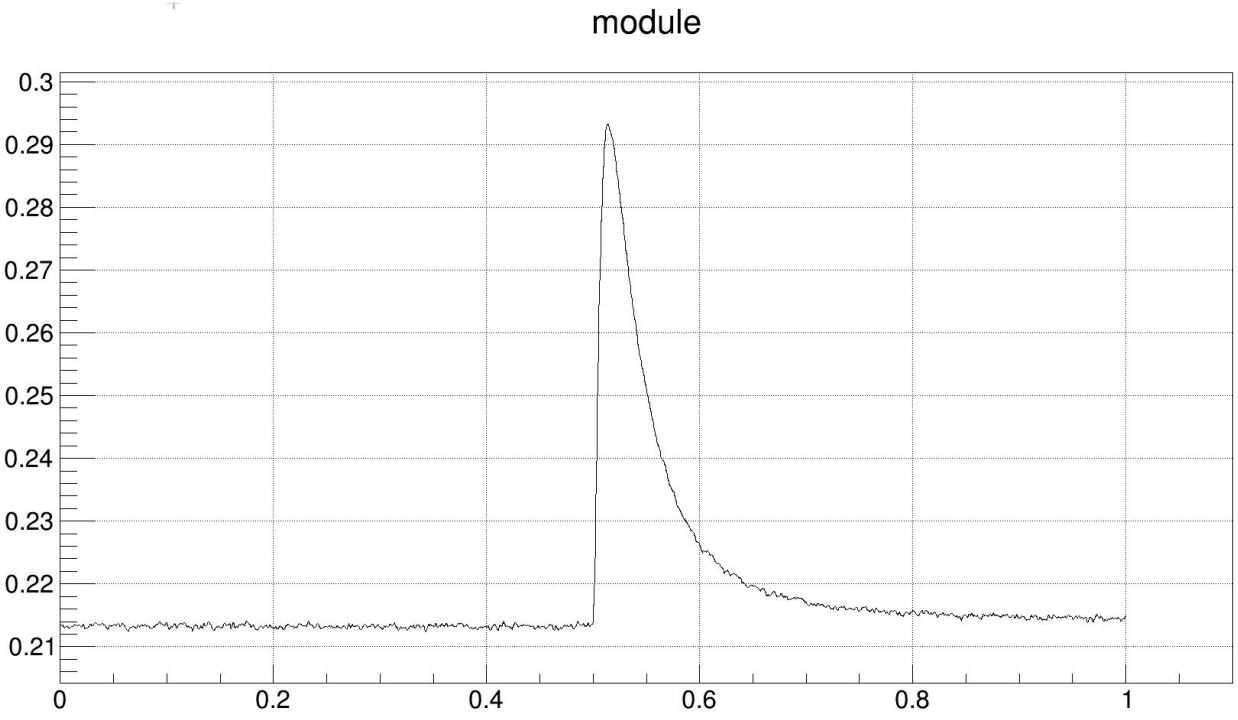
  ```
  RunOnlyOnGoodEvents = false
  ```

  This option runs the module over all events, and sets an output variable “good” to false in the output file

Calling modules multiple times

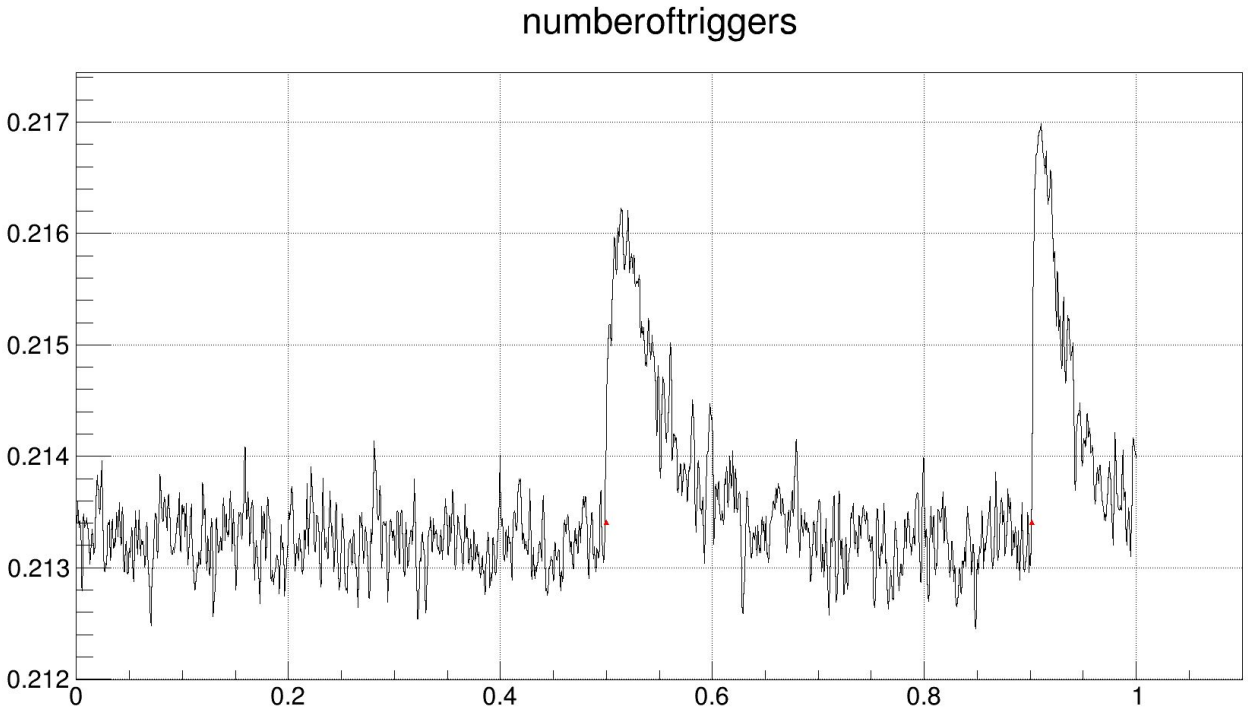
- Modules can be called multiple times, just adding an extra-label in the module name:
  `[module.averagepowerspectrum.heat.ap]`
  …
  `[module.averagepowerspectrum.heat.anps]`
- Extra-labels are not hard-coded anywhere, the user is free to use whatever naming
- Extra-labels are attached to the output variable names (TTrees, TH1Ds, …)
  → Please use self-explaining labels
  → If I find this in your output files I delete your cupid-login account:
  [module.averagepowerspectrum.heat.test1]
- Extra-labels can be used also if you call a module just once, if you really want to

Basic module



- Retrieves waveform from raw data
- Retrieves muon, pulser, noise and signal flags and stores them to output

Module numberoftriggers



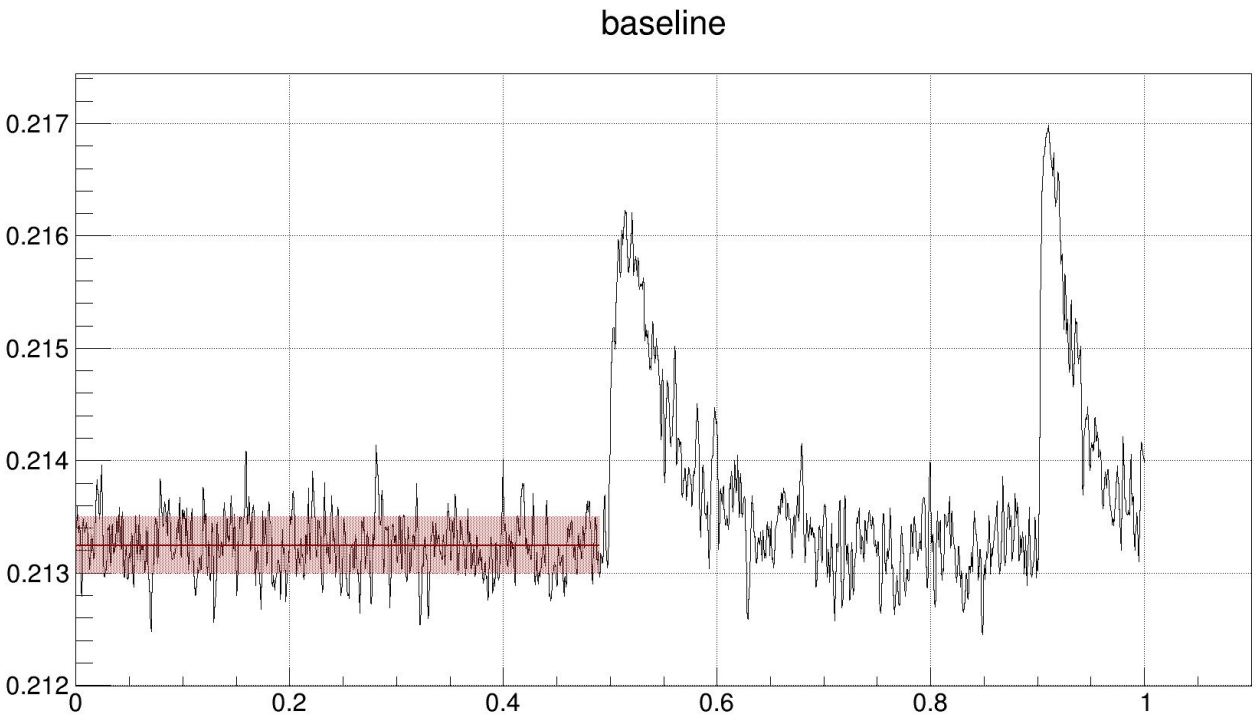
- Retrieves number of triggers in windows and stores it to output file

Module baseline

- Computes baseline and baseline RMS
- User-defined window for baseline calculation

```
[module.baseline.heat]
StartTime = [0.0]  # in seconds
StopTime  = [0.49] # in seconds
Draw = false
```

Module baseline



Modules logic

- Depending on the situation, modules can be run sequentially with other modules, or on their own
  a. If a module just needs the waveform and writes a per-event output (e.g. baseline)
     → Run sequentially
       → Loop over channels, loop over events, loop over modules
  b. If a module writes a “global” output (e.g. ANPS)
     → Run sequentially, but triggers and end-event-loop afterwards
  c. If a module needs the “global” output of a previous module (e.g. optimum filter)
     → Restart the loop over events
     → Might need to re-run some previous modules that are no more in memory
  d. If a module needs an internal loop over events (e.g. numberoftriggers)
     → Standalone run
       → Loop over channels, loop over modules, loop over events

Available modules

- Timestamp
- Number of triggers
- Baseline
- Baseline slope
- Baseline subtraction
- Max-minus-baseline
- Trigger delay + correction
- Risetime
- Decay time
- Pulse integral
- Fourier transform + inverse
- Integral of power spectrum
- Average power spectrum
- Average pulse
- Iterative average pulse
- Optimum filter
- Pole-zero correction
- Time-based convolution with any filter
- Stabilization
- Calibration
- Chi2 on time or freq domain
- Synthetic pulses

Modules to be developed in the future

- Truncated pulse fit
- Notch filter in complex space (to subtract noise that keeps the phase)
- Frequency-based trigger?
- …

Module documentation

- Module documentation available through ModuleHelp executable
- Documentation automatically detected from module structure, no hardcoded s**t!
- Interface to be improved

```
$ ModuleHelp
...
Debug: Available modules:
Debug   : Found module: averagepowerspectrum
Debug   : Found module: averagepulse
Debug   : Found module: baseline
Debug   : Found module: baselineslope
Debug   : Found module: baselinesubtraction
Debug   : Found module: calibration
...
```

Module documentation

```
Debug   : List of modules' options with corresponding default values:
Debug   : --------------------
Debug   : Module averagepowerspectrum.myalias
Debug   : Option Draw:  false
Debug   : Option InheritFiltersFrom:
Debug   : Option InputWaveform: original
Debug   : Option OutputWaveform:
Debug   : Option RunOnlyOnGoodEvents:   false
Debug   : Option pretrigger
Debug   : Ch    Value
Debug   : -1    0.500000
Debug   : Option windowlength
Debug   : Ch    Value
Debug   : -1    1.000000
...
```

Performance and bottlenecks

- Triggering of ~5 days of data, single channel, 2 kHz sampling frequency in ~1 min
- Full data production until OF in ~5 min
- Current bottlenecks (by educated guess, haven't used a profiler yet):
  - IO: need to read the raw data for each sequence of modules
    Solution: load full datastream in memory
  - Event filters repeated for each module
    Solution: reimplement the filter logic using e.g. event lists
  - Fourier transform re-allocating memory
    Solution: need to look into it in more detail

Limitations of current design

- Multi-channel analysis not feasible
  → E.g. denoising or decorrelation not possible

New big features to be included in the future

- Interface to some DB
  → Must be optional also at compile time
- Automated documentation (e.g. doxygen)
- Choice of output data format (e.g. ROOT vs HDF5)
- Python sandbox for development?