Waffles tutorial meeting
05/12/2024

DEEP UNDERGROUND NEUTRINO EXPERIMENT

WAFFLES
Waveform Analysis Framework For Light Emission Studies

Tutorial (Part 1)

A. Cervera
J. Ureña

(IFIC-Valencia)

Introduction

- Waffles is a software framework, written in python, whose purpose is to facilitate the PDS data quality assessment and PDS-only performance evaluation
- Waffles was initially developed for NP04 but with the goal of making it general enough such that it could be used for NP02 and the FDs
- The main developer is Julio Ureña (IFIC) but had contributions from many other people
- The git repository:

https://github.com/DUNE/waffles

Specific branch for this tutorial

```
git clone https://github.com/DUNE/waffles.git
cd waffles
git checkout tutorial_05122024
```

A collaborative effort

- Crucial for the success of this project
- This is realised as follows:
    1. Your analysis should be integrated into the main git branch ASAP
        - But please follow the the analysis structure and coding conventions
    2. No hard distinction between users and developers. Users should become developers:
        - The framework is quite light and can be ‘easily’ understood. If you need to do something that is not available, try to understand how to do it and commit it, such that others can use it
    3. Specific analyses are part of the framework such that newcomers can use existing analyses as reference
    4. Utilities developed for a specific analysis should be promoted to general utilities if they are useful for other analyzers

This tutorial

- We had planned a somehow more ambitious tutorial with data people could run over, event loops, beam information, etc, but didn't have the time to do all commits and implement them in the talk
- This talk will be updated with some more info before Christmas and an email will be sent
- A new tutorial with all this things fixed and more details about the waffles data models and functionality (e.g. plotting) will be given early next year

Table of contents

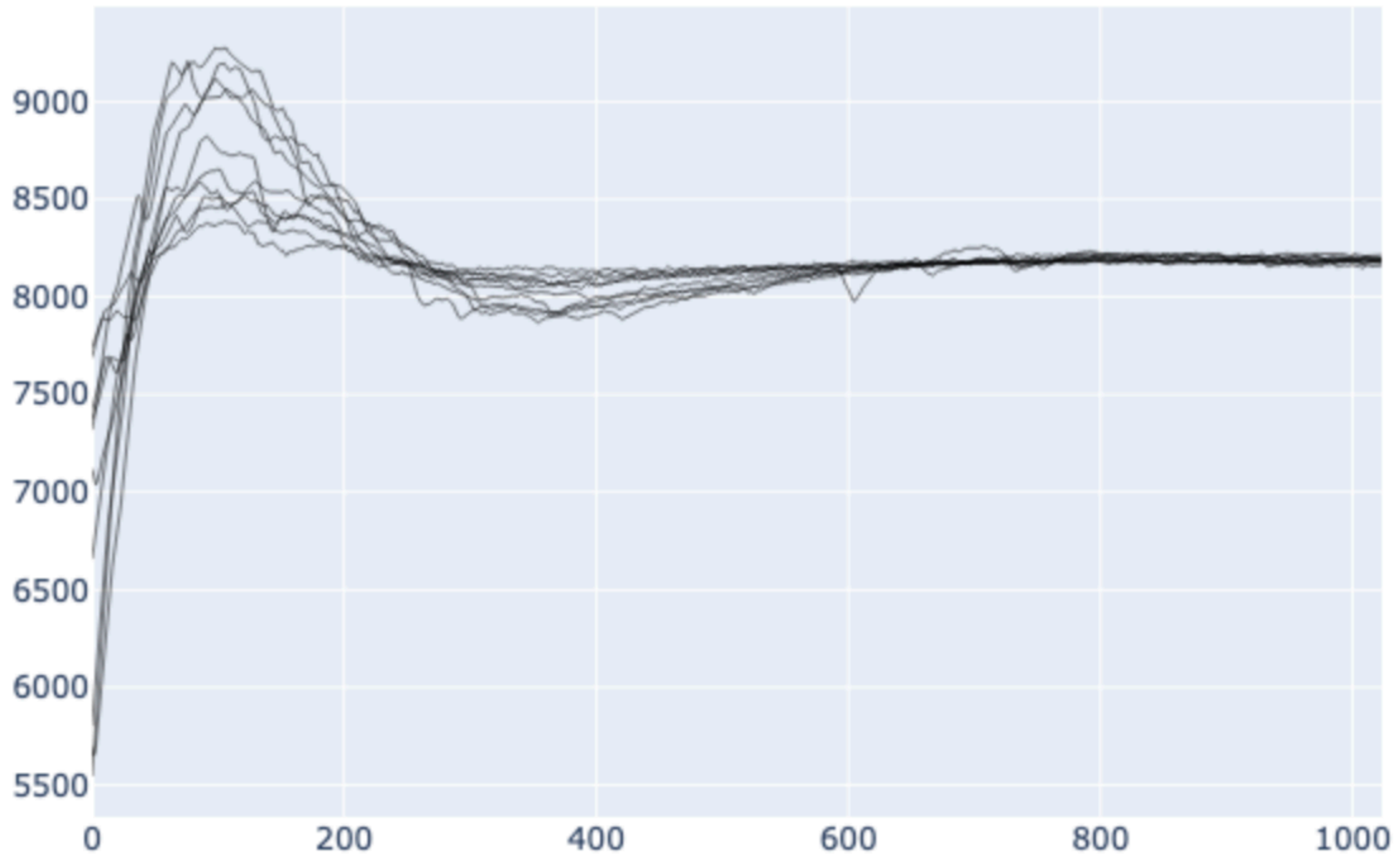
Definitions and Waffles data model

NP04 or ProtoDUNE-HD

DAPHNE readout electronics

Waveform

- Waffles works with the concept of Waveform



- Which has an array of adc values distributed in time ticks (16 ns width). NP04 has two times of waveforms:
  - **APA1**: full streaming mode
  - **APA2-4:** self-trigger mode. 1024 time ticks

Waffles objectives

- Read the raw data from the detector, in hdf5 format
  - Waveforms
- Convert that raw data to the WAFFLES format (python classes)
- Provide tools for managing those waveforms:
  - baseline, integration, amplitude, peak finding, denoising, deconvolution, filtering, selection, charge histogram, S/N, gain etc.
  - Plotting the previous results

Waffles data model

- The framework provides a series of python classes where the input data can be saved. For the moment these are the ones we should know about:
  - **Waveform**: array of adc values, time stamps and channel ID
  - **WaveformSet**: a collection of waveforms

Waffles structure

Folder structure

- These are the folders are inside waffles
  - **docs**: documentation and examples
  - **src**: the source code of the framework
  - **scripts**: python/bash scripts and c++ code, mainly related with hdf5 decoders

Folder structure in src/waffles

- This is the folder structure under waffles/src/waffles

core

data_classes

input_output

utils

plotting

np04_data_classes

np04_utils

np04_data

np04_analysis

np02_data_classes

np02_utils

np02_data

np02_analysis

NP02 folders
Do not exist yet

Loosing freedom

- In order to allow collaboration between analyzers it is crucial to have some strict rules to follow. We will have to renounce to some freedom in coding our analysis in order to gain in transparency. This is realised as follows:
  - The way of calling analyses will be always the same. From inside our analysis folder (a subfolder in np04_analysis)

```
python ../../core/main.py
```

  - The files and folders inside a analysis subfolder will be always the same:

```
Analysis1.py
Analysis2.py
…
utils.py
imports.py
params.yml
steering.yml
configs
output
scripts
data
```

  - Code should follow pep8 convention

Analysis structure

waffles/src/waffles/np04_analysis/my_analysis/

- These are the files and folders inside a specific analysis subfolder

`Analysis1.py` `Analysis2.py` …	This is the main analysis code. It could be run in several steps. The output of Analysis1.py would be input for Analysis2.py Should be very simple such that the analysis flow can be easily understood looking at the code
`steering.yml`	It contains the sequence in which AnalysisN.py are run
`utils.py`	Complex algorithms for this analysis should be here
`params.yml`	Configuration parameters (numbers, file paths, etc)
`imports.py`	All imports needed by AnalysisN.py should be here
`configs`	Folder with configuration parameters that do not change (often)
`output`	The analysis output should appear in this folder
`scripts`	bash, python, root macros, jupyter notebooks … NOT MANDATORY
`data`	Recommended folder for input data. NOT MANDATORY

The main program and the steering file

steering.yml

waffles/src/waffles/np04_analysis/led_calibration/streering.yml

- As mentioned above a specific analysis could be performed in several steps
- This file would be equivalent to a bash or python script calling the different AnalysisN.py files in the folder, but it is better to keep a unified way of doing that

LED_calibration

```yaml
1:
  name: "Analysis1"
  parameters: "params.yml"
  parameters_is_file: True
```

tau_slow_convolution

```yaml
# Step 1. Create the average waveform for the response
1:
  name: "Analysis1"
  parameters: "params.yml"
  parameters_is_file: True

# Step 2. Create the average waveform for the template
2:
  name: "Analysis1"
  parameters: "params_template.yml"    # To be restored whe
  parameters_is_file: True

# Step 3. Perform the convolution fit
3:
  name: "Analysis2"
  parameters: "params.yml"
  parameters_is_file: True
```

- By default steering.yml is used but you can use a different steering file, as explained in the next slide

The main program

waffles/src/waffles/core/main.py

- There are several ways of running a given analysis

```
python ../../core/main.py                           # use mandatory steering.yml and params.py

python ../../core/main.py -s alt_steering.py        # use a different steering file

python ../../core/main.py -p alt_params.yml         # use a different params file

python ../../core/main.py -a alt_Analysis           # use a different Analysis algorithm
```

- The user can also overwrite parameters in the params.yml file giving them as arguments. TO BE EXPLAINED IN A NEW VERSION OF THIS TALK

```
-s, --steering: str
    Name of the steering file.
-a, --analysis: str
    The name of the analysis class to be
    executed
-p, --params: str
    Name of the parameters file.
-v, --verbose: bool
    Whether to run with verbosity.
```

WafflesAnalysis base class

```
Analysis1.py
Analysis2.py
…
utils.py
imports.py
params.yml
steering.yml

configs/
output/
scripts/
data/
```

WafflesAnalysis

waffles/src/waffles/data_classes/WafflesAnalysis.py

- Files AnalysisN.py should contain a class AnalysisN, inheriting from WafflesAnalysis
- This class has 5 abstract methods, which must be implemented by the derived class

```
class WafflesAnalysis(ABC):
    """This abstract class implements a Waffles Analysis.
    It fixes a common interface and workflow for all
    Waffles analyses.

    Attributes
    ----------
    read_input_loop: list
        # Add description of this parameter
    analyze_loop: list
        # Add description of this parameter
    analyze_itr: list
        # Add description of this parameter
    read_input_itr: list
        # Add description of this parameter

    Methods
    ----------
    get_input_params_model():
        Class abstract method which is responsible for
        defining and returning a validation model for the
        input parameters of the analysis. This method must
        be implemented by each derived analysis class.
    initialize(input_parameters: BaseInputParams):
        Abstract method which is responsible for defining
        both, the common instance attributes (namely
        self.read_input_loop, self.analyze_loop,
        self.analyze_itr and self.read_input_itr) and
        whichever further attributes are required by the
        analysis. The defined attributes are potentially
        used by the read_input(), analyze() and write_output()
        methods.
    read_input():
        Abstract method which is responsible for reading
        the input data for the analysis, p.e. Waffles
        objects such as WaveformSet's. For more information,
        refer to its docstring.
    analyze():
        Abstract method which is responsible for performing
        the analysis on the input data. For more information,
        refer to its docstring.
    write_output():
        Abstract method which is responsible for writing
        the output of the analysis. For more information,
        refer to its docstring.
    """
```

Annotations (arrows pointing to the methods):

- Define command line arguments → `get_input_params_model()`
- Manage command line arguments and parameters. Do all operations prior to reading the input file → `initialize(input_parameters: BaseInputParams)`
- read one or several input files → `read_input()`
- Perform the actual analysis → `analyze()`
- Write the output → `write_output()`

LED_calibration

Introduction

- This analysis happens in two steps:
    1. Create the calibration histogram for all channels in one APA and compute the signal to noise and gain
    2. Create plots of S/N vs channel for different OV (PDS) or calibration batches
- But only the first one is adapted to the new framework structure. The second step will be added soon
- This is the steering file

```
1:
  name: "Analysis1"
  parameters: "params.yml"
  parameters_is_file: True
```

LED_calibration

waffles/src/waffles/np04_analysis/LED_calibration/Analysis1.py

- This is an example of Analysis1 for the led_calibration analysis

Import all necessary files and methods

The Analysis1 class, inheriting from WafflesAnalysis base class

```
from waffles.np04_analysis.LED_calibration.imports import *

class Analysis1(WafflesAnalysis):

    def __init__(self):
        pass
```

Analysis1: parameters

waffles/src/waffles/np04_analysis/LED_calibration/Analysis1.py

- These are all the parameters that will appear in the params.yml file or could be overwritten in the command line

```python
class Analysis1(WafflesAnalysis):

    def __init__(self):
        pass

    @classmethod
    def get_input_params_model(
        cls
    ) -> type:
        """Implements the WafflesAnalysis.get_input_params_model()
        abstract method. Returns the InputParams class, which is a
        Pydantic model class that defines the input parameters for
        this analysis.

        Returns
        -------
        type
            The InputParams class, which is a Pydantic model class"""

        class InputParams(BaseInputParams):
            """Validation model for the input parameters of the LED
            calibration analysis."""

            apa: int = Field(
                ...,
                description="APA number",
                example=2
            )

            pde: float = Field(
                ...,
                description="Photon detection efficiency",
                example=0.4
            )

            batch: int = Field(
                ...,
                description="Calibration batch number",
                example=2
            )

              ⋮

            plots_saving_folderpath: str = Field(
                default="./",
                description="Path to the folder where "
                "the plots will be saved."
            )

        return InputParams
```

Analysis1: initialize

waffles/src/waffles/np04_analysis/LED_calibration/Analysis1.py

- Parameters created in the previous slide are passed to **initialize** method and saved in a argument **self.params**

- In this method we do everything it can be done before reading the input file(s)

```python
def initialize(
    self,
    input_parameters: BaseInputParams
) -> None:
    """Implements the WafflesAnalysis.initialize() abstract
    method. It defines the attributes of the Analysis1 class.

    Parameters
    ----------
    input_parameters : BaseInputParams
        The input parameters for this analysis

    Returns
    -------
    None
    """

    self.read_input_loop = [None,]
    self.analyze_loop = [None,]
    self.params = input_parameters
    self.wfset = None
    self.output_data = None
```

Analysis1: read_input

waffles/src/waffles/np04_analysis/LED_calibration/Analysis1.py

- One of several files are read here
- In this case waveforms from several files are read and saved into a WaveformSet
- This class will be discussed later, but it is basically a smart collection of waveforms
- The method returns True if reading was successful

```python
def read_input(self) -> bool:
      ⋮

    # get all runs for a given calibration batch, apa and PDE value
    runs = run_to_config[self.params.batch][self.params.apa][self.params.pde]

    # Loop over runs
    for run in runs.keys():
        channels_and_endpoints = config_to_channels[self.params.batch][self.params.apa][self.params.pde][runs[run]]

        # Loop over endpoints using the current run for calibration
        for endpoint in channels_and_endpoints.keys():

            # List of channels in that endpoint using that run for calibration
            channels = channels_and_endpoints[endpoint]

            print("\n Now loading waveforms from:")
            print(f" - run {run}")
            print(f" - endpoint {endpoint}")
            print(f" - channels {channels} \n")

            # Get the filepath to the input data for this run
            input_filepath = led_utils.get_input_filepath(
                self.params.input_path,
                self.params.batch,
                self.params.apa,
                self.params.pde,
                run
            )

            # Read all files for the given run
            new_wfset = led_utils.read_data(
                input_filepath,
                self.params.batch,
                self.params.apa,
                is_folder=False,
                stop_fraction=1.,
            )

            # Keep only the waveforms coming from
            # the targeted channels for this run
            new_wfset = new_wfset.from_filtered_WaveformSet(
                new_wfset,
                led_utils.comes_from_channel,
                endpoint,
                channels
            )

            if first:
                self.wfset = new_wfset
                first=False
            else:
                self.wfset.merge(new_wfset)

    return True
```

Analysis1: analyze

waffles/src/waffles/np04_analysis/LED_calibration/analysis_1.py

1

```python
def analyze(self) -> bool:
    """
    Implements the WafflesAnalysis.analyze() abstract method.
    It performs the analysis of the waveforms contained in the
    self.wfset attribute, which consists of the following steps:

    1. Analyze the waveforms in the WaveformSet by computing
    their baseline and integral.
    2. Create a grid of WaveformSets, so that their are ordered
    according to the APA ordering, and all of the waveforms in a
    WaveformSet come from the same channel.
    3. Compute the charge histogram for each channel in the grid
    4. Fit peaks of each charge histogram
    5. Plot charge histograms
    6. Compute gain and S/N for every channel.

    Returns
    -------
    bool
        True if the method ends execution normally
    """

    # ------------- Analyse the waveform set -------------

    # get parameters input for the analysis of the waveforms
    analysis_params = led_utils.get_analysis_params(
        self.params.apa,
        # Will fail when APA 1 is analyzed
        run=None
    )

    checks_kwargs = IPDict()
    checks_kwargs['points_no'] = self.wfset.points_per_wf

    aux = 'standard'

    # Analyze all of the waveforms in this WaveformSet:
    # compute baseline, integral and amplitud
    _ = self.wfset.analyse(
        aux,
        BasicWfAna,
        analysis_params,
        *[],  # *args,
        analysis_kwargs={},
        checks_kwargs=checks_kwargs,
        overwrite=True
    )
```

2

```python
# ------------- Compute charge histogram -------------

# Create a grid of WaveformSets for each channel in one
# APA, and compute the charge histogram for each channel
grid_apa = ChannelWsGrid(
    APA_map[self.params.apa],
    self.wfset,
    compute_calib_histo=True,
    bins_number=led_utils.get_nbins_for_charge_histo(
        self.params.pde,
        self.params.apa
    ),
    domain=np.array((-10000.0, 50000.0)),
    variable="integral",
    analysis_label=aux
)

# ------------- Fit peaks of charge histogram -------------

# Fit peaks of each charge histogram
fit_peaks_of_ChannelWsGrid(
    grid_apa,
    self.params.max_peaks,
    self.params.prominence,
    self.params.half_points_to_fit,
    self.params.initial_percentage,
    self.params.percentage_step
)
```

3

```python
# ------------- Plot charge histograms -------------

figure = plot_ChannelWsGrid(
    grid_apa,
    figure=None,
    share_x_scale=False,
    share_y_scale=False,
    mode="calibration",
    wfs_per_axes=None,
    analysis_label=aux,
    plot_peaks_fits=True,
    detailed_label=False,
    verbose=True
)

title = f"APA {self.params.apa} - Runs {list(self.wfset.runs)}"

figure.update_layout(
    title={
        "text": title,
        "font": {"size": 24}
    },
    width=1100,
    height=1200,
    showlegend=True
)

if self.params.show_figures:
    figure.show()

figure.write_image(
    f"{self.params.plots_saving_folderpath}"
    f"/apa_{self.params.apa}_calibration_histograms.png"
)

""" ------------- Compute gain and S/N ------------- """

# Compute gain and S/N for every channel
self.output_data = led_utils.get_gain_and_snr(
    grid_apa,
    excluded_channels[self.params.batch][self.params.apa][self.params.pde]
)

return True
```

IMPORTANT:
Analize all waveforms in the WS, (see slide 56)

TODO: move to write_output

Analysis1: write_output

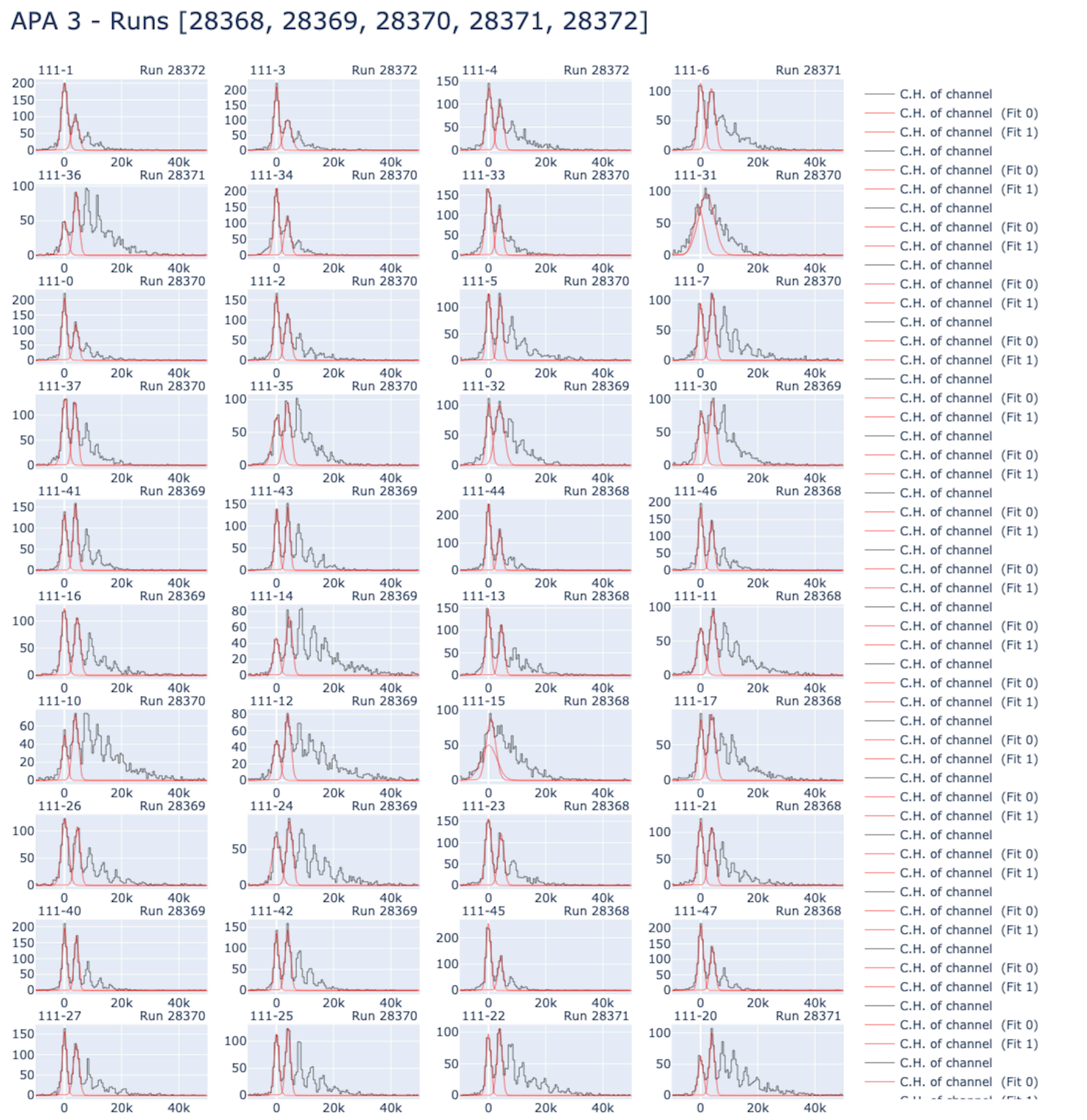
waffles/src/waffles/np04_analysis/LED_calibration/Analysis1.py

- The output file(s) could be in principle anything
  - plot(s) in .png file(s)
  - Many plots in a .pdf file
  - A collection of waveforms in a pickle file
  - A dataframe in a pickle file
  - etc
- Some of those files could be input to the next analysis step
- We are working in a standard way for presenting analysis results

```python
def write_output(self) -> bool:
    """Implements the WafflesAnalysis.write_output() abstract
    method. It saves the results of the analysis to a dataframe,
    which is written to a pickle file.

    Returns
    -------
    bool
        True if the method ends execution
    """

    # ------------- Save results to a dataframe ------------

    led_utils.save_data_to_dataframe(
        self,
        self.output_data,
        self.params.output_path,
    )

    return True
```

This is the output

- output/df.pkl. with gain and S/N for all 40 channels
- output/apa_3_calibration_histograms.png



Other files and folders in the analysis folder

```
Analysis1.py
Analysis2.py
…
utils.py
imports.py
params.yml
steering.yml

configs/
output/
scripts/
data/
```

imports.py

waffles/src/waffles/np04_analysis/LED_calibration/imports.py

- Not a very important file but helps in reducing the size of the AnalisisN.py files

```
import os
import plotly.subplots as psu
import numpy as np
import pandas as pd
import argparse
import pickle
import plotly.graph_objects as pgo
from pydantic import Field

from waffles.data_classes.Waveform import Waveform
from waffles.data_classes.WaveformSet import WaveformSet
from waffles.data_classes.ChannelWsGrid import ChannelWsGrid
from waffles.data_classes.IPDict import IPDict
from waffles.data_classes.BasicWfAna import BasicWfAna
from waffles.input.raw_root_reader import WaveformSet_from_root_files
from waffles.input.pickle_file_reader import WaveformSet_from_pickle_files
from waffles.utils.fit_peaks.fit_peaks import fit_peaks_of_ChannelWsGrid
from waffles.plotting.plot import plot_ChannelWsGrid
from waffles.np04_utils.utils import get_channel_iterator
from waffles.np04_analysis.LED_calibration.configs.calibration_batches.LED_configuration_to_channel import config_to_channels
from waffles.np04_analysis.LED_calibration.configs.calibration_batches.run_number_to_LED_configuration import run_to_config
from waffles.np04_analysis.LED_calibration.configs.calibration_batches.excluded_channels import excluded_channels
from waffles.np04_data.ProtoDUNE_HD_APA_maps import APA_map
from waffles.np04_analysis.LED_calibration import utils as led_utils
from waffles.data_classes.WafflesAnalysis import WafflesAnalysis, BaseInputParams
from waffles.np04_analysis.LED_calibration.configs.calibration_batches.metadata import metadata
```

params.yml

waffles/src/waffles/np04_analysis/LED_calibration/parms.py

- No hardcoded parameters (numbers, strings, etc) should appear in the code (AnalysisN.py and utils.py)
- All parameters should be in the params.yml file
- Those parameters can be overwritten by command line arguments (need to be defined in the mandatory get_input_params_model method)

1

```yaml
# -------- General parameters --------

# Path to the folder where the plots
plots_saving_folderpath: 'output/'

# APA number
apa: 3 # 1, 2, 3, 4

# Photon detection efficiency
pde: 0.45 # 0.40, 0.45, 0.50

# Calibration-batch number
batch: 3 # 1, ...

# Path to the folder containing the data
input_path: 'data'

# Path to the file where the
# output dataframe will be saved
output_path: 'output/df.pkl'

# PDE-to-OV mapping for HPK sipms
hpk_ov:
    0.4: 2.0
    0.45: 3.5
    0.50: 4.0

# PDE-to-OV mapping for FBK sipms
fbk_ov:
    0.4: 3.5
    0.45: 4.5
    0.50 : 7.0

# Enumeration of PDE values
ov_no:
    0.4: 1
    0.45: 2
    0.50: 3
```

2

```yaml
# ------- Parameters for charge histogram -------

analysis_label: 'standard'

# Lower integration limits. For APA 1 the limits are
# run-wise, while for APA 2, 3, 4 they are the same
# for all runs
starting_tick:
    1:
        27818: 621
        27820: 615
        27822: 615
        27823: 615
        27824: 615
        27825: 615
        27826: 615
        27827: 632
        27828: 626
        27898: 635
        27899: 635
        27900: 618
        27921: 602
        27901: 615
        27902: 615
        27903: 615
        27904: 630
        27905: 620
        27906: 610
        27907: 608
        27908: 602
    2:
        125
    3:
        125
    4:
        125

# Baseline limits
baseline_limits:
    1:
        - 100
        - 400
    2:
        - 0
        - 100
        - 900
        - 1000
    3:
        - 0
        - 100
        - 900
        - 1000
    4:
        - 0
        - 100
        - 900
        - 1000

# Integration window width, in time ticks
integ_window: 40
```

3

```yaml
# -------- Parameters for peak fitting --------

# Maximum number of peaks to fit
max_peaks: 2

# Minimal prominence, as a fraction
# of the y-range, for a peak to be detected
prominence: 0.15 # [0.10 - 0.2]

# The number of points to fit on either side
# of the peak maximum. P.e. setting this to
# 2 will fit 5 points in total: the maximum
# and 2 points on either side
half_points_to_fit: 2 # [2 - 3]

# Check the documentation of the
# 'initial_percentage' and 'percentage_step'
# parameters of the
# __spot_first_peaks_in_CalibrationHistogram
# function defined in
# utils/fit_peaks/fit_peaks_utils.py
initial_percentage: 0.15
percentage_step: 0.05
```

utils.py

waffles/src/waffles/np04_analysis/LED_calibration/utils.py

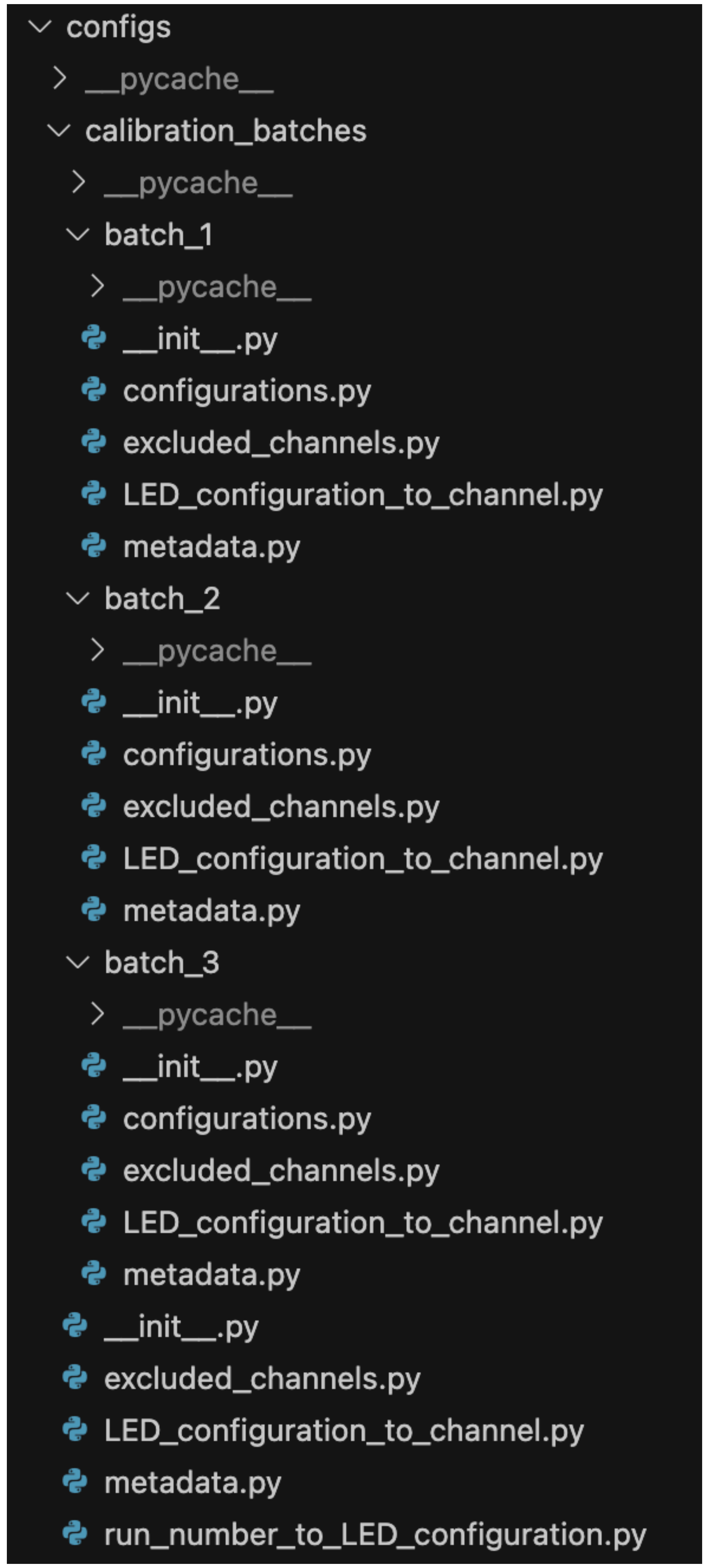
- As mentioned before, the code in AnalysisN.py should be such that the analysis flow can be easily understood by reading the code (+ comments)
- That means that heavy algorithmic should be in utils.py, keeping AnalysisN.py as small as possible
- On the left one of the methods in utils.py

```python
def get_gain_and_snr(
        grid_apa: ChannelWsGrid,
        excluded_channels: list
    ):

    data = {}

    for i in range(grid_apa.ch_map.rows):
        for j in range(grid_apa.ch_map.columns):

            endpoint = grid_apa.ch_map.data[i][j].endpoint
            channel = grid_apa.ch_map.data[i][j].channel

            if endpoint in excluded_channels.keys():
                if channel in excluded_channels[endpoint]:
                    print(f"Excluding channel {channel} from endpoint {endpoint}...")
                    continue

            try:
                fit_params = grid_apa.ch_wf_sets[endpoint][channel].calib_histo.gaussian_fits_parameter
            except KeyError:
                print(f"Endpoint {endpoint}, channel {channel} not found in data. Continuing...")
                continue

            # Handle a KeyError the first time we access a certain endpoint
            try:
                aux = data[endpoint]
            except KeyError:
                data[endpoint] = {}
                aux = data[endpoint]

            # compute the gain
            try:
                aux_gain = fit_params['mean'][1][0] - fit_params['mean'][0][0]
            except IndexError:
                print(f"Endpoint {endpoint}, channel {channel} not found in data. Continuing...")
                continue

            # this is to avoid a problem the first time ch is used
            try:
                aux_2 = aux[channel]
            except KeyError:
                aux[channel] = {}
                aux_2 = aux[channel]

            aux_2['gain'] = aux_gain

            # compute the signal to noise ratio
            aux_2['snr'] = aux_gain/np.sqrt(fit_params['std'][0][0]**2 + fit_params['std'][1][0]**2)

    return data
```

configs

waffles/src/waffles/np04_analysis/LED_calibration/configs/

- Configuration parameters that will not be changed frequently
- On the right the folder structure with configurations for this particular analysis
  - It basically tells you which run should be used for a calibration batch, pde (over-voltage) and channel
- We are working in a standardised way of doing this kind of things



tau_slow_convolution

Introduction

- This analysis happens in three steps:
    1. Create the average waveform for the run(s) and channel(s) with what to analyse
    2. Create the average waveform for the templates used for the run(s) and channel(s) with what to analyse
    3. Perform the convolution fit using the output from steps 1 and 2

```yaml
# Step 1. Create the average waveform for the response
1:
  name: "Analysis1"
  parameters: "params.yml"
  parameters_is_file: True

# Step 2. Create the average waveform for the template
2:
  name: "Analysis1"
  parameters: "params_template.yml"    # To be restored whe
  parameters_is_file: True

# Step 3. Perform the convolution fit
3:
  name: "Analysis2"
  parameters: "params.yml"
  parameters_is_file: True
```

fit WF_{avg}^{resp} to WF_{avg}^{temp} x model

Analysis1.py

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis1.py

- This is an example of Analysis1 for the tau_slow_convolution analysis

Import all necessary files and methods

The Analysis1 class, inheriting from WafflesAnalysis base class

```
# import all necessary files and classes
from waffles.np04_analysis.tau_slow_convolution.imports import *

class Analysis1(WafflesAnalysis):

    def __init__(self):
        pass
```

Analysis1: parameters

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis1.py

- These are all the parameters that will appear in the params.yml file or could be overwritten in the command line

```
###################################################################
@classmethod
def get_input_params_model(
    cls
) -> type:
    """Implements the WafflesAnalysis.get_input_params_model()
    abstract method. Returns the InputParams class, which is a
    Pydantic model class that defines the input parameters for
    this analysis.

    Returns
    -------
    type
        The InputParams class, which is a Pydantic model class"""

    class InputParams(BaseInputParams):
        """Validation model for the input parameters of the LED
        calibration analysis."""

        channels:                 list = Field(...,          description="work in progress")
        dry:                      bool = Field(default=False,description="work in progress")
        force:                    bool = Field(default=False,description="work in progress")
        response:                 bool = Field(default=False,description="work in progress")
        template:                 bool = Field(default=False,description="work in progress")
        runlist:                   str = Field(...,          description="work in progress")
        runs:                     list = Field(...,          description="work in progress")
        showp:                    bool = Field(default=False,description="work in progress")
        blacklist:                list = Field(...,          description="work in progress")
        baseline_threshold:      float = Field(...,          description="work in progress")
        baseline_wait:             int = Field(...,          description="work in progress")
        baseline_start:            int = Field(...,          description="work in progress")
        baseline_finish_template: int = Field(...,          description="work in progress")
        baseline_finish_response: int = Field(...,          description="work in progress")
        baseline_minimum_frac:   float = Field(...,          description="work in progress")

    return InputParams
```

Analysis1: initialize

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis1.py

1

```
##########################################################################
def initialize(
    self,
    input_parameters: BaseInputParams
) -> None:


    self.params = input_parameters

    self.endpoint = self.params.channels[0]//100

    self.safemode = True
    if self.params.force:
        self.safemode = False

    # make sure only -r or -t is chosen, not both
    if not self.params.response and not self.params.template:
        print("Please, choose one type --response or --template")
        exit(0)

    if self.params.response:
        self.selection_type='response'
    elif self.params.template:
        self.selection_type='template'

    # ReaderCSV is in np04_data
    dfcsv = ReaderCSV()
```

2

```
# these runs should be analyzed only on the last half
try:
    tmptype = 'Run'
    if self.params.template:
        tmptype = 'Run LED'
    runs = np.unique(dfcsv.dataframes[self.params.runlist][tmptype].to_numpy())
except Exception as error:
    print(error)
    print('Could not open the csv file...')
    exit(0)

if self.params.runs is not None:
    for r in self.params.runs:
        if r not in runs:
            print(f"Run {r} is not in database... check {self.params.runlist}_runs.csv")
    runs = [ r for r in runs if r in self.params.runs ]


self.baseliner = SBaseline()
# Setting up baseline parameters
self.baseliner.binsbase       = np.linspace(0,2**14-1,2**14)
self.baseliner.threshold      = self.params.baseline_threshold
self.baseliner.wait           = self.params.baseline_wait
self.baseliner.minimumfrac    = self.params.baseline_minimum_frac
self.baseliner.baselinestart  = self.params.baseline_start
self.baseliner.baselinefinish = self.params.baseline_finish_template
if self.selection_type=='response':
    self.baseliner.baselinefinish = self.params.baseline_finish_response


# read_input will be iterated over run numbers
self.read_input_loop = runs

# analyze will be iterated over channels
self.analyze_loop = self.params.channels
```

Configure the algorithm to compute the waveform baseline

Input loop iterates over runs ←

analyse loop iterates over channels ←

Analysis1: read_input

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis.py

Current iteration over runs

Read the pickle file and save it into a WaveformSet

```python
#########################################################################
def read_input(self) -> bool:

    # item for current iteration
    run = self.read_input_itr

    # this will be the input file name
    file = f"{self.params.input_path}/{self.endpoint}/wfset_run0{run}.pkl"

    if not os.path.isfile(file):
        print("No file for run", run, "endpoint", self.endpoint)
        return False
    if self.params.dry:
        print(run, file)
        return False

    # read all waveforms fro the pickle file
    self.wfset = 0
    try:
        self.wfset = WaveformSet_from_pickle_file(file)
    except Exception as error:
        print(error)
        print("Could not load the file... of run ", run, file)
        return False

    return True
```

Analysis1: analyze

waffles/src/waffles/np04_analysis/tau_slow_convolution/analysis_1.py

- Captures 1,2,3 are one after the other in the original file

1

```python
##################################################################
def analyze(self) -> bool:

    # items for current iteration
    run     = self.read_input_itr
    channel = self.analyze_itr

    # --------- perform the analysis for channel in run -----------
    #-------- This block should be moved to input when a double loop is available in read_input ------

    self.wfset_ch:WaveformSet = 0
    self.pickle_selec_name = f'{self.params.output_path}/{self.selection_type}s/{self.selection_type}_run0{run}_ch{channel}.pkl'
    self.pickle_avg_name   = f'{self.params.output_path}/{self.selection_type}s/{self.selection_type}_run0{run}_ch{channel}_avg.pkl'
    os.makedirs(f'{self.params.output_path}/{self.selection_type}s', exist_ok=True)

    if self.safemode and os.path.isfile(self.pickle_selec_name):
        val:str
        val = input('File already there... overwrite? (y/n)\n')
        val = val.lower()
        if val == "y" or val == "yes":
            pass
        else:
            return False
```

2

```python
# --------- perform waveform selection  -----------

# create an instance of the class with the sequence of cuts
extractor = Extractor(self.params,self.selection_type, run) #here because I changed the baseline down..

wch = channel
if (self.wfset.waveforms[0].channel).astype(np.int64) - 100 < 0: # the channel stored is the short one
    wch = int(str(channel)[3:])
    extractor.channel_correction = True

print ('#Waveforms: ', len(self.wfset.waveforms))

# select waveforms in the interesting channels
self.wfset_ch = WaveformSet.from_filtered_WaveformSet(self.wfset,
                                                    extractor.allow_certain_endpoints_channels,
                                                    [self.endpoint], [wch],
                                                    show_progress=self.params.showp)

print ('#Waveforms in channel: ', len(self.wfset_ch.waveforms))

try:
    self.wfset_ch = WaveformSet.from_filtered_WaveformSet(self.wfset_ch,
                                                        extractor.apply_cuts,
                                                        show_progress=self.params.showp)
except Exception as error:
    print(error)
    print(f"No waveforms for run {run}, channel {wch}")
    return False

print ('#Waveforms selected: ', len(self.wfset_ch.waveforms))
```

Analysis1: analyze

3

```
# --------- compute the baseline -----------

# Substract the baseline and invert the result
wf_arrays = np.array([(wf.adcs.astype(np.float32) - wf.baseline)*-1 for wf in self.wfset_ch.waveforms if wf.channel == wch])

# special treatment for runs in the blacklist
if run in self.params.blacklist:
    print("Skipping first half...")
    skip = int(0.5*len(wf_arrays))
    wf_arrays = wf_arrays[skip:]

# compute the average waveform
avg_wf = np.mean(wf_arrays, axis=0)

# Create an array with 500 numbers from -20 to 20
self.baseliner.binsbase = np.linspace(-20,20,500)

# compute the baseline again with a different method
res0, status = self.baseliner.compute_baseline(avg_wf)

# --------- compute final average waveform ----------

# subtract the baseline
avg_wf -= res0

# save the results into the WaveformSet
self.wfset_ch.avg_wf = avg_wf
self.wfset_ch.nselected = len(wf_arrays)

print(f'{run} total: {len(self.wfset.waveforms)}\t {channel}: {len(wf_arrays)}')

return True
```

Analysis1: write_output

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis1.py

pickle file with all waveforms contributing to the average waveform

Pick file with the average waveform

```python
####################################################################
def write_output(self) -> bool:

    # save all the waveforms contributing to the average waveform
    with open(self.pickle_selec_name, "wb") as f:
        pickle.dump(self.wfset_ch, f)

    # save the average waveform, the time stamp of the first waveform and the number of selected waveforms
    output = np.array([self.wfset_ch.avg_wf, self.wfset_ch.waveforms[0].timestamp, self.wfset_ch.nselected], dtype=object)

    with open(self.pickle_avg_name, "wb") as f:
        pickle.dump(output, f)
    print('Saved... ')

    return True
```

Analysis2.py

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

Import all necessary files and methods

The Analysis1 class, inheriting from WafflesAnalysis base class

```python
# import all necessary files and classes
from waffles.np04_analysis.tau_slow_convolution.imports import *

class Analysis2(WafflesAnalysis):

    def __init__(self):
        pass
```

Analysis2: parameters

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

```
#######################################################################
@classmethod
def get_input_params_model(
    cls
) -> type:
    """Implements the WafflesAnalysis.get_input_params_model()
    abstract method. Returns the InputParams class, which is a
    Pydantic model class that defines the input parameters for
    this analysis.

    Returns
    -------
    type
        The InputParams class, which is a Pydantic model class"""

    class InputParams(BaseInputParams):

        runs:         list = Field(...,          description="work in progress")
        channels:     list = Field(...,          description="work in progress")
        fix_template: bool = Field(...,          description="work in progress")
        the_template: int  = Field(...,          description="work in progress")
        namespace:    str  = Field(...,          description="work in progress")
        runlist:      str  = Field(...,          description="work in progress")
        print:        bool = Field(...,          description="work in progress")
        interpolate:  bool = Field(...,          description="work in progress")
        no_save:      bool = Field(...,          description="work in progress")
        scan:         int  = Field(...,          description="work in progress")

    return InputParams
```

Analysis2: initialize

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

```
########################################################################
def initialize(
    self,
    input_parameters: BaseInputParams
) -> None:

    self.params = input_parameters

    if self.params.runs is None:
        print('Please give a run')
        exit(0)

    runs = [ r for r in self.params.runs ]

    dfcsv = ReaderCSV()
    df = dfcsv.dataframes[self.params.runlist]
    runs2    = df['Run'].to_numpy()
    led_runs = df['Run LED'].to_numpy()

    self.run_pairs = { r:lr for r, lr in zip(runs2, led_runs) }

    # use a fix template
    self.led_run_template = self.params.the_template

    # results subfolder
    self.output_subfolder="results"
    if self.params.runlist != "purity":
        self.output_subfolder += f"_{self.params.runlist}"
    if self.params.namespace != "":
        self.output_subfolder += f"_{self.params.namespace}"
    if self.params.fix_template:
        self.output_subfolder += "_fixtemplate"

    # create the Convolution Fitter
    self.cfit = ConvFitter(threshold_align_template = 0.27,
                           threshold_align_response = 0.1,
                           error=10, usemplhep=True,
                           dointerpolation=self.params.interpolate,
                           interpolation_fraction = 8,
                           align_waveforms = True)

    if self.params.scan > 0:
        self.cfit.reduce_offset = True

    self.cfit.dosave = not self.params.no_save

    # loop over runs
    self.read_input_loop = runs

    # loop over channels
    self.analyze_loop = self.params.channels
```

Input loop iterates over runs

analyse loop iterates over channels

Analysis2: read_input

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

- WORK IN PROGRESS !!!
- Actually not reading anything since a double loop is currently not supported and we want a different file for each run and each channel
  - Temporarily the reading is moved to analize, where the double loop is accesible

```python
#################################################################
def read_input(self) -> bool:

    # items for current iteration (run number)
    self.run = self.read_input_itr

    print(f"Processing run {self.run}")
    if self.run not in self.run_pairs:
        print('Run not found in runlist, check it')
        exit(0)
    self.runled = self.run_pairs[self.run]

    # change template in the case it is fixed at 0 for endpoint 112
    if self.led_run_template == 0 and self.run > 27901:# and ch//100 == 112:
        self.led_run_template = 29177

    if self.params.fix_template:
        self.runled = self.led_run_template

    return True
```

Analysis2: analyze

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

```python
##################################################################
def analyze(self) -> bool:

    #-------- This block should be moved to input when a double loop is available in read_input ------

    # items for current iteration (channel number)
    self.channel = self.analyze_itr


    file_response = f"{self.params.output_path}/responses/response_run0{self.run}_ch{self.channel}_avg.pkl"
    file_template = f'{self.params.output_path}/templates/template_run0{self.runled}_ch{self.channel}_avg.pkl'

    if os.path.isfile(file_template) is not True:
            print(f'file {file_template} does not extst !!!')
            print(f"No match of LED run {self.runled}.. using \'the_template\' instead: {self.led_run_template} ")
            self.runled = self.led_run_template
            file_template = f'templates/template_run0{self.runled}_ch{self.channel}_avg.pkl'

    print ('file response: ', file_response)
    print ('file template: ', file_template)

    # read the average waveforms for the template and the response
    self.cfit.read_waveforms(file_template, file_response)

    #--------------------------------------------------

    # prepare the template and response waveforms for the current iteration (run number)
    # performs interpolation and time alignment between template and response waveforms
    self.cfit.prepare_waveforms()

    # perform the actual convolution fit
    self.cfit.fit(self.params.scan, self.params.print)

    return True
```

Analysis2: write_output

waffles/src/waffles/np04_analysis/tau_slow_convolution/Analysis2.py

text file containing data frame with fit values

text file fit results and con. matrix

The fit plot

```
##################################################################
def write_output(self) -> bool:

    #---------- do the convolution and fit plot -----------
    # do the plot
    plt = self.cfit.plot()

    #add legend to plot
    plt.legend(title=f'run {self.run}')

    # ---------- Save results and plot -----------

    dirout = f'{self.params.output_path}/{self.output_subfolder}/run0{self.run}'
    os.makedirs(dirout, exist_ok=True)

    nselected  = self.cfit.wf_response["nselected"]
    first_time = self.cfit.wf_response["firsttime"]

    with open(f"{dirout}/convolution_output_{self.run}_{self.runled}_ch{self.channel}.txt", "w") as fout:
        fout.write(f"{first_time} {self.cfit.m.values['fp']} {self.cfit.m.values['t1']}"
                   "{self.cfit.m.values['t3']} {self.cfit.m.fmin.reduced_chi2} {nselected} \n")

    with open(f"{dirout}/run_output_{self.run}_{self.runled}_ch{self.channel}.txt", "w") as fout:
        print(self.cfit.m, file=fout)

    # save the plot
    plt.savefig(f'{dirout}/convfit_data_{self.run}_template_{self.runled}_ch{self.channel}.png')

    return True
```

Analysis2 output

- output/results_new_analysis/run025171/convolution_output_25171_26081_ch11114.txt.txt

```
107155155107228308 0.23630732779375324 36.593279420086816 1558.5703976676164 9.01882632226919 5271
```

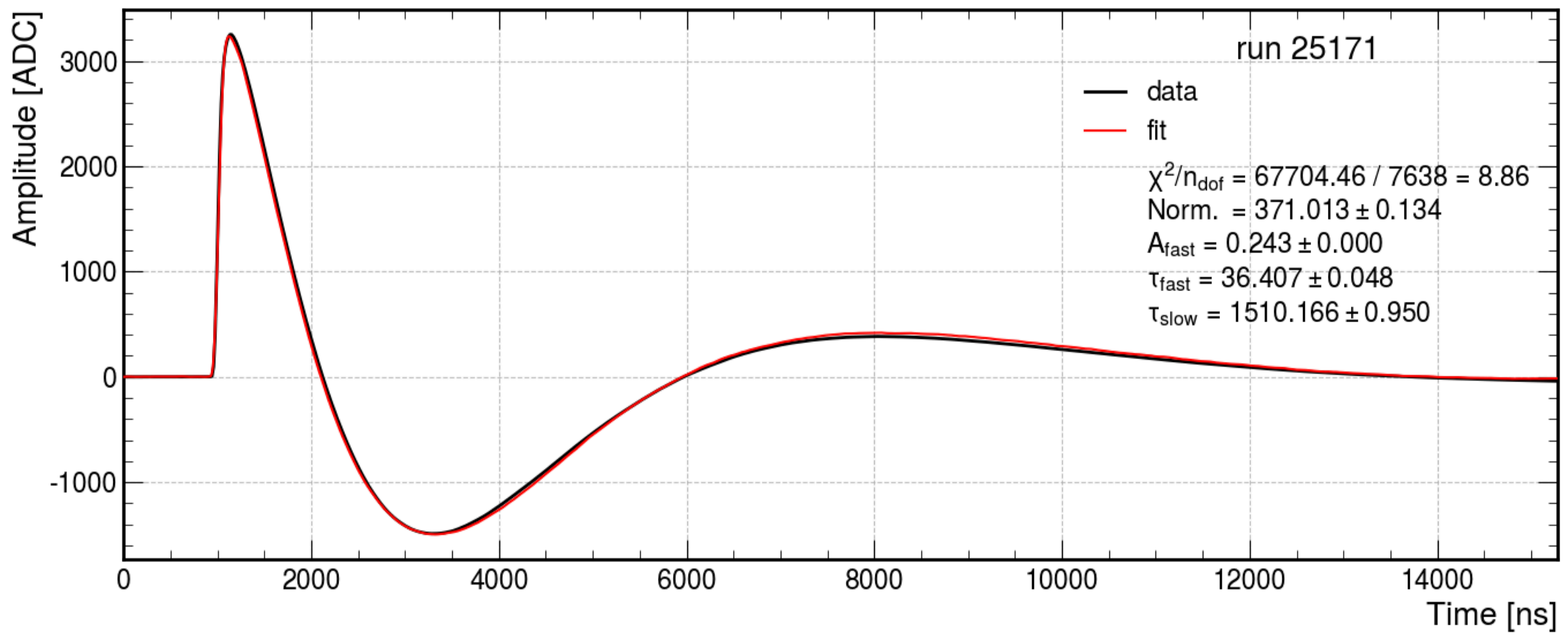
- output/results_new_analysis/run025171/run_output_25171_26081_ch11114.txt

Migrad	
FCN = 6.889e+04 (χ^2/ndof = 9.0) EDM = 7.23e-10 (Goal: 0.0002)	Nfcn = 627
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	A	360.59	0.14			0		
1	fp	236.31e-3	0.08e-3			0	1	
2	t1	36.59	0.07			2	50	
3	t3	1.5586e3	0.0011e3			500	2000	

	A	fp	t1	t3
A	0.0203	-2.254e-6	0.005	0.149
fp	-2.254e-6	6.06e-09	917e-9	-7.345e-6
t1	0.005	917e-9	0.00494	0.035
t3	0.149	-7.345e-6	0.035	1.29

- output/results_new_analysis/run025171/convfit_data_25171_template_26081_ch11114.png



Framework data classes

Folder structure in src/waffles

- This is the folder structure under waffles/src/waffles

core

data_classes

input_output

utils

plotting

np04_data_classes

np04_utils

np04_data

np04_analysis

NP02 folders
Do not exist yet

np02_data_classes

np02_utils

np02_data

np02_analysis

Folder structure in src/waffles

- **Bold orange names are folders**
- Bullets in white are files .py
- Sub-bullets in yellow are functions inside those files

data_classes

- BasicWfAna.py
- BeamWfAna.py
- CalibrationHistogram.py
- ChannelMap.py
- ChannelWs.py
- ChannelWsGrid.py
- Event.py
- IODict.py
- IPDict.py
- Map.py
- ORDict.py
- PeakFindingWfAna.py
- TrackedHistogram.py
- UniqueChannel.py
- WafflesAnalysis.py
- Waveform.py
- WaveformAdcs.py
- WaveformSet.py
- WfAna.py
- WfAnaResult.py
- WfPeak.py

input_output

- input_utils.py
- pickle_file_reader.py
- raw_hdf5_reader.py
- raw_root_reader.py
- persistence_utils.py

utils

- filtering_utils.py
- numerical_utils.py
- event_utils.py
- utils.py
- check_utils.py
- wf_maps_utils.py
- **deconvolution**
- **denoising**
- **baseline**
- **fit_peaks**

plotting

- plot.py
  - plot_wfs
  - plot_CalibHisto
  - plot_ChannelWSGrid
  - plot_WfAdcs
- plot_utils.py
- drawing_tools.py
- **display**

np04_data_classes

- APAMap.py
- …

np04_data

- ProtoDUNE_HD_APA_maps.py
- **tau_slow_runs**
  - beam_runs.csv
  - load_runs_csv.py
  - purity_runs.csv

np04_utils

- utils.py

np04_analysis

- **LED_calibration**
- **tau_slow_convolution**

Waveform and WaveformSet

waffles/src/waffles/data_classes/

WaveformSet

```python
# Getters
@property
def waveforms(self):
    return self.__waveforms

@property
def points_per_wf(self):
    return self.__points_per_wf

@property
def runs(self):
    return self.__runs

@property
def record_numbers(self):
    return self.__record_numbers

@property
def available_channels(self):
    return self.__available_channels

@property
def mean_adcs(self):
    return self.__mean_adcs

@property
def mean_adcs_idcs(self):
    return self.__mean_adcs_idcs
```

Smart collection of waveforms

Mention Analyse method de waveform set

Waveform

```python
# Getters
@property
def timestamp(self):
    return self.__timestamp

@property
def daq_window_timestamp(self):
    return self.__daq_window_timestamp

@property
def run_number(self):
    return self.__run_number

@property
def record_number(self):
    return self.__record_number

@property
def endpoint(self):
    return self.__endpoint

@property
def channel(self):
    return self.__channel

@property
def starting_tick(self):
    return self.__starting_tick
```

Inheritance

WaveformAdcs

```python
# Getters
@property
def time_step_ns(self):
    return self.__time_step_ns

@property
def adcs(self):
    return self.__adcs

@property
def time_offset(self):
    return self.__time_offset

@property
def analyses(self):
    return self.__analyses
```

WaveformSet

- A collection of Waveforms

```
# read all waveforms from a pickle file and save them in a WaveformSet
wfset =  reader.WaveformSet_from_pickle_file (file)

# Loop over all waveforms
for wf in wfset.waveforms:
   print (wf.endpoint, wf.channel, len(wf.adcs), wf.adcs[0])
```

Analysing a Waveform

waffles/src/waffles/data_classes/

- A Waveform can be analyzed. This means for example finding the baseline, the amplitude and the integral

```python
class WfAnaResult(ORDict):
    """Stands for Waveform Analysis Result. This class
    inherits from the ORDict class, and it is intended
    to store the results of an analysis (i.e. a class
    which derives from WfAna) which has been performed
    over a certain Waveform. It adds nothing to ORDict,
    i.e. it is just a renaming of ORDict

    Attributes
    ----------
    This class has no attributes

    Methods
    ----------
    This class has no methods
    """

    pass
```

WfAnaResult

```python
# Getters
@property
def input_parameters(self):
    return self.__input_parameters

@property
def result(self):
    return self.__result
```

WfAna

WaveformAdcs

```python
# Getters
@property
def time_step_ns(self):
    return self.__time_step_ns

@property
def adcs(self):
    return self.__adcs

@property
def time_offset(self):
    return self.__time_offset

@property
def analyses(self):
    return self.__analyses
```

Collection of WfAna

Analysing a Waveform: examples

- This is done by calling the analize method of a WaveformSet (see slide 27)

PeakFindingWfAna —Inheritance→ BasicWfAna —Inheritance→ WfAna

```python
# Getters
@property
def baseline_limits(self):
    return self.__baseline_limits

@property
def int_ll(self):
    return self.__int_ll

@property
def int_ul(self):
    return self.__int_ul

@property
def amp_ll(self):
    return self.__amp_ll

@property
def amp_ul(self):
    return self.__amp_ul
```

```python
# Getters
@property
def input_parameters(self):
    return self.__input_parameters

@property
def result(self):
    return self.__result
```

In addition finds implements a peak-finding algorithm based on scipy.signal.find_peaks()

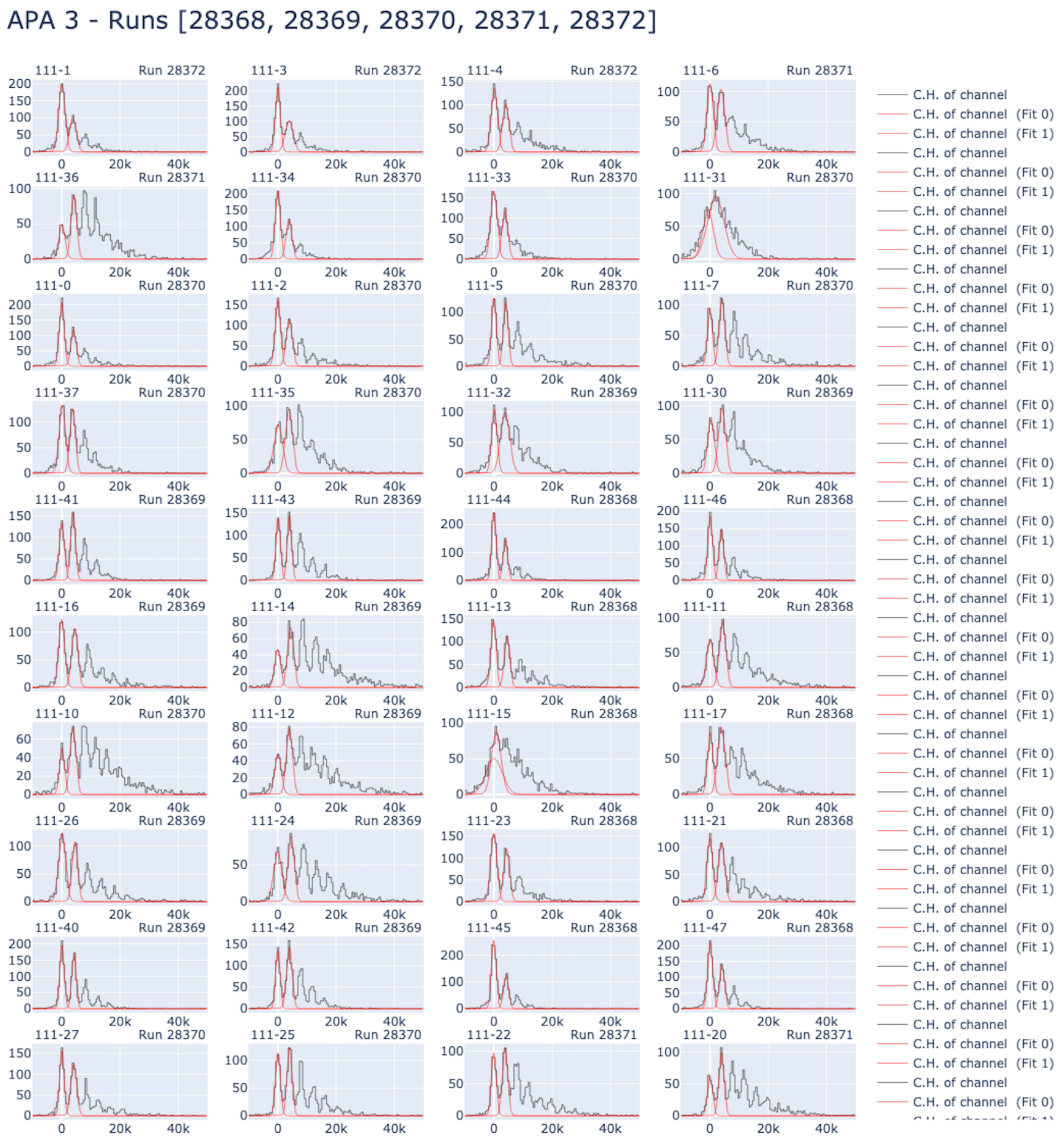
Calculates baseline, amplitud and integral

Detector definition

- We are working on a more general detector definition
- For the moment we use a 2D grid of WaveformSets, which represents an APA

```python
# Create a grid of WaveformSets for each channel in one
# APA, and compute the charge histogram for each channel
grid_apa = ChannelWsGrid(
    APA_map[self.params.apa],
    self.wfset,
    compute_calib_histo=True,
    bins_number=led_utils.get_nbins_for_charge_histo(
        self.params.pde,
        self.params.apa
    ),
    domain=np.array((-10000.0, 50000.0)),
    variable="integral",
    analysis_label=aux
)
```

```python
figure = plot_ChannelWsGrid(
    grid_apa,
    figure=None,
    share_x_scale=False,
    share_y_scale=False,
    mode="calibration",
    wfs_per_axes=None,
    analysis_label=aux,
    plot_peaks_fits=True,
    detailed_label=False,
    verbose=True
)
```



ChannelWsGrid

```python
# Getters
@property
def ch_map(self):
    return self.__ch_map

@property
def ch_wf_sets(self):
    return self.__ch_wf_sets
```

A collection

ChannelWs

Inheritance

WaveformSet

ChannelMap & Map

```python
# Getters
@property
def rows(self):
    return self.__rows

@property
def columns(self):
    return self.__columns

@property
def type(self):
    return self.__type

@property
def data(self):
    return self.__data
```

To be decoupled

```python
# Getters
@property
def endpoint(self):
    return self.__endpoint

@property
def channel(self):
    return self.__channel

@property
def calib_histo(self):
    return self.__calib_histo
```

```python
# Getters
@property
def waveforms(self):
    return self.__waveforms

@property
def points_per_wf(self):
    return self.__points_per_wf

@property
def runs(self):
    return self.__runs

@property
def record_numbers(self):
    return self.__record_numbers

@property
def available_channels(self):
    return self.__available_channels

@property
def mean_adcs(self):
    return self.__mean_adcs

@property
def mean_adcs_idcs(self):
    return self.__mean_adcs_idcs
```

Work in progress

Ongoing developments

- Define an Event class that allows clustering waveforms close in time
- Integrate beam information
- A better detector definition that can be used for any detector
- Ability to overwrite single parameters in the steering file
- Documentation exists in GitHub but needs to be updated

- We expect to have all this before Christmas